Time-Dependent Graph Generation: A Survey of Discrete-Time and Continuous-Time Perspectives

Quang Nguyen * , Muhammad Farhan , Asiri Wijesinghe

Posted Date: 14 July 2026

doi: 10.20944/preprints202607.1019.v1

Keywords: graph generation; graph generative models; diffusion models; flow matching; continuoustime generation; discrete-time generation; graph neural networks

 doi:10.20944/preprints202607.1019.v1

Article

Time-Dependent Graph Generation: A Survey of Discrete-Time and Continuous-Time Perspectives

Quang Nguyen [1,*], Muhammad Farhan [1] and Asiri Wijesinghe [2]

[1] School of Computing, The Australian National University, Canberra, ACT 2601, Australia
[2] Data61, CSIRO, Canberra, ACT 2601, Australia
* Correspondence: quang.nguyen@anu.edu.au

Abstract

Recent advances in graph generative modeling have increasingly reframed graph generation as the problem of learning probability paths that transport simple reference distributions toward complex graph distributions. This time-dependent perspective unifies a broad spectrum of generative paradigms, including finite-step denoising, masking, editing, and refinement methods, as well as continuous-time flows, score-based and diffusion models, Schrödinger bridge formulations, and continuous-time Markov processes over discrete graph states. In this survey, we present a unified temporal and transport-based framework for graph generation, organizing existing methods according to their time parameterization, state space, probability-path construction, evolution dynamics, training objectives, and sampling mechanisms. By systematically comparing discrete-time and continuous-time formulations, we reveal fundamental connections between seemingly disparate model families and highlight the design trade-offs governing scalability, structural fidelity, controllability, interpretability, and computational efficiency. We further review evaluation protocols, benchmark datasets, and key application domains, and identify emerging challenges in permutation symmetry, graph validity, large-scale generation, reproducibility, and fair empirical comparison.

Keywords: graph generation; graph generative models; diffusion models; flow matching; continuous-time generation; discrete-time generation; graph neural networks

CCS Concepts: **General and reference** → **Surveys and overviews**; **Computing methodologies** → **Machine learning**; **Neural networks**; **Unsupervised learning**; **Simulation evaluation**; **Artificial intelligence**

1. Introduction

Graph generation aims to learn distributions over graph-structured data and synthesize realistic graphs that preserve the complex structural, relational, and semantic properties observed in real-world networks. The problem lies at the core of numerous applications, including molecular design, drug discovery, materials science, knowledge graph completion, recommendation systems, and social-network analysis [38]. Unlike Euclidean domains such as images or text, graphs exhibit unique challenges arising from permutation symmetry, variable graph size, sparse and combinatorial dependencies, and domain-specific validity constraints. To address these challenges, modern graph generative modeling has evolved from static graph construction paradigms toward dynamic generation processes that progressively transform simple reference distributions into target graph distributions. This perspective has inspired a diverse range of methods, including autoregressive density estimation [100], variational autoencoders [56], adversarial learning [33], diffusion models [40], flow matching approaches [68], Schrödinger bridges, and continuous-time stochastic processes [23]. Despite their apparent differences, these methods share a common principle: graph generation is formulated as a trajectory through a sequence of intermediate states connecting a source distribution to a target graph distribution. This

 doi:10.20944/preprints202607.1019.v1

survey adopts this temporal and transport-based viewpoint as a unifying framework for understanding modern graph generative models, with a particular focus on the relationship between discrete-time and continuous-time formulations.

The foundations of graph generation predate modern deep learning and can be traced to classical random-graph and complex-network models, including Erdős–Rényi random graphs [27], Watts–Strogatz small-world networks [109], Barabási–Albert preferential attachment [3], and statistical descriptions of complex networks [80]. These models clarified network-formation mechanisms and global graph statistics, but their fixed assumptions limited their ability to represent the structural diversity and domain-specific constraints of real data. The rise of deep generative modeling shifted the field toward data-driven graph synthesis. Early neural generators explored sequential construction, latent-variable modeling, autoregressive factorization, adversarial learning, and block-wise generation as complementary strategies for learning graph distributions [6,63,64,96,115]. Although these methods generally did not formulate generation as an explicit temporal transport process, they established foundations of modern graph generation, including learned representations, iterative node and edge construction, neural graph decoding, and structured probabilistic factorization. These ideas later became important components of diffusion, flow-based, bridge, continuous-time Markov, and iterative-refinement frameworks.

Viewed through the lens of temporal evolution, many seemingly distinct graph generative models can be interpreted as alternative mechanisms for constructing probability paths between a simple reference distribution and a target graph distribution. Whether generation proceeds through graph construction, denoising, diffusion, transport, or jump dynamics, these approaches share a common objective: learning trajectories that transform distributions while preserving graph structure. This perspective reveals fundamental connections across model families and motivates a central question: *how should probability paths over graph spaces be designed and learned?*

Existing surveys have provided valuable overviews of graph generative modeling [28,38,121], but many predate recent advances in transport-based and continuous-time generation, while newer surveys often focus on diffusion models or specific applications such as molecular generation [15,69,105]. As a result, the relationships among diffusion models, flow matching, continuous normalizing flows, Schrödinger bridges, continuous-time Markov processes, and graph refinement methods remain fragmented. Moreover, evaluation remains challenging due to the sensitivity of results to datasets, metrics, sample sizes, and reporting protocols [97], making fair comparisons across model families difficult.

Motivated by these challenges, this survey presents a unified temporal and transport-based perspective on graph generation. We organize graph generative models through the framework of probability-path evolution, compare discrete-time and continuous-time formulations, review evaluation practices and applications, and highlight open challenges in scalability, structural validity, controllability, permutation symmetry, and reproducible evaluation. By treating temporal evolution as a common language for graph generation, we aim to provide a principled framework for understanding existing methods and guiding future research.

Definition and Scope. We use the term *time-dependent graph generative model* to describe methods that formulate graph generation as an explicit temporal evolution process, progressively transforming a simple reference distribution into a target graph distribution through intermediate graph states. These states may be discrete graphs, corrupted or partially observed graphs, continuous graph relaxations, or latent representations. From this perspective, graph generation can be viewed as the problem of constructing and learning probability paths over graph spaces.

We adopt the term *transport-based* in a broad sense, encompassing any mechanism that moves probability mass between source and target graph distributions. Such transport may be deterministic, stochastic, or jump-based, and may be realized through transition kernels, velocity fields, score functions, bridge processes, differential equations, or transition-rate operators. Under this view, diffusion models, flow-based methods, Schrödinger bridges, and continuous-time Markov processes can all be interpreted as alternative strategies for learning probability paths.

 doi:10.20944/preprints202607.1019.v1

Our survey covers both discrete-time and continuous-time graph generation. The former includes denoising, diffusion, masking, absorbing-state, editing, and refinement methods, while the latter includes ODE- and SDE-based models, continuous normalizing flows, flow matching, Schrödinger bridges, continuous-time Markov chains, and related hybrid approaches. We focus on methods in which temporal evolution plays a central role in generation, training, or sampling, and discuss classical VAE-, GAN-, and autoregressive-based graph generators primarily as historical precursors unless they explicitly incorporate time-evolution mechanisms.

Why Time Parameterization Matters. Time parameterization is a fundamental design choice in graph generative modeling because it determines how probability paths are represented and traversed during generation. Discrete-time methods model graph generation as a finite sequence of graph transformations, such as construction, masking, editing, or iterative denoising, providing interpretable intermediate states and facilitating stepwise control and constraint enforcement. In contrast, continuous-time methods view generation as the evolution of a dynamical system governed by velocity fields, differential equations, stochastic processes, or transition-rate operators, enabling flexible path design, endpoint coupling, geometry-aware transport, and uncertainty modeling. These choices influence the meaning of intermediate states, the preservation of graph structure, and the inductive biases of the generative process. Rather than viewing discrete-time and continuous-time approaches as competing paradigms, we treat time parameterization as a complementary design dimension that shapes how probability paths over graph spaces are constructed and learned.

Contributions. The main contributions of this survey are as follows:

- We review graph generative models whose generation, training, or sampling procedure is organized around an explicit temporal process.
- We analyze discrete-time and continuous-time formulations not only as different time parameterizations, but as different choices of probability path, state space, and evolution law.
- We connect denoising, diffusion, flow matching, jump processes, bridge methods, and geometry-aware transport under a shared graph-generation template.
- We summarize evaluation protocols, benchmark datasets, representative applications, and open challenges, with particular emphasis on reproducibility and fair comparison across model families.

Organization of the Survey. Figure 1 summarizes the survey structure. We begin by introducing the necessary mathematical foundations and preliminaries in Section 2, followed by a unified framework and taxonomy for time-dependent graph generation in Section 3. We then review representative discrete-time and continuous-time graph generative models in Secs. 4 and 5, respectively. Section 6 provides a comparative analysis of these approaches across common design dimensions, including state spaces, temporal parameterizations, evolution dynamics, and probability-path construction. We next discuss evaluation methodologies, benchmark datasets, and reproducibility considerations in Section 7, followed by an overview of major application domains in Section 8. Finally, Section 9 outlines open challenges and future research directions, and Section 10 concludes the survey.

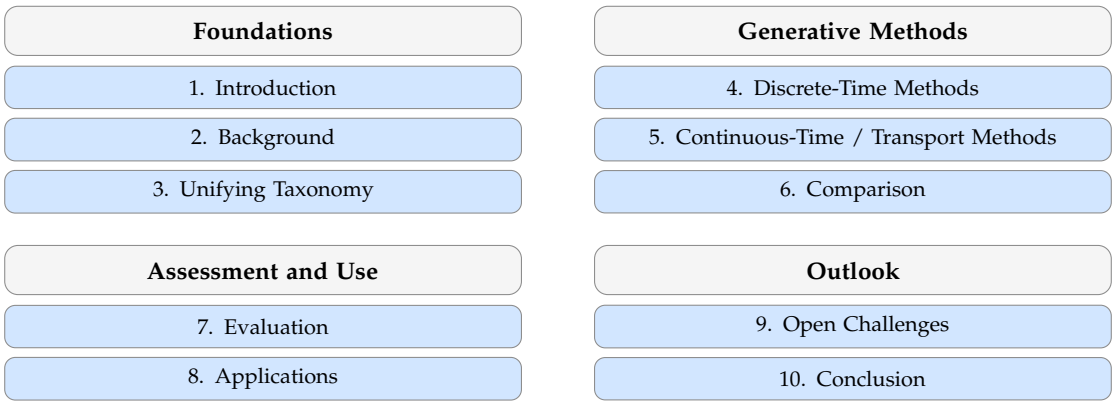

Figure 1. Organization of the survey.

 doi:10.20944/preprints202607.1019.v1

2. Background and Preliminaries

Problem Setting and Notation. We consider the graph generation problem, where the objective is to learn a probability distribution over graphs and generate new samples that exhibit the structural and statistical properties of observed graph data. Formally, let $\mathcal{D} = \{G^{(i)}\}_{i=1}^{N}$ denote a collection of training graphs drawn independently from an underlying data distribution p_{data}, i.e., $G^{(i)} \overset{\text{i.i.d.}}{\sim} p_{\text{data}}$. The goal is to learn a parameterized generative model p_θ that approximates p_{data} and enables the synthesis of realistic graph samples. A graph is represented as $G = (V, E, \mathbf{X})$, where $V = \{1, \dots, n\}$ is the node set, E is the edge set, and $\mathbf{X} \in \mathbb{R}^{n \times d}$ is an optional node-feature matrix. We use A to denote the corresponding adjacency matrix. Edge attributes or edge types can be incorporated analogously. Unless otherwise specified, the number of nodes n may vary across graphs, reflecting the variable-size nature of many real-world graph datasets [7,18,25,79].

A fundamental challenge in graph generation is permutation symmetry. For unlabeled graphs, node indices serve only as representational artifacts rather than semantic identities. Consequently, $(A, \mathbf{X})$ and $(PAP^\top, P\mathbf{X})$ represent the same graph for any permutation matrix P. Graph generative models must therefore respect this invariance and define distributions that are independent of node ordering. In practice, this is commonly achieved through permutation-equivariant architectures together with invariant likelihoods, graph readout functions, or decoding mechanisms [10,32,77, 111]. This requirement plays a central role in the design of modern graph generative models and distinguishes graph generation from generative modeling in Euclidean domains such as images and text. Throughout this survey, $\mathbf{X}$ is reserved exclusively for the node-feature matrix. We use x to denote a clean graph representation, e.g., $x = (A, \mathbf{X})$, with $x \sim p_{\text{data}}$, and use $\mathcal{S}$ to denote the state space in which a time-dependent generative process evolves. Depending on the modeling framework, $\mathcal{S}$ may correspond to a discrete graph space, a space of partially corrupted or masked graphs, a continuous relaxation of graph variables, a spectral or belief representation, or a learned latent space. This flexibility in state-space design strongly influences the dynamics, scalability, and structural properties of the generation process.

Time-dependent graph generators introduce intermediate states $z_t \in \mathcal{S}$, which describe the evolving graph representation at time t. These states may be discrete graphs, continuous graph relaxations, or latent embeddings, depending on the chosen formulation. When the clean graph is the initial state, we identify $z_0 = x$. When describing local evolution, we use $z \in \mathcal{S}$ for the current state and $z' \in \mathcal{S}$ for a possible successor state. The marginal distribution of states at time t is denoted by p_t. In discrete-time models, evolution is typically governed by a transition kernel $K_t(z' \mid z)$, whereas continuous-time models specify local dynamics through velocity fields, drift or score functions, bridge processes, or transition-rate operators. Under this notation, time-dependent graph generation constructs a trajectory that transports a simple source distribution toward a target graph distribution through intermediate states.

Discrete-Time and Continuous-Time Formulations. Time-dependent graph generation can be organized according to how time is parameterized. Discrete-time models use a finite chain

$$z_0 \to z_1 \to \cdots \to z_T. \tag{1}$$

Depending on the convention, the chain may describe a forward corruption or construction process, or a learned reverse denoising or generative process. For a finite state space $\mathcal{S}$, a stochastic transition kernel $K_t(z' \mid z)$ induces the update

$$p_{t+1}(z') = \sum_{z \in \mathcal{S}} K_t(z' \mid z) p_t(z),$$

with the sum replaced by an integral when $\mathcal{S}$ is continuous. Continuous-time models instead define a process indexed by $t \in [0, 1]$. The evolution may be deterministic, stochastic, or jump-based. In practice, continuous-time models are still simulated using finitely many solver or sampling steps. Thus,

 doi:10.20944/preprints202607.1019.v1

discrete-time and continuous-time methods are best viewed as complementary ways of describing temporal graph generation.

State Spaces. The state space $\mathcal{S}$ specifies the type of intermediate object z_t. Several choices are common in graph generation. First, the process may evolve directly in a discrete graph space, where each state is a graph or a partially corrupted graph. This makes graph edits and constraints easier to interpret. Second, the process may evolve in a continuous relaxation of adjacency matrices, node features, edge features, spectra, or categorical beliefs. This representation is convenient for ODE- and SDE-based models, but final discretization or decoding is usually required. Third, the process may evolve in a learned latent space, followed by a decoder that maps latent states back to graphs. Finally, hybrid models may combine these choices, for example by using discrete node or edge types together with continuous coordinates, beliefs, or latent variables.

The choice of state space affects validity, interpretability, scalability, and the ease of enforcing graph constraints. Discrete graph states are closer to combinatorial constraints, whereas relaxed and latent states are easier to optimize with continuous tools but often require additional mechanisms to recover valid discrete graphs.

Common Mathematical Tools. We provide the mathematical foundations for continuous-time graph generation, covering deterministic ODE flows, stochastic diffusion processes, discrete CTMC dynamics, score/denoising objectives, flow-matching training, and transport or bridge-based path construction.

ODE-based flows. A deterministic continuous-time generator can be written as

$$\frac{d}{dt}z_t = v_\theta(z_t, t), \tag{2}$$

where v_θ is a learned velocity field. If $z_0 \sim p_0$, solving the ODE transports samples from p_0 to later distributions p_t. When densities exist, the distribution evolves according to the continuity equation

$$\partial_t p_t(z) + \nabla_z \cdot \big(p_t(z) v_\theta(z, t)\big) = 0. \tag{3}$$

This is the basis of neural ODEs and continuous normalizing flows [16,34].

SDE-based diffusion. A stochastic continuous-time process can be written as

$$dz_t = f(z_t, t)\, dt + g(t)\, dW_t,$$

where f is a drift term, $g(t)$ controls the noise scale, and W_t is Brownian motion. In score-based generative modeling, the forward perturbation process is usually specified, while the reverse-time process depends on the score $\nabla_z \log p_t(z)$, which is approximated by a neural score model.

Discrete-state CTMCs. For discrete graph states, continuous-time evolution can be modeled by a continuous-time Markov chain (CTMC). Let $\mathcal{S}$ be a discrete graph-state space. A CTMC is defined by transition rates $Q_t(z, z')$ for $z \neq z'$, with

$$Q_t(z, z) = -\sum_{z' \neq z} Q_t(z, z'). \tag{4}$$

The marginal distribution follows the master equation

$$\frac{d}{dt}p_t(z) = \sum_{z' \neq z} p_t(z') Q_t(z', z) - p_t(z) \sum_{z' \neq z} Q_t(z, z'). \tag{5}$$

This formulation is useful when node labels, edge labels, or graph edits remain discrete throughout generation [13].

Score matching and denoising. Score matching provides the main training principle for SDE-based diffusion models. In the continuous-time formulation, the reverse-time SDE depends on the score

 doi:10.20944/preprints202607.1019.v1

$\nabla_z \log p_t(z)$, which is generally unknown and must be approximated by a neural score model. A typical denoising score-matching loss is

$$\mathcal{L}_{\mathrm{DSM}}(\theta) = \mathbb{E}_{t,z_0,z_t}\left[\left\| s_\theta(z_t,t) - \nabla_{z_t} \log q_{t|0}(z_t \mid z_0)\right\|_2^2\right], \tag{6}$$

where $q_{t|0}$ is a perturbation kernel. Thus, denoising is the practical mechanism by which SDE-based generators learn the reverse dynamics. In graph generation, this principle appears directly in continuous or relaxed graph diffusion models and analogously in discrete denoising models through categorical reconstruction or denoising objectives.

Flow matching. Flow matching trains ODE-based generative flows without simulating the ODE during training. Instead, it directly regresses the model velocity to a target velocity along a prescribed probability path:

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{t\sim\mathcal{U}[0,1],\, z\sim p_t}\left[\|v_\theta(z,t) - u_t(z)\|_2^2\right], \tag{7}$$

where u_t is a target vector field. After training, samples are generated by integrating the learned ODE in Equation 2. Thus, flow matching can be viewed as an ODE-flow framework with a supervised velocity-matching objective and an explicitly chosen probability path [68]. Related straightening ideas appear in rectified flow [71].

Optimal transport and bridge viewpoints. Optimal transport provides tools for comparing and coupling source and target distributions [86,103]. Schrödinger bridge methods add a stochastic path perspective: they seek a process whose marginals connect a source distribution to a target distribution while staying close to reference dynamics [23,61]. In this survey, these ideas are useful for understanding path design, endpoint coupling, and transport-based graph generation.

3. A Unifying Framework for Time-Dependent Graph Generation

Unified Generative View. Despite their methodological diversity, most time-dependent graph generative models can be understood as defining a time-indexed family of states or probability distributions that connects a simple reference distribution to the target graph distribution. During generation, a sample evolves from the reference endpoint toward the data endpoint through discrete transitions, deterministic continuous dynamics, stochastic dynamics, jump processes, or iterative refinement. Training learns or parameterizes the corresponding transition rule, vector field, score, jump rate, or refinement operator, whereas sampling simulates or numerically solves the learned dynamics to produce a graph. This view is conceptual rather than chronological: the time representation, state space, probability path, evolution dynamics, training objective, sampling strategy, and graph-specific mechanisms are typically designed jointly.

Taxonomy Axes for Time-Dependent Graph Generation. Building on this unified view, we organize time-dependent graph generative models through a comparative taxonomy that emphasizes fundamental design choices rather than mutually exclusive model categories. As illustrated in Figure 2, the taxonomy is structured around seven dimensions: time representation, state space, probability-path construction, evolution dynamics, training objective, sampling behavior, and graph-specific design choices. The root of the figure also summarizes the common reference-to-target evolution described above, while the surrounding axes characterize the interacting choices used to instantiate that evolution. These dimensions provide a common basis for analyzing the methods reviewed in Sections 4 and 5, and form the foundation of the comparative discussion in Section 6.

These taxonomy axes should not be viewed as independent design choices of equal importance. In practice, the behavior of a graph generative model emerges from interactions among its state space, probability-path construction, evolution dynamics, constraint mechanisms, and sampling strategy. For example, methods operating directly in discrete graph spaces provide interpretable graph-edit operations and a natural interface for enforcing hard structural constraints, but often face challenges associated with combinatorial state spaces and computationally expensive sampling. In contrast,

 doi:10.20944/preprints202607.1019.v1

relaxed or latent-state approaches, which are common in continuous-time flow and diffusion models, enable smooth optimization, geometry-aware transport, and solver-controlled sampling, but typically require decoding, projection, or discretization to recover valid graph structures. Continuous-time Markov chain (CTMC) methods combine continuous time with discrete graph states, preserving graph semantics while often requiring approximate or factorized jump-process simulation. Understanding these interactions is essential for comparing model families and motivates the multidimensional analysis developed throughout this survey.

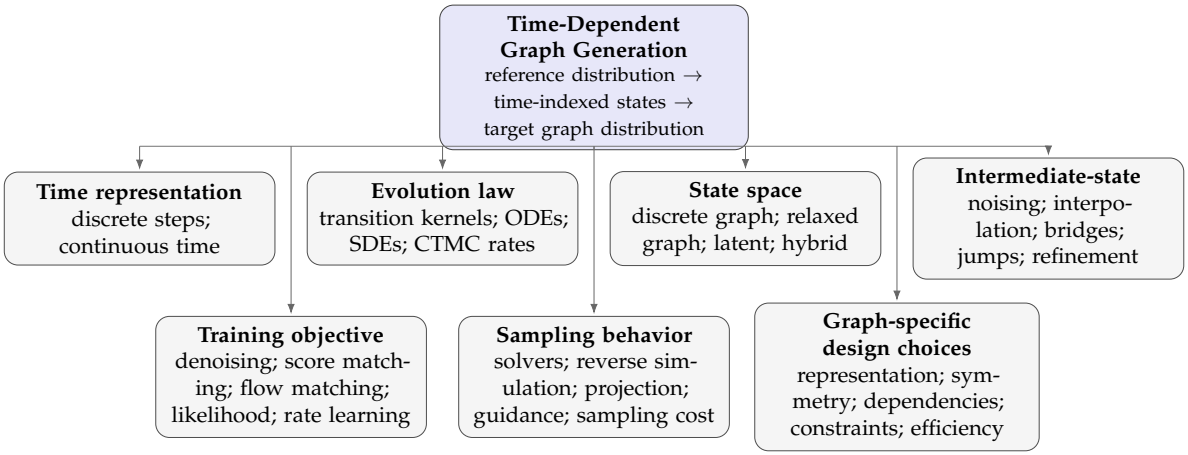

Figure 2. Main taxonomy axes for time-dependent graph generative models.

A Challenge-Oriented Reading Guide. While Figure 2 characterizes graph generative models through their underlying design choices, it does not directly explain how those choices address the fundamental challenges of graph generation. To complement the taxonomy, we adopt a challenge-oriented perspective that evaluates models according to the key difficulties they are designed to overcome. Table 1 summarizes six recurring challenges in graph generation, adapted from prior survey literature [38], and introduces the qualitative S/M/W ratings used throughout our comparative analyses. These ratings indicate the extent to which a model's design explicitly addresses a particular challenge and should be interpreted as design-level assessments rather than empirical performance measures. Consequently, they are intended to be read alongside the accompanying rationales and discussions for each method.

Table 1. Challenge-oriented reading guide for graph generative models. S, M, and W denote strong, moderate, and weak design-level support, respectively.

Challenge	Core question	Evidence of stronger support
Representation ambiguity	Does the model handle equivalent node permutations or construction orders?	Permutation-invariant or permutation-equivariant distributions, order marginalization, canonicalization, or graph-space updates.
Complex dependencies	Does the model capture coupled node, edge, attribute, and higher-order structure?	Joint node–edge prediction, message passing, graph transformers, autoregressive factorization, or graph-aware paths.
Large output spaces	Does the model reduce the combinatorial cost of graph search or sampling?	Sparse priors, factorized transitions, active-node selection, low-rank or spectral states, latent states, or efficient jump simulation.
Global / implicit properties	Does the model address global or indirect graph properties beyond local statistics?	Spectral or Laplacian representations, global readouts, property guidance, projectors, bridge constraints, or structure-aware transport.
Validity requirements	Does the model enforce domain-specific constraints such as valency, connectivity, acyclicity, or type consistency?	Restricted transitions, masks, constrained decoding, projection, rejection, or intrinsic validity by construction.
Interpretability / reliability	Can the model's generation process be inspected, diagnosed, or controlled?	Discrete graph edits, jump semantics, confidence or critic signals, calibrated rates, explicit constraints, or interpretable intermediate states.

Rating rule. S: explicit and central mechanism, preferably with a structural guarantee or direct control. **M**: partial support through architectural bias, learned representations, guidance, or empirical regularization. **W**: no direct mechanism, or support mainly through final decoding, thresholding, projection, or post-processing.

 doi:10.20944/preprints202607.1019.v1

4. Discrete-Time Graph Generation

Figure 3 summarizes the common template behind discrete-time graph generation. In the forward process, an observed graph state z_0 is transformed through a finite sequence of intermediate states z_t into a terminal state z_T by corruption, masking, editing, or partial construction. The learned reverse process starts from z_T and iteratively applies a time-conditioned transition model $\phi_\theta(z_t, t)$ to recover or generate a graph state z_0. This finite-step view covers discrete diffusion, absorbing or masking models, autoregressive constructions, and refinement-based generators.

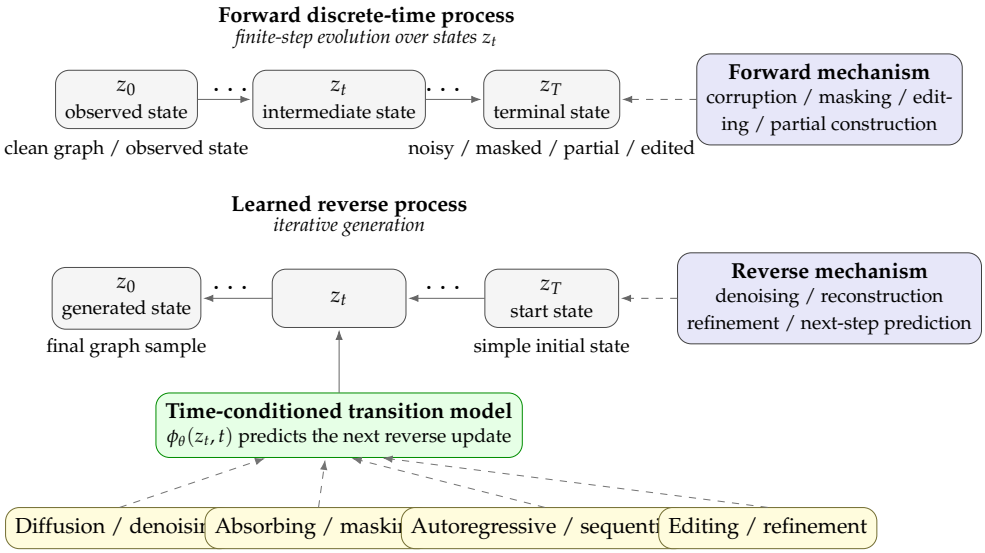

Figure 3. Generic architecture of discrete-time graph generation.

4.1. Discrete-Time Denoising and Diffusion Models

Discrete-time denoising and diffusion models constitute one of the most influential paradigms in modern graph generation. These methods formulate generation as a finite sequence of transformations that progressively corrupt, mask, edit, or otherwise modify graph representations, together with a learned reverse process that reconstructs graph structure step by step. Importantly, *discrete-time* refers to the temporal parameterization of the generative process rather than the nature of the state space itself. While some methods operate directly on discrete graph states, others evolve continuous adjacency matrices, bounded graph variables, spectral representations, categorical beliefs, or hybrid structured states.

Despite their diversity, most discrete-time methods share a common principle: they define a probability path through a sequence of intermediate graph states and learn a reverse process that progressively transports samples toward the target graph distribution. To highlight the major design choices within this family, we organize existing approaches into four broad categories: direct graph-variable denoising, transformed-representation denoising, hybrid absorbing or autoregressive generation, and iterative refinement. The first three categories are discussed in this section, as they primarily differ in their state-space representations, probability-path constructions, and factorization strategies. Iterative refinement methods are examined separately in Section 4.2, since their main distinction lies in the reverse sampling and correction mechanism rather than the choice of corrupted representation. Table 2 summarizes representative models, Figures 4 and 5 provide visual overviews of their underlying mechanisms.

 doi:10.20944/preprints202607.1019.v1

Table 2. Representative finite-step denoising and diffusion models for graph generation. Discrete-time refers to the temporal parameterization, not necessarily to a discrete state space.

Model	Space	Path	Key idea
GraphGUIDE	discrete edge space	Bernoulli edge kernels	interpretable edge editing and control
EDP-GNN	continuous adjacency	Gaussian noise levels	permutation-equivariant score model and Langevin sampling
DiGress	discrete attributed graph	categorical Markov	joint node–edge denoising
ConStruct	discrete constrained graph	edge absorbing	projector for edge-deletion-invariant constraints
EDGE	discrete sparse graph	edge removal	active-node and degree-guided generation
GBD	bounded graph variables	beta diffusion	bounded topology and attributes
GGSD	spectral graph variables	Gaussian spectral diffusion	Laplacian eigenpair denoising
GraphARM	discrete topology	node absorbing	learned autoregressive reverse order
LayerDAG	DAG layers	autoregressive + diffusion	layerwise DAG generation

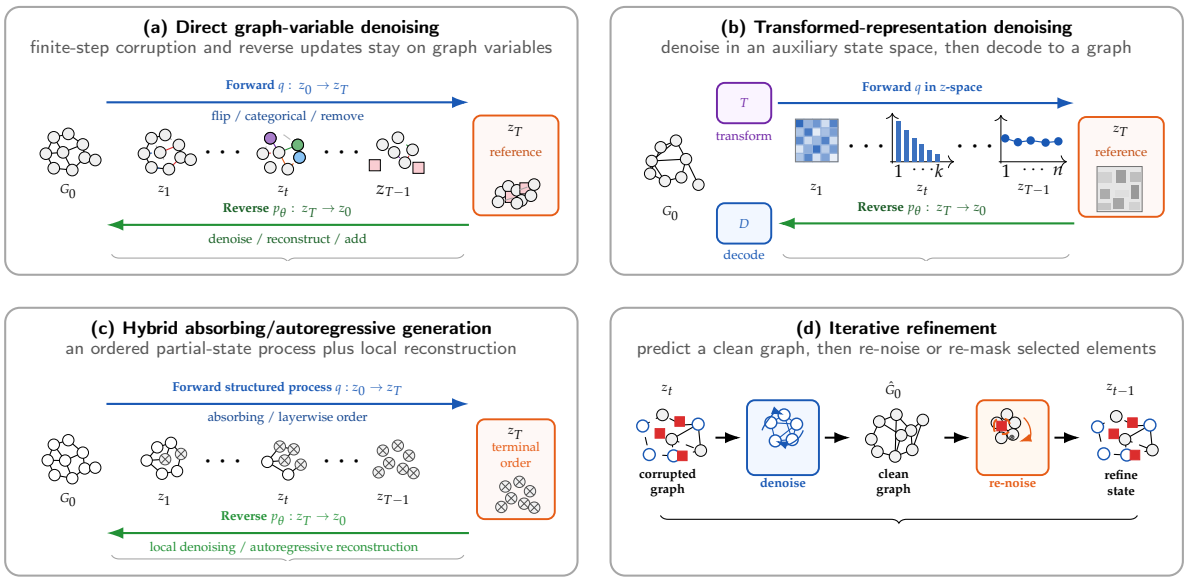

Figure 4. Common branch templates for discrete-time graph generation, arranged in a 2×2 layout. (a) Direct graph-variable denoising applies finite-step corruption and reverse denoising directly to graph-valued states. (b) Transformed-representation denoising maps graphs into an auxiliary state space, denoises there, and decodes back to graphs. (c) Hybrid absorbing/autoregressive methods combine a structured partial-state order with local reconstruction. (d) Iterative refinement repeatedly predicts a cleaner graph and then re-noises or re-masks selected elements, optionally using critic guidance.

Direct graph-variable diffusion and denoising. Direct graph-variable methods apply finite-step corruption and denoising to graph variables such as adjacency entries, node types, edge types, or masks. In fully discrete cases, each intermediate state is itself a graph, making the trajectory interpretable as edge flips, categorical relabeling, masking, deletion, or insertion. This also gives a natural interface for structural control during sampling. A boundary case is continuous adjacency-space denoising, where the process still follows finite noise levels but intermediate states are continuous adjacency variables and final discretization is required. This branch corresponds to the direct graph-variable template in Figure 4(a). Figure 5(a) illustrates the same idea through DiGress as a representative categorical node–edge denoising model, while GraphGUIDE, EDP-GNN, ConStruct, and EDGE are covered in the text and Table 2.

Binary edge-space denoising. GraphGUIDE [99] defines finite-step Bernoulli diffusion directly on binary edge variables. Its kernels flip edges, set edges to one, or set edges to zero, so every intermediate state is a well-defined graph. This makes the reverse process interpretable and allows manual control over desired or undesired edges and motifs during generation. Its main contribution is therefore not only denoising quality, but also the ability to intervene in graph-theoretic terms during sampling.

Continuous adjacency-space score denoising. EDP-GNN [82] is a boundary case within direct graph-variable denoising. It also operates on adjacency matrices, but its trajectory does not remain in discrete graph space. Instead, the model perturbs symmetric adjacency matrices with Gaussian noise, learns

 doi:10.20944/preprints202607.1019.v1

a permutation-equivariant score network, samples by annealed Langevin dynamics, and thresholds the final continuous adjacency matrix to recover a binary graph. Thus, EDP-GNN is better understood as continuous adjacency-space score-based denoising rather than binary edge-space diffusion.

Categorical node–edge diffusion. Earlier categorical generative models such as Argmax Flows and Multinomial Diffusion provide foundations for extending flows and diffusion from continuous variables to categorical state spaces [42]. Discrete diffusion over categorical variables is commonly built from structured transition matrices, including uniform, nearest-neighbor, and absorbing-state transitions [2]. DiGress [102] extends discrete denoising from binary edge existence to attributed discrete graphs. It diffuses node types and edge types with categorical Markov transitions, treating edge absence as a special edge category. The whole trajectory remains discrete, and the reverse problem becomes a joint node–edge classification task.

Constraint-aware and sparse variants. ConStruct [76] and EDGE [17] modify the direct discrete diffusion template for two different goals. ConStruct targets structural validity. It uses an edge-absorbing forward process and a projector in the reverse process so that generated graphs remain within a class of edge-deletion-invariant constraints, such as planarity or acyclicity. EDGE targets scalability. It sets the empty graph as the terminal reference distribution, treats the forward process as edge removal, and reverses it by predicting active nodes and adding edges only among selected nodes. It also uses degree guidance to better match degree-related graph statistics.

In sum, direct graph-variable denoising supports interpretable graph-space updates and gives a natural interface for constraints, but dense graph-state parameterizations can limit scalability and hard validity often still requires restricted transitions, projectors, or domain-specific correction.

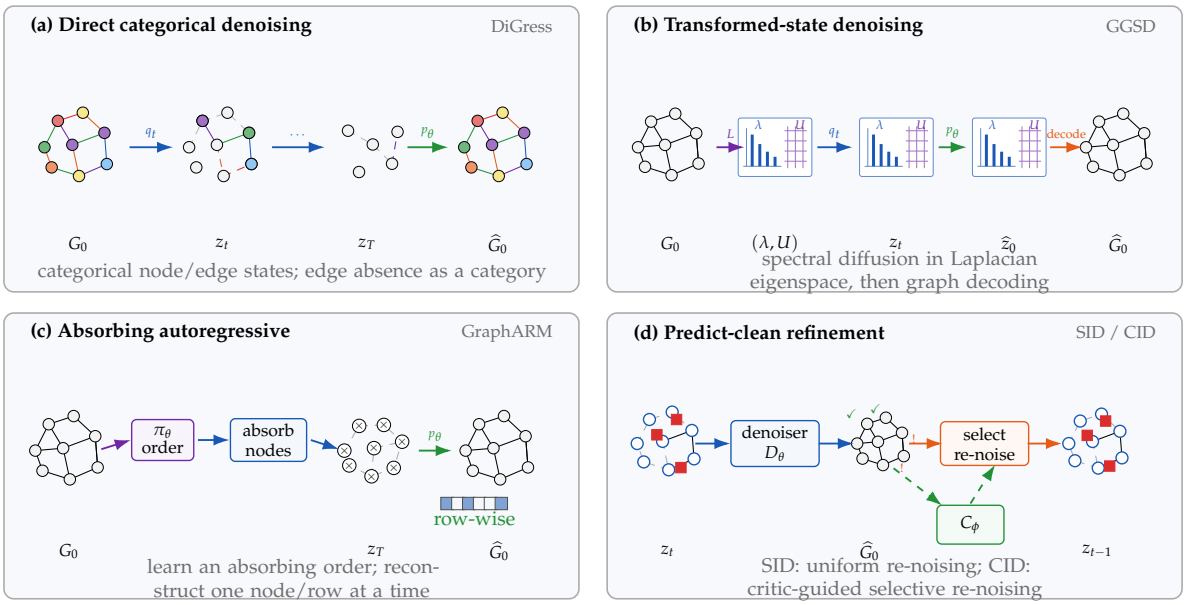

Figure 5. Representative discrete-time graph-generation mechanisms. (a) Direct categorical graph denoising, illustrated by DiGress. (b) Transformed-representation denoising, illustrated by GGSD. (c) Hybrid absorbing/autoregressive reconstruction, illustrated by GraphARM. (d) Iterative clean-prediction refinement, illustrated by SID with CID as an optional critic-guided extension.

Diffusion in transformed graph representations. A second branch keeps the discrete-time denoising template but changes the representation being diffused. Instead of keeping every intermediate state as a discrete graph, these models diffuse transformed graph variables, such as bounded continuous topology–attribute variables or spectral graph representations. This choice can make the trajectory smoother, better matched to graph statistics, or more sensitive to global structure. However, it also creates a reconstruction gap: the intermediate states are no longer ordinary graphs, and an additional decoding, thresholding, or graph-prediction step is needed to recover the final discrete graph object. This branch corresponds to the transformed-representation template in Figure 4(b). Figure 5(b) uses

 doi:10.20944/preprints202607.1019.v1

GGSD as a representative example because spectral diffusion makes the transform–denoise–decode structure explicit; GBD is retained in the text and Table 2.

Bounded continuous graph variables. GBD [72] uses multiplicative beta diffusion over bounded graph variables. Topology and node attributes are mapped into a continuous bounded space, corrupted by beta-distributed transitions, and denoised back toward clean bounded variables. Its concentration modulation assigns larger concentration values to important components, such as central edges or domain-relevant bonds, so that key structures are less aggressively corrupted and can re-emerge earlier during generation.

Spectral diffusion. GGSD [78] takes a more indirect route by performing denoising in a spectral representation of the graph Laplacian. Instead of noising the adjacency matrix directly, it applies diffusion to Laplacian eigenvalues and eigenvectors. The denoised spectral variables are then used to reconstruct Laplacian information, and a graph predictor outputs the final adjacency matrix. The main advantage of this representation is that spectral variables can encode global graph structure more directly than local edge variables. By truncating the spectrum, GGSD also reduces the cost of the diffusion stage compared with dense adjacency diffusion. However, the method introduces an additional decoding stage between the diffusion trajectory and the generated graph. Thus, transformed-representation diffusion broadens the discrete-time design space beyond direct graph edits, but it introduces a decoding gap: validity, constraints, and interpretable graph-space operations must be recovered through decoding, projection, or additional graph predictors.

Hybrid absorbing and autoregressive models. A third branch combines denoising-style refinement with a structured generation order. These models do not simply corrupt a full graph toward noise and reverse the process symmetrically. Instead, they embed denoising inside absorbing or autoregressive transition structures that impose a more explicit factorization of graph generation. This makes the finite-step trajectory more structured than standard full-graph denoising. This branch corresponds to the hybrid absorbing/autoregressive template in Figure 4(c). The common template consists of an outer structured process that defines an ordered sequence of partial states, together with a learned reverse process that performs local denoising or autoregressive reconstruction. Discrete diffusion over categorical variables can be built from absorbing-state transitions [2]. Figure 5(c) illustrates this branch through GraphARM, while LayerDAG is retained in the text and Table 2.

Absorbing-order reverse generation. GraphARM [57] uses a node-absorbing transition path together with autoregressive reverse reconstruction. In the forward process, one node is absorbed at each step by masking the node and its incident edges, until the graph reaches an empty masked state. A diffusion ordering network learns the node absorbing order from graph topology. In the reverse process, a denoising network reconstructs the graph in the opposite order by predicting one row of the adjacency matrix at a time. This places GraphARM between diffusion-style corruption/recovery and autoregressive graph construction. It still uses a finite-step corruption and recovery logic, but the number of steps is tied to the number of nodes, and the reverse process is organized around ordered node-wise prediction rather than full-graph denoising. This design is especially useful for unattributed graph topology generation, where a learned ordering can reduce the sensitivity of fixed-order autoregressive models.

Layerwise DAG generation with inner denoising. LayerDAG [62] is specialized to directed acyclic graphs. It uses the partial order of a DAG to define a layerwise decomposition, generating each new layer conditioned on previously generated layers. At each layer, the model predicts the number of new nodes, their attributes, and their incoming edges. Edge and attribute generation are handled with inner discrete diffusion, while the outer process is autoregressive over layers.

Hybrid absorbing and autoregressive models connect denoising with explicit graph construction orders, such as node-wise growth or directional layerwise generation. Their main limitation is specialization: they still require expressive denoisers and, outside the encoded structure such as acyclicity in LayerDAG, additional mechanisms for broader constraints. Before discussing iterative

 doi:10.20944/preprints202607.1019.v1

refinement methods, we summarize the challenge-oriented strengths and limitations of the discrete-time approaches reviewed thus far in Table 3.

Table 3. Challenge-oriented ratings matrix for representative discrete-time graph generation methods. Each entry rates how well a method addresses the corresponding challenge: W = weak support, M = moderate support, and S = strong support. Slash labels such as W/M or M/S indicate intermediate support levels. Full rationales are provided in Appendix Table A2.

Model	Representation ambiguity	Complex dependencies	Large output spaces	Global / implicit properties	Validity requirements	Interpretability / reliability
Direct graph-variable diffusion and denoising						
GraphGUIDE	W/M	M	W/M	M	M	S
EDP-GNN	S	M	W	W/M	W/M	W/M
DiGress	S	M/S	W/M	M	M	M
ConStruct	M	M	W/M	M/S	S	M/S
EDGE	M	M	S	M	W/M	M
Transformed graph-representation diffusion						
GBD	M	M/S	M	M	W/M	W/M
GGSD	M/S	M	M	S	W/M	W/M
Hybrid absorbing and autoregressive denoising						
GraphARM	M	M	M	M	M	W/M
LayerDAG	S	S	M	M/S	S	M
Iterative denoising and critic-guided refinement						
SID	S	M/S	M	M	M	M
CID	S	M/S	M	M	M/S	M/S

4.2. Iterative Denoising and Critic-Guided Refinement

Iterative denoising and refinement methods form the fourth discrete-time template introduced in Section 4.1. Unlike the previous three branches, they keep the finite-step graph-variable setting but reorganize the reverse update rule. Instead of following a standard Markov denoising chain in which each intermediate state depends directly on the previous noisy state, these methods repeatedly predict a clean graph from the current corrupted graph and then re-noise or re-mask selected elements. Thus, intermediate states are still graph variables, but the update rule is better understood as refinement rather than as a fixed diffusion reversal. This branch corresponds to the iterative-refinement template in Figure 4(d). Figure 5(d) compactly illustrates SID's predict-clean–then–re-noise mechanism and CID's critic-guided selective re-noising strategy.

Simple iterative denoising. Simple Iterative Denoising (SID) [4] assumes that intermediate noisy states are conditionally independent given the clean graph. During sampling, the model predicts a clean graph from the current noisy state and then samples a new intermediate state from a mixture of the predicted clean distribution and the noise distribution. This may reduce compounding denoising errors and can reuse pretrained discrete diffusion or flow-matching denoisers. In graph generation, the denoiser is typically permutation-equivariant and jointly predicts node and edge variables.

Critic-guided iterative denoising. Critical Iterative Denoising (CID) extends SID by making re-noising selective. Instead of re-noising all elements with the same probability, CID uses a critic to estimate the reliability of predicted components. Less reliable elements are re-noised more strongly, while high-confidence elements are retained. This makes refinement more targeted and may reduce unnecessary sampling steps. Thus, iterative denoising reframes sampling as adaptive error correction. Its benefits mainly come from the sampler rather than from a different graph state space, so permutation handling, validity, and structural fidelity still depend on the denoiser, critic, and any additional constraint mechanisms used during generation.

4.3. Summary of Discrete-Time Graph Generation

Discrete-time graph generators synthesize graphs through finite sequences of intermediate states, but differ in how those states are represented and updated. Direct graph-variable methods keep the process close to graph edits, giving interpretable transitions and a natural interface for constraints.

 doi:10.20944/preprints202607.1019.v1

Transformed-representation methods denoise bounded, spectral, or continuous variables, improving global-structure modeling but introducing a decoding gap. Hybrid absorbing/autoregressive models impose structured construction orders, while iterative refinement improves samples through repeated correction. Taken together, discrete-time methods are controllable, and interpretable, but still face scalability, global-structure, and hard-validity challenges.

5. Continuous-Time Graph Generation

Continuous-time graph generation formulates graph synthesis as the evolution of a stochastic or deterministic process over a continuous time horizon $t \in [0, 1]$. Rather than defining generation through a fixed sequence of transitions, these methods specify local dynamics via ordinary differential equations (ODEs), continuous normalizing flows (CNFs), stochastic differential equations (SDEs), continuous-time Markov chains (CTMCs), flow-matching objectives, or bridge processes, and generate samples by numerically simulating the resulting dynamics [24,26,53,54,89,114]. This perspective provides a flexible framework for constructing probability paths between a reference distribution and a target graph distribution, encompassing deterministic transport, stochastic diffusion, discrete-state jump processes, and endpoint-conditioned bridge formulations.

The key design choices in continuous-time graph generation concern the choice of state space, probability-path construction, evolution dynamics, and graph recovery mechanism. Depending on the formulation, the evolving state may correspond to relaxed graph variables, categorical probability representations, latent graph representations, or discrete node and edge states. Figure 6 illustrates the common conceptual template underlying these approaches, while Table 4 summarizes representative methods. Figures 7 and 8 further highlight the principal continuous-time paradigms. The former focuses on transport-based approaches, including ODE/CNF transport, flow matching, categorical flows, and graph-aware interpolation strategies, whereas the latter summarizes stochastic diffusion, projection-guided generation, discrete-state jump processes, and endpoint-conditioned bridge formulations. Together, these methods demonstrate the diverse ways in which continuous-time dynamics can be leveraged to model and generate complex graph distributions.

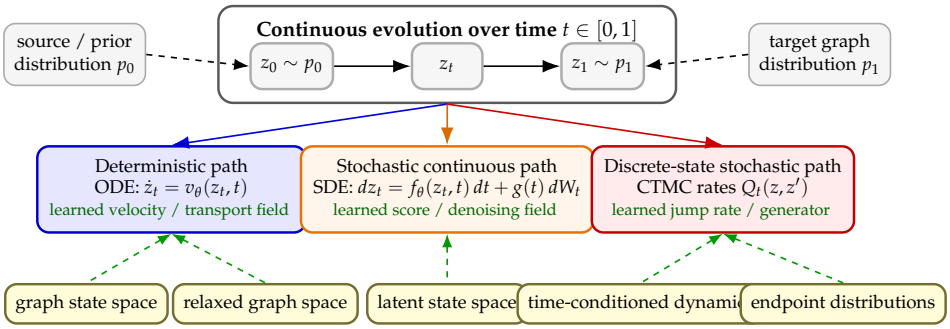

Figure 6. Generic architecture of continuous-time graph generation.

Table 4. Representative continuous-time graph generation models.

Model	Family	Space	Path / Dynamics	Key idea
CGF	ODE / CNF	relaxed graph	neural ODE transport	exact-likelihood continuous graph flow
ModFlow	ODE / CNF	relaxed graph	coupled node ODE	equivariant molecular flow
CatFlow	flow matching	relaxed graph	conditional interpolation	categorical flow matching
GGFlow	discrete flow matching	discrete graph	categorical interpolation + OT coupling	sparsity-aware molecular graph flow
DeFoG	discrete flow matching	discrete graph	clean prediction + CTMC sampling	decoupled training and sampling
BWFlow	geometry-aware transport	relaxed graph	Bures–Wasserstein path	graph-aware interpolation geometry
GGBall	geometry-aware transport	hyperbolic latent	geodesic interpolation	curvature-aware latent graph transport
PRODIGY	constrained sampling	relaxed graph	reverse diffusion + projection	plug-and-play constraint control
GraphBSI	SDE / belief-space	relaxed belief	stochastic belief evolution	categorical belief-space generation
DisCo	CTMC diffusion	discrete graph	continuous-time jump process	direct discrete-state graph diffusion
GruM	bridge mixture	relaxed graph	OU bridge mixture	graph-mixture prediction of terminal topology

 doi:10.20944/preprints202607.1019.v1

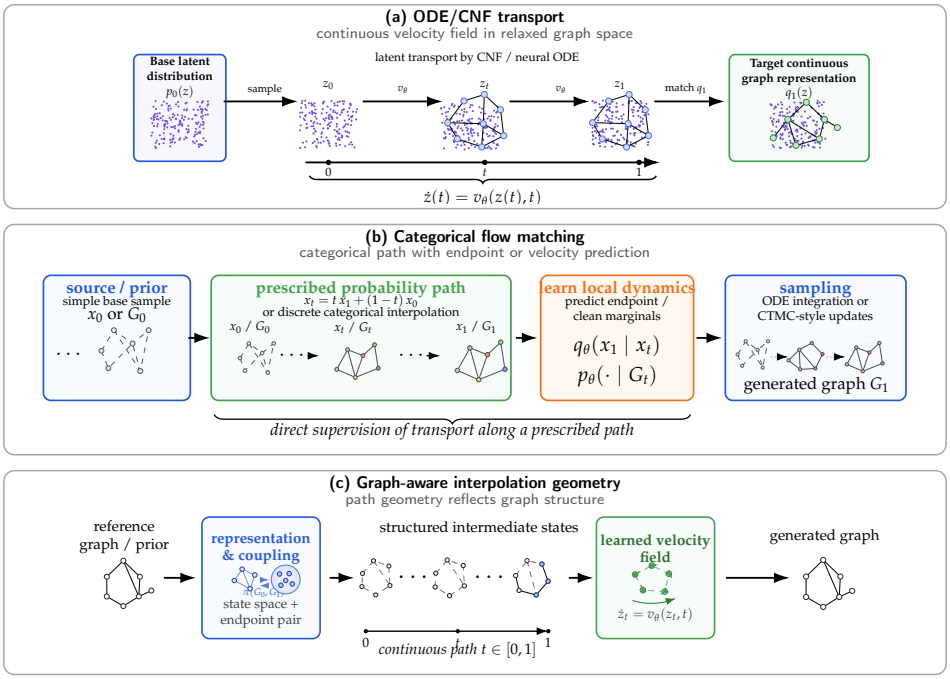

Figure 7. Continuous transport and path-design mechanisms. (a) ODE/CNF transport, illustrated by CGF, moves relaxed graph variables through a learned continuous velocity field and decodes them into a graph. (b) Categorical flow matching, illustrated by GGFlow, uses a prescribed categorical path, often with OT-informed endpoint coupling, and learns endpoint or velocity information along the path. (c) Graph-aware path design, illustrated by BWFlow, replaces component-wise interpolation with a structure-aware graph representation and interpolation geometry.

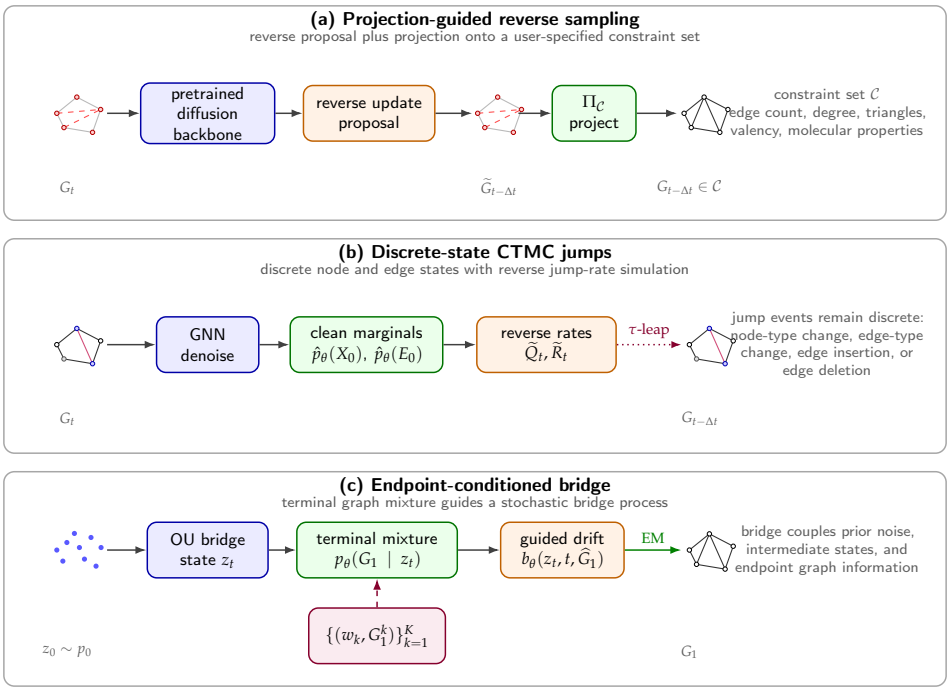

Figure 8. Stochastic, jump, and bridge-based continuous-time mechanisms. (a) Projection-guided sampling, illustrated by PRODIGY, modifies reverse diffusion/SDE updates by projecting proposed states onto a constraint set. (b) Discrete-state CTMC generation, illustrated by DisCo, predicts clean node–edge marginals and converts them into reverse jump rates over discrete graph states. (c) Endpoint-conditioned bridge generation, illustrated by GruM, uses a predicted terminal graph mixture to guide a stochastic bridge process toward plausible graph endpoints.

 doi:10.20944/preprints202607.1019.v1

5.1. Continuous-Time Transport and Flow-Based Models

Continuous-time transport and flow-based models generate graphs by moving samples from a simple reference distribution toward the data distribution through a learned velocity field or probability path. CNF methods learn invertible ODE dynamics and support likelihood evaluation. Flow-matching methods instead prescribe or construct a probability path and train the model to match local transport along that path. In graph domains, these methods usually operate in relaxed graph space, categorical probability space, or learned latent space, followed by decoding or discretization.

Continuous normalizing flows. Continuous normalizing flows (CNFs) define generation through an invertible ODE transport from a simple base distribution to a target data distribution. Their main advantage is that they provide a likelihood-based formulation through the continuous change-of-variables formula. Their main limitation is that the trajectory usually does not live directly in discrete graph space, so dequantization, decoding, thresholding, or categorical projection is needed to recover graph samples. Figure 7(a) summarizes this ODE/CNF transport mechanism through CGF as a representative example. ModFlow is retained in the text and Table 4 as a geometry-aware molecular CNF.

Graph-structured continuous flow. Continuous Graph Flow (CGF) [24] adapts CNFs to graph-structured variables by replacing fixed-depth message passing with continuous-time message passing. Starting from a simple base distribution, the model transports graph variables through a learned graph-structured velocity field. Each node or graph variable evolves according to messages from its neighbors, and the log-density is updated through the CNF trace term. Binary variables are handled through variational dequantization.

Geometry-aware molecular continuous flow. ModFlow [101] is a molecular CNF based on coupled node-wise ODEs. Each node is represented by a continuous atom-score vector, and the node scores co-evolve through an E(3)-equivariant graph neural differential operator conditioned on molecular structure and geometry. The final continuous scores are mapped to atom probabilities and decoded by an argmax step. Thus, ModFlow is best viewed as a geometry-aware molecular atom-score flow rather than a general topology generator. CNFs therefore provide a principled continuous-time transport and likelihood framework for graph-structured data, but relaxed trajectories usually need additional decoding or domain-specific mechanisms to recover valid discrete graphs.

Flow matching and conditional flow matching. Flow matching replaces likelihood-based CNF training with direct supervision of local transport along a prescribed probability path [68]. This separates path design from dynamics learning, making it natural to combine transport with graph-specific choices such as categorical endpoint prediction, sparse priors, optimal-transport couplings, and CTMC-style discrete sampling. Discrete flow matching extends this idea to categorical state spaces by defining probability paths between source and target distributions [30]. Figure 7(b) summarizes the categorical flow-matching mechanism through GGFlow as a representative example. CatFlow and DeFoG are retained in the text and Table 4 as related variants with different state spaces and sampling methods.

Relaxed categorical flow matching. CatFlow [26] performs variational flow matching in a relaxed categorical graph representation. Node and edge categories are represented continuously during transport and decoded at the end. The method uses a linear interpolant between source and target categorical representations and trains an endpoint posterior with a cross-entropy objective. The learned velocity is induced by the predicted endpoint distribution, giving a simplex-aware flow for categorical graph generation.

Sparse categorical graph flow matching. GGFlow [44] keeps node and edge variables categorical and transports a sparsity-aware prior toward molecular graphs. It defines categorical interpolation paths between source and target graphs and uses an optimal-transport coupling based on graph Hamming distance to improve source–target pairing. Its GraphEvo network uses edge-augmented transformer updates to model molecular bond interactions, and it also supports goal-guided generation.

 doi:10.20944/preprints202607.1019.v1

Discrete flow matching with CTMC-style sampling. DeFoG [89] is a discrete flow-matching framework whose sampling process is implemented with CTMC updates. It trains a clean node–edge marginal predictor from noisy graphs at arbitrary times, then constructs reverse rates during sampling. This decouples training from sampling and allows post-training control over time distortion, guidance, stochasticity, and sampling cost. Flow-matching-based graph generators make path design a central modeling choice. They improve control over transport paths, but scalability, hard validity, global structure, and interpretability still depend on the chosen state space, endpoint predictor, coupling, and sampler.

Geometry-aware transport and interpolants. Geometry-aware transport methods make the probability path itself graph-aware. Instead of interpolating node and edge variables independently in Euclidean space, they define paths in representation spaces whose geometry reflects graph structure. This can be done directly in structured graph representations, such as Laplacian or Graph Markov Random Field (GraphMRF) space, or in learned latent spaces, such as hyperbolic embeddings. In this sense, these methods can be viewed as transport methods with topology-aware path design. Figure 7(c) summarizes graph-aware path design through BWFlow as a representative example. GGBall is retained in the text and Table 4 as a hyperbolic latent-space variant.

Graph-space Bures–Wasserstein transport. BWFlow [48] modifies the graph-space probability path using a Graph Markov Random Field representation and Bures–Wasserstein interpolation. Node-feature means follow a simple interpolation, while graph structure is transported through a Laplacian-based Bures–Wasserstein path. This produces more coherent intermediate states than component-wise interpolation and directly targets the path-design problem in flow matching.

Hyperbolic latent-space transport. GGBall [11] performs Riemannian flow matching in a hyperbolic latent space. It first learns a hyperbolic graph autoencoder that maps graphs into curvature-aware latent codes, then trains a flow prior using geodesic interpolation in the Poincaré ball. This design is well suited to hierarchical, modular, or tree-like graph structure, but final validity depends on the encoder–decoder and codebook. Geometry-aware transport primarily improves the structure of the generative path rather than only the neural dynamics. It can strengthen global, hierarchical, or modular inductive bias, but usually adds representation assumptions, linear-algebra or manifold cost, and nontrivial decoding.

5.2. Projection-Guided and Belief-Space Stochastic Models

This subsection covers two uses of stochastic continuous-time ideas in graph generation. Projection-guided methods modify the sampler of an existing diffusion or SDE-style generator to enforce constraints, while belief-space methods evolve continuous uncertainty over categorical graph variables. Both families rely on a reverse-time simulator, but differ in whether their main contribution is sampling-time correction or belief-state evolution.

Boundary convention. EDP-GNN [82] also uses score-based denoising and annealed Langevin sampling over continuous adjacency variables. However, its generation is organized around a finite sequence of Gaussian noise levels, so we treat it as a discrete-time boundary case in Section 4.1 rather than as a continuous-time SDE model. For consistency, EDP-GNN should also be excluded from the continuous-time comparison table in Section 6 and retained with the discrete-time or boundary-case methods.

Figure 8(a) illustrates projection-guided reverse sampling through PRODIGY as a representative example of sampling-time constraint control. GraphBSI is retained in the text and Table 4 as a belief-space stochastic generation method.

Projection-guided sampling. PRODIGY [93] is a projection-guided sampling method for pre-trained graph diffusion models. It modifies reverse sampling by projecting intermediate samples onto a user-specified constraint set. The constraint set may encode edge count, degree, triangle count, valency, atom count, molecular weight, or molecular properties. Since PRODIGY does not train a new generator, its modeling capacity comes from the base diffusion model, while its contribution is explicit sampling-time constraint control.

 doi:10.20944/preprints202607.1019.v1

Belief-space stochastic generation. GraphBSI [85] evolves continuous logits that parameterize categorical beliefs over node and edge variables. Rather than directly sampling discrete graph states, it refines a belief distribution through a stochastic process derived from Bayesian Sample Inference. A reconstruction network predicts clean graph variables from the current belief, and the final logits are quantized into a discrete graph. The method also provides noise-controlled samplers that interpolate between deterministic probability-flow behavior and more stochastic sampling.

Projection-guided models improve controllability by modifying a sampler, while belief-space models change the evolving state to represent uncertainty over categorical graph variables. Both improve flexibility, but both still rely on a base model, reconstructor, projection design, or final decoding step for global validity.

5.3. CTMC and Discrete-State Continuous-Time Models

CTMC-based models are continuous in time but discrete in state. Instead of evolving relaxed adjacency matrices or latent vectors, the process evolves through jump events over categorical node and edge states. This is appealing for graph generation because node types, edge types, and edge absence can remain discrete throughout the trajectory, while the number of simulation steps can be adjusted at sampling time.

For discrete data, continuous-time denoising can be formulated as a CTMC whose forward and reverse processes are defined by jump rates [12,13]. This gives a continuous-time analogue of graph editing: infinitesimal events may correspond to node-type changes, edge-type changes, edge insertion, or edge deletion. Figure 8(b) illustrates the CTMC mechanism through DisCo, where clean node–edge marginals are converted into reverse jump rates over discrete graph states.

Discrete-state continuous-time graph diffusion. DisCo [114] is a discrete-state continuous-time diffusion model for graph generation. It represents graphs with categorical node types and edge types, treating edge absence as an additional edge category. The forward CTMC corruption process is factorized over nodes and edges using shared rate matrices and a time-dependent corruption schedule. The terminal reference distribution can be uniform or based on empirical node–edge marginals. The reverse process is also formulated as a CTMC. Instead of learning the full graph-level reverse rate matrix, DisCo predicts clean node and edge marginals from a noisy graph using a graph-to-graph neural backbone. These marginals are then used to construct reverse rates for node and edge updates. Training uses a cross-entropy denoising objective connected to reverse-rate approximation, and sampling uses reverse CTMC simulation with τ-leaping for a quality–efficiency trade-off. CTMC-based generation preserves discrete graph semantics while retaining continuous-time sampling flexibility. Its main trade-off is tractability: practical methods usually require factorized rates and approximate simulation.

5.4. Bridge-Based Formulations

Bridge-based formulations generate samples through stochastic processes conditioned on endpoint information. Rather than learning only a local denoising rule, score, or velocity, they emphasize the probability path connecting a reference distribution to the data distribution. This is useful for graph generation because terminal structure often determines whether a graph is valid or meaningful: small edge changes can alter connectivity, planarity, molecular validity, or other global properties. Figure 8(c) illustrates endpoint-conditioned bridge generation through GruM, where a predicted terminal graph mixture guides the stochastic bridge process.

OU-bridge mixture generation. GruM [53] is a bridge-mixture graph generator based on Ornstein–Uhlenbeck bridge processes. Instead of reversing a standard noising process, it constructs a mixture of endpoint-conditioned bridges whose endpoints lie in the data distribution. The sampler is guided by a predicted graph mixture, which can be interpreted as the model's estimate of plausible terminal graphs given the current state. GruM operates in relaxed continuous graph space. Node features and adjacency variables evolve continuously, and discrete graphs are recovered by quantization when

 doi:10.20944/preprints202607.1019.v1

needed. Training does not require simulating the bridge SDE, because intermediate states can be sampled analytically from tractable OU bridge marginals. The learned process is then sampled with Euler–Maruyama integration.

Bridge-based models make endpoint information central to graph generation. They can improve topology-aware guidance and structural coherence, but current graph bridge models still rely on relaxed states, neural terminal predictors, stochastic simulation, and final quantization. Table 5 consolidates the challenge-oriented assessment for representative continuous-time graph generation methods as a compact C1–C6 ratings matrix. Full rationales are provided in Appendix Table A3. The ratings are qualitative design-level annotations rather than empirical scores.

Table 5. Challenge-oriented ratings matrix for representative continuous-time graph generation methods. Each entry rates how well a method addresses the corresponding challenge: W = weak support, M = moderate support, and S = strong support. Slash labels such as W/M or M/S indicate intermediate support levels. Full rationales are provided in Appendix Table A3.

Model	Representation ambiguity	Complex dependencies	Large output spaces	Global / implicit properties	Validity requirements	Interpretability / reliability
ODE-based continuous normalizing flows						
CGF	M	S	M	W/M	W/M	W/M
ModFlow	M/S	S	M	M	M	W/M
Flow matching and categorical transport						
CatFlow	M/S	M	W/M	M	W/M	W/M
GGFlow	M/S	S	M/S	M	M	M
DeFoG	S	M/S	M/S	M	M	M
Geometry-aware transport and interpolants						
BWFlow	M	S	M	S	W/M	M
GGBall	M	M/S	M	S	W/M	W/M
Projection-guided and belief-space stochastic models						
PRODIGY	M	M	M	M/S	S	M
GraphBSI	M/S	M/S	M	M	W/M	M
Discrete-state CTMC graph generation						
DisCo	S	M/S	M/S	M	M	M
Bridge-based graph generation						
GruM	M	S	M/S	M/S	M	M

5.5. Permutation Consistency in Continuous-Time Transport

Remark. Permutation symmetry requires equivalent node orderings of an unlabeled graph $z = (A, \mathbf{X})$ to induce equivalent continuous-time trajectories. For ODE-, CNF-, and flow-matching-based generators, this first requires equivariant local dynamics [10,32,77,111]:

$$v_\theta(P \cdot z, t) = P \cdot v_\theta(z, t).$$

However, architectural equivariance alone is insufficient, because the probability path must also respect graph isomorphism [16,68]. For a deterministic interpolant, this requires

$$I_t(P \cdot z_0, P \cdot z_1) = P \cdot I_t(z_0, z_1),$$

with analogous pushforward conditions for stochastic or conditional paths. This issue is especially important in relaxed graph spaces, where interpolation between adjacency or feature tensors implicitly assumes node correspondence. Different labelings can therefore induce different intermediate states, transport directions, or velocity targets. Thus, endpoint couplings, probability-path design, intermediate representations, and graph recovery should all be checked for compatibility with permutation symmetry.

 doi:10.20944/preprints202607.1019.v1

5.6. Summary of Continuous-Time Graph Generation

Continuous-time graph generation replaces fixed transition chains with local dynamics over $t \in [0,1]$, including deterministic transport, stochastic diffusion, CTMC jumps, flow matching, and bridge-based paths. These methods offer flexible path design and solver-controlled sampling, but their behavior is strongly shaped by the chosen state space. Relaxed, latent, spectral, and belief-space formulations support smooth optimization and geometry-aware transport, yet require decoding, projection, quantization, or post-processing to recover valid graphs. CTMC formulations preserve discrete graph semantics, but depend on tractable rate parameterizations and approximate jump simulation. Projection-guided and belief-space methods improve control and uncertainty, while bridge models use endpoint information to encourage structural coherence. The family is flexible but shifts difficulty from defining transitions to designing representations, paths, and recovery mechanisms.

6. Cross-Family Comparison and Synthesis

The preceding sections demonstrate that discrete-time and continuous-time graph generative models are not competing paradigms, but rather complementary approaches to constructing probability paths between a simple reference distribution and a target graph distribution. Although they differ in how time is represented and how trajectories are realized, both seek to model the progressive evolution of graph states through intermediate representations. Their primary distinctions lie in the choice of state space, probability-path construction, evolution dynamics, sampling strategy, and the extent to which graph structure is preserved throughout the generation process.

To facilitate a unified comparison, Tables A4 and A5 summarize representative continuous-time and discrete-time methods across three core design dimensions: evolution dynamics, state-space representation, and intermediate-state construction. These tables highlight the diverse ways in which graph generative models instantiate temporal evolution, while also revealing common underlying principles. Consistent with the taxonomy introduced in Sections 4 and 5, methods are classified primarily according to the organization of their generative process rather than the numerical nature of their variables. Consequently, a model may employ continuous states within a discrete-time framework, or preserve discrete graph states while evolving in continuous time, as in certain continuous-time Markov chain formulations. This perspective enables a more nuanced comparison of graph generative models and provides the foundation for the synthesis and trade-off analysis presented in the remainder of this section.

Boundary-case convention. EDP-GNN [82] is discussed with discrete-time direct graph-variable denoising because its generation procedure is organized around a finite sequence of Gaussian noise levels and annealed Langevin updates. If included in a cross-family summary, it should be marked as a finite-noise-level score-denoising boundary case rather than as a core continuous-time SDE method.

6.1. Comparison by Time Domain

One of the most fundamental distinctions among time-dependent graph generative models is how temporal evolution is parameterized. As summarized in Table A5, discrete-time methods model generation through a finite sequence of transitions, such as denoising updates, diffusion steps, masking or absorbing operations, autoregressive construction rules, or iterative refinement procedures. This formulation often endows intermediate states with a clear operational interpretation, enabling explicit graph edits, stepwise control, and straightforward incorporation of structural constraints [76,99,102].

In contrast, continuous-time methods define graph generation through local dynamics evolving over a continuous time interval, as summarized in Table A4. Depending on the formulation, these dynamics may be specified by ordinary differential equations (ODEs), continuous normalizing flows (CNFs), stochastic differential equations (SDEs), continuous-time Markov chains (CTMCs), flow-matching objectives, or bridge processes [24,26,53,54,89,114]. A key advantage of this framework is the separation between the learned dynamics and the numerical procedure used to simulate them, allowing flexible solver choices and adjustable sampling resolution at inference time. However, this

 doi:10.20944/preprints202607.1019.v1

flexibility may come at the cost of increased sensitivity to discretization error, numerical stability, and computational overhead.

Importantly, the time domain should not be conflated with the underlying state space. While discrete-time methods are often associated with discrete graph transformations and continuous-time methods with continuous dynamics, the two design dimensions are largely independent. Continuous-time models can preserve discrete graph states through CTMC-based formulations, whereas discrete-time methods may operate on relaxed or continuous graph representations, as exemplified by EDP-GNN, GBD, and GGSD. Consequently, the distinction between discrete-time and continuous-time generation is best understood as a difference in how probability paths are parameterized and traversed, rather than a strict separation based on the nature of the graph representation itself.

6.2. Comparison by Evolution Law

The evolution dynamics of a graph generative model define the local mechanism by which graph states are transformed along a probability path toward the target distribution. As shown in Table A5, discrete-time methods are predominantly based on transition kernels and reverse-denoising procedures, including Bernoulli and categorical denoising, masking and absorbing transitions, edge-removal refinement, node-reconstruction processes, and iterative clean-prediction strategies. These dynamics are closely tied to graph operations and are therefore often interpretable as explicit graph edits or graph-state refinements.

In contrast, continuous-time methods employ a broader range of evolution dynamics, reflecting the greater flexibility of continuous-time probability paths. Table A4 distinguishes deterministic transport mechanisms, such as ODE-based continuous normalizing flows and flow-matching models, from stochastic dynamics based on SDEs, projection-guided diffusion, belief-space drift–diffusion processes, and Schrödinger bridge formulations. A third class comprises jump-process dynamics, most notably CTMCs, which evolve graphs through discrete state transitions governed by transition-rate operators. These formulations offer complementary perspectives on graph generation: ODE and flow-matching methods emphasize smooth transport in relaxed or latent spaces, stochastic models capture uncertainty and iterative refinement, while CTMC-based approaches preserve discrete graph semantics throughout the generative process.

The choice of evolution dynamics therefore strongly influences both the interpretability and flexibility of a graph generative model. Discrete-time approaches typically provide a more direct graph-edit interpretation through explicit transition rules, whereas continuous-time approaches offer a richer dynamical framework based on velocity fields, score functions, stochastic processes, jump rates, bridge dynamics, and geometry-aware transport. This broader set of modeling tools enables more flexible probability-path design, but often comes with additional challenges in simulation, scalability, and graph recovery.

6.3. Cross-Family Trade-Offs: State Space, Sampling, and Constraints

The main trade-offs across time-dependent graph generators arise from the interaction between state space, sampling procedure, and constraint handling, rather than from the discrete-time versus continuous-time distinction alone. Methods that operate directly in graph space preserve graph semantics throughout generation and often provide interpretable operations, such as edge edits, node reconstruction, masking, absorbing transitions, or layerwise construction. This makes them attractive for stepwise control and constraint enforcement through restricted transitions, admissible graph edits, construction orders, or masked prediction spaces. Their main limitation is combinatorial complexity: the number of graph configurations and graph-edit operations grows rapidly with graph size, so practical models often rely on factorized updates, sparse transitions, active-node selection, blockwise generation, or approximate simulation.

Relaxed, spectral, belief-space, and latent-state methods make a different trade-off. By moving generation into continuous or auxiliary representations, they support smoother optimization, geometry-aware interpolation, hierarchical abstractions, global structural representations, and flexible probability

 doi:10.20944/preprints202607.1019.v1

paths. These properties are especially useful for ODE-, SDE-, flow-matching-, and bridge-based models. However, such state spaces usually extend beyond the set of valid discrete graphs, creating a graph-recovery gap. Final graph samples must therefore be obtained through thresholding, quantization, projection, constrained decoding, guidance, repair, or post-processing. Thus, these methods often improve transport flexibility and expressivity, but weaken direct graph-space fidelity.

Two discretization issues should be distinguished. Continuous-time models may incur numerical time-discretization error when ODEs, SDEs, CTMCs, or bridge processes are simulated with finitely many solver steps. Separately, relaxed or latent-state models may incur graph-discretization error when continuous outputs are converted into discrete graphs. The former is from approximating continuous dynamics whereas the latter is from leaving and re-entering graph space.

Training and sampling reflect the same distinction. Discrete-time models typically learn reverse transition kernels, denoisers, or clean-state predictors and sample through the same finite reverse chain. This gives predictable inference cost and stable sampling, but each step may require a full node–edge update. Continuous-time models decouple learned dynamics from the numerical sampler: ODE and CNF methods use numerical integration, SDE and bridge models use stochastic simulation, flow-matching models regress transport velocities, and CTMC models use jump-process simulation such as τ-leaping. This flexibility enables adjustable sampling resolution and post-training solver choices, but introduces solver sensitivity, numerical discretization error, and potentially high function-evaluation, projection, or jump-simulation cost.

CTMC-based methods occupy an intermediate position. They are continuous in time but discrete in state, so node and edge variables can remain categorical throughout generation. This avoids continuous graph relaxation and final thresholding, while also allowing constraints to be incorporated through restricted jump rates or transition support. However, exact graph-level rate matrices are intractable for realistic graphs, so practical CTMC methods usually require factorized rates, sparse jump support, and approximate simulation.

Structural validity guarantees. A central challenge in graph generation is ensuring that generated samples satisfy domain-specific structural constraints. It is useful to distinguish between *intrinsic* validity, where invalid states are excluded by construction; *extrinsic* validity, where projection, constrained decoding, rejection sampling, repair, or post-processing restores validity; and *implicit* validity, where the model learns valid structures from data and architectural bias without formal guarantees. Local constraints, such as node types, edge types, degree limits, or valency masks, can often be enforced through restricted transitions or constrained prediction spaces, whereas global constraints, such as connectivity, acyclicity, planarity, motif consistency, schema compliance, or chemical validity, usually require global state tracking, projection, structured decoding, or domain-specific repair. Discrete graph-space and CTMC methods are generally better positioned for intrinsic validity when valid moves can be characterized, while relaxed, spectral, belief-space, and latent-state methods usually rely on extrinsic or implicit mechanisms.

6.4. Comparison by Intermediate-State Construction

The construction of intermediate states is a defining characteristic of time-dependent graph generative models, as it determines how probability paths are formed between source and target distributions. As shown in Tables A4 and A5, a key distinction emerges between corruption-based and transport-based approaches. Most discrete-time diffusion and denoising models rely on predefined forward processes that progressively transform graph representations through prescribed perturbations, such as Bernoulli noising in GraphGUIDE, Gaussian adjacency perturbations in EDP-GNN, edge absorption in ConStruct, categorical noising in DiGress, edge removal in EDGE, beta-noising in GBD, spectral Gaussian perturbations in GGSD, and node absorption in GraphARM. In these methods, the forward trajectory is largely fixed, and learning is primarily devoted to approximating the reverse process that reconstructs the target graph distribution.

 doi:10.20944/preprints202607.1019.v1

In contrast, many continuous-time transport-based methods treat probability-path construction as a central modeling component rather than a predefined design choice. Approaches such as CatFlow, GGFlow, BWFlow, and GGBall construct intermediate states through interpolation, optimal-transport couplings, endpoint pairings, Bures–Wasserstein geometry, or hyperbolic geodesics, explicitly shaping the trajectory followed during generation. Consequently, the quality and geometry of the probability path become integral to the model design itself. Bridge-based approaches, such as GruM, introduce a complementary paradigm in which intermediate states are guided by endpoint-conditioned stochastic dynamics, encouraging trajectories that remain consistent with both source and target distributions.

This comparison highlights a broader distinction between the two families. Discrete-time methods typically emphasize learning a reverse process over a fixed path, whereas many continuous-time methods place greater emphasis on designing the path itself. As a result, intermediate-state construction evolves from a predefined corruption mechanism into a modeling choice that directly influences transport efficiency, structural preservation, controllability, and the geometry of graph generation.

6.5. Summary

The comparative analysis presented in Tables A4 and A5 highlights that time parameterization, state-space representation, probability-path construction, and evolution dynamics are distinct yet interconnected design dimensions. Consequently, the landscape of graph generative modeling cannot be reduced to a simple discrete-versus-continuous dichotomy. Discrete-time methods typically offer more interpretable intermediate states, explicit graph-edit semantics, and a natural framework for stepwise control and constraint enforcement. In contrast, continuous-time methods provide a richer set of dynamical tools, including velocity fields, score functions, jump processes, bridge dynamics, and geometry-aware transport mechanisms, enabling greater flexibility in path design and sampling.

More broadly, the comparison reveals that the strengths and limitations of graph generative models are often determined by the interaction between their temporal representation, state space, and constraint-handling mechanisms. Discrete graph-space methods are particularly attractive when interpretability, graph-space fidelity, and hard validity constraints are essential, whereas continuous-time transport methods are well suited to scenarios that benefit from smooth interpolation, adaptive sampling, stochastic refinement, or geometry-aware generation. Neither paradigm is universally superior; rather, each offers distinct advantages that are appropriate for different modeling objectives and application domains.

7. Evaluation Protocols and Experimental Practice

Evaluating graph generative models remains a challenging and evolving problem because generation quality is inherently multi-dimensional and often depends on the characteristics of the graph domain, representation, and intended application. Unlike conventional predictive tasks, no single metric can fully capture the fidelity, diversity, validity, structural realism, and utility of generated graphs. Moreover, meaningful evaluation frequently requires balancing statistical similarity to observed data with application-specific requirements, such as chemical validity in molecular generation, structural consistency in knowledge graphs, or topological properties in network modeling. These challenges are further compounded by differences in datasets, experimental protocols, sample sizes, and reporting practices, which can make comparisons across studies difficult and sometimes inconsistent.

To provide a structured overview of current practice, we organize this section around two complementary perspectives. First, we review the evaluation metrics and benchmark protocols commonly used to assess graph generative models across different domains. Second, we examine representative empirical studies to illustrate how methodological choices, datasets, and evaluation procedures influence the interpretation of experimental results. Together, these perspectives highlight both the progress made in graph-generation evaluation and the ongoing need for more standardized, reproducible, and application-aware assessment protocols.

 doi:10.20944/preprints202607.1019.v1

7.1. Evaluation Metrics

A useful evaluation metric for graph generative models should capture both *fidelity* to the reference distribution and *diversity* among generated samples, while remaining computationally efficient [97]. This view is consistent with the *expressivity*, *robustness*, and *efficiency* perspectives discussed in [84]. In practice, graph-generation metrics can be grouped into four families: descriptor-based distributional metrics, learned-feature metrics, classifier-based metrics, and intrinsic-quality metrics.

Descriptor-based distributional metrics. These metrics compute hand-crafted graph statistics for each graph and compare their empirical distributions between generated and reference sets. Common descriptors include degree distributions, clustering coefficients, orbit counts, and Laplacian spectra. Each score corresponds to a recognizable graph property, which makes these metrics interpretable, but the conclusions depend heavily on descriptor choice. Kernel distances such as Maximum Mean Discrepancy (MMD) [35] and Wasserstein distances can also be sensitive to kernel, bandwidth, and implementation choices [58].

Learned-feature metrics. These metrics first embed graphs into a learned representation space, usually with a graph neural network, and then compare generated and reference samples in that space. They can capture richer global and local dependencies than hand-crafted descriptors, but are less interpretable and may inherit biases from the encoder architecture or training procedure. Feature extractors may be random, contrastively trained, pretrained, or self-supervised, with comparisons based on MMD or Fréchet-style distances [58,95,97].

Classifier-based metrics. These metrics evaluate generation quality by training a discriminator to separate generated graphs from reference graphs. The classifier's accuracy, AUC, or likelihood-based objective is then used as a proxy for distributional discrepancy [58]. This approach can combine multiple descriptors or learned representations into a single bounded score, but it depends on discriminator capacity, sample size, and train–test protocol.

Intrinsic-quality metrics. These metrics assess the generated set without considering the reference distribution. Common examples include *validity*, *uniqueness*, and *novelty*: validity checks whether samples satisfy structural or domain-specific constraints, uniqueness measures the proportion of non-duplicate samples, and novelty measures the proportion not copied from the training set [38]. These metrics are especially common in molecular generation [53]. A sound evaluation protocol should therefore report complementary metrics rather than relying on a single score. For the illustrative comparison below, we further convert raw metric values into tolerance-adjusted ranks. This makes it easier to see whether different benchmarks and metric families produce consistent model orderings.

7.2. Illustrative Empirical Comparison on Synthetic and Molecular Benchmarks

To complement the methodological discussion above, we include an illustrative empirical comparison under a unified evaluation protocol. The purpose of this experiment is not to construct a leaderboard of graph generators, but to examine the consistency and complementarity of different benchmark signals. For synthetic graph benchmarks, we study whether descriptor-based MMDs, feature-space MMD, and classifier-based distributional tests lead to compatible conclusions when applied to the same generated graph samples. For molecular benchmarks, we focus on attribute-aware distributional metrics and intrinsic quality metrics.

The model set is selected to represent the methodological families discussed in Sections. 4–5. It covers several state spaces and evolution laws, including discrete graph variables, continuous or relaxed graph variables, continuous-time jump processes, and bridge-based stochastic dynamics.

Datasets. We evaluate on two synthetic graph benchmarks and two molecular benchmarks. The synthetic datasets are stochastic block models (SBM) and planar graphs. The molecular benchmarks are QM9 and ZINC. QM9 consists of small organic molecules whose graph representation contains categorical atom types as node attributes and categorical bond types as edge attributes [91]. ZINC

 doi:10.20944/preprints202607.1019.v1

provides a drug-like molecular benchmark and is commonly used to evaluate molecular graph generation under validity, uniqueness, novelty, and distributional-matching criteria [47].

Metrics and Rank Reporting. The empirical comparison uses complementary metric families. For synthetic graph benchmarks, we report descriptor-based distributional metrics, learned-feature distributional metrics, and classifier-based distributional metrics. For descriptor-based metrics, we use MMD on degree distributions, clustering coefficients, orbit counts, and spectral statistics.

For descriptor-based and feature-space MMDs, lower values indicate closer agreement between generated and reference graphs. For the classifier-based metric, we use PGS-JS, the Jensen–Shannon-distance version of PolyGraphScore [58]. PGS-JS fits a probabilistic discriminator on graph descriptors to distinguish generated graphs from reference graphs. The discriminator log-likelihood provides an empirical lower bound on the Jensen–Shannon divergence, and the square-root transformation yields a JS-distance-style separability score. The resulting score lies in $[0, 1]$, where values closer to 0 indicate that generated and reference graph sets are difficult to distinguish under the selected descriptor-based classifier test, while larger values indicate stronger separability. Thus, PGS-JS is ranked in the lower-is-better direction. The midpoint 0.5 should not be interpreted as the uncertainty point of the final PGS-JS score; classifier uncertainty corresponds to discriminator probabilities near 0.5, which maps to a small JS-distance estimate.

For molecular benchmarks, we use molecule-specific attributed-graph metrics and intrinsic-quality metrics. Atom-type MMD and bond-type MMD measure whether generated molecules match the categorical node and edge attribute distributions of the reference set. Feature-space MMD and PGS-JS provide additional distributional signals. We also report validity, uniqueness, and novelty: validity measures the proportion of chemically valid generated molecules, uniqueness measures the proportion of non-duplicate generated molecules, and novelty measures the proportion of generated molecules that do not appear in the training set. Lower values are better for MMD and PGS-JS, whereas higher values are better for validity, uniqueness, and novelty. Because the goal is to assess consistency rather than small numerical differences, Tables 7 and 8 report *tolerance-adjusted ranks* instead of raw metric values. For each dataset and metric column, models are sorted in the favorable direction: ascending for MMD and PGS-JS, and descending for validity, uniqueness, and novelty. Two raw scores are treated as tied when their ratio lies within a 1% tolerance, i.e., when one score is between 99% and 101% of the other. We use dense ranks, so all models in the best tolerance group receive rank 1, and the next distinct group receives rank 2. The mean-rank and top-rank-count columns are diagnostic summaries rather than formal leaderboard scores.

Model Selection. For the synthetic benchmarks, we evaluate GraphGUIDE, DiGress, ConStruct, EDP-GNN, DisCo, and GruM. This set covers binary edge-space denoising, categorical node–edge denoising, constraint-aware diffusion, continuous adjacency-space score denoising, discrete-state continuous-time jump generation, and bridge-based stochastic generation. For the molecular benchmarks, we report only models whose current benchmark implementations support attributed graphs. In particular, GraphGUIDE and EDP-GNN are omitted from QM9 and ZINC because the implementations used here are designed for featureless topology generation and do not directly support categorical atom and bond attributes[1]. Table 6 summarizes the dataset–model compatibility used in the reported experiments.

Table 6. Illustrative benchmark setup.

Category	Models / datasets	Notes
Synthetic datasets	SBM, Planar	Featureless structural graph benchmarks
Synthetic models	GraphGUIDE, DiGress, ConStruct, EDP-GNN, DisCo, GruM	Evaluated on featureless graphs
Molecular datasets	QM9, ZINC	Attributed atom–bond molecular graph benchmarks

[1] This is an implementation-level limitation rather than a claim that the underlying ideas cannot be extended to attributed molecular graphs.

 doi:10.20944/preprints202607.1019.v1

Table 6. *Cont.*

Category	Models / datasets	Notes
Molecular models	DiGress, ConStruct, DisCo, GruM	Attributed-graph support in the current benchmark
Omitted from molecular benchmarks	GraphGUIDE, EDP-GNN	Current implementations do not support atom/bond attributes

7.3. Results and Discussion

Tables 7 and 8 report tolerance-adjusted ranks. Rank 1 denotes the best tolerance group in each dataset–metric column. Bold entries indicate rank 1. The main observation is that model performance is domain-specific rather than uniformly ordered. On the synthetic benchmarks, the strongest behavior is concentrated around models whose generative process remains close to graph structure. ConStruct is particularly stable when descriptor-based MMDs and PGS-JS are considered together, which is consistent with its constraint-aware edge-absorbing formulation: when the target distribution is defined by structural properties such as planarity or community-like organization, preserving graph-space semantics during generation can be more beneficial than optimizing only a learned embedding-space similarity. DiGress remains highly competitive, especially on degree and feature-space signals, but its advantage is less uniform once classifier-based separability is included.

Table 7. Tolerance-adjusted rank comparison on synthetic graph benchmarks. Rank 1 is best. Ranks are computed within each dataset and metric column; raw scores within 1% are tied. MMD and PGS-JS are ranked in the lower-is-better direction.

Dataset	Model	Degree	Cluster	Orbit	Spectral	Feature	PGS-JS	Mean	Top-1
Planar	ConStruct	**1**	**1**	**1**	**1**	4	**1**	1.50	5/6
Planar	DiGress	2	3	3	3	**1**	4	2.67	1/6
Planar	DisCo	4	4	4	4	4	3	3.83	0/6
Planar	EDP-GNN	5	6	5	5	3	5	4.83	0/6
Planar	GraphGUIDE	6	5	6	6	2	5	5.00	0/6
Planar	GruM	3	2	2	2	3	2	2.33	0/6
SBM	ConStruct	2	**1**	**1**	**1**	4	**1**	1.67	4/6
SBM	DiGress	**1**	2	2	2	**1**	3	1.83	2/6
SBM	DisCo	3	3	3	3	4	2	3.00	0/6
SBM	EDP-GNN	5	6	6	6	5	6	5.67	0/6
SBM	GraphGUIDE	6	5	5	5	2	5	4.67	0/6
SBM	GruM	4	4	4	4	3	4	3.83	0/6

Table 8. Tolerance-adjusted rank comparison on molecular benchmarks. Rank 1 is best. Ranks are computed within each dataset and metric column; raw scores within 1% are tied. Atom-type MMD, bond-type MMD, feature-space MMD, and PGS-JS are lower-is-better; validity, uniqueness, and novelty are higher-is-better.

Dataset	Model	Atom MMD	Bond MMD	Validity	Unique-ness	Novelty	Feature MMD	PGS-JS	Mean rank	Top-1 cols.
QM9	ConStruct	2	**1**	3	**1**	**1**	**1**	2	1.57	4/7
QM9	DiGress	3	3	2	**1**	**1**	**1**	3	2.00	3/7
QM9	DisCo	4	2	4	**1**	**1**	**1**	**1**	2.00	4/7
QM9	GruM	**1**	**1**	**1**	**1**	**1**	2	4	1.57	5/7
ZINC	ConStruct	2	3	3	**1**	**1**	2	2	2.00	2/7
ZINC	DiGress	4	2	4	**1**	**1**	3	4	2.71	2/7
ZINC	DisCo	3	**1**	2	**1**	**1**	2	3	1.86	3/7
ZINC	GruM	**1**	4	**1**	**1**	**1**	**1**	**1**	1.43	6/7

The second observation is that the metric families do not collapse to a single ordering. Descriptor MMDs, feature-space MMD, and PGS-JS test different projections of the generated distribution. Descriptor MMDs compare specific structural summaries, feature-space MMD depends on the representation induced by the graph encoder, and PGS-JS measures how separable generated and reference graphs are under a descriptor-based probabilistic classifier. Therefore, disagreement across metrics should not be treated simply as evaluation noise. It is informative: it identifies which structural aspects are well matched and which remain distinguishable.

 doi:10.20944/preprints202607.1019.v1

On the molecular benchmarks, the dominant factors shift from pure topology to attributed graph validity and atom–bond distribution matching. GruM is the most consistently ranked molecular model, particularly on ZINC, suggesting that bridge-based endpoint conditioning and constrained molecular post-processing can be effective for producing valid and diverse molecular graphs. However, its weaker bond-type MMD on ZINC shows that high validity does not necessarily imply accurate bond-distribution matching. DisCo provides a complementary pattern: although it is not always the best model by mean rank, its discrete-state CTMC formulation is naturally aligned with categorical node and edge updates, which can help explain its stronger bond-type matching behavior.

A further molecular observation is that uniqueness and novelty are often saturated or nearly saturated. When these intrinsic-quality metrics are tied across models, they contribute less information about distributional fidelity. In such cases, atom-type MMD, bond-type MMD, feature-space MMD, and PGS-JS become more important for distinguishing whether generated molecules are not only valid and non-duplicated, but also close to the reference molecular distribution. Taken as a consistency analysis, the benchmark is best interpreted as a consistency analysis. A model provides stronger evidence of distributional fidelity when it performs well across several structurally different metric families. Conversely, a model that is strong under one metric family but weak under another reveals a specific evaluation gap, such as matching local degree statistics while remaining separable under a classifier-based descriptor test, or achieving high molecular validity while mismatching bond-type distributions. These results support the use of a multi-metric benchmark rather than a single scalar score for graph generation evaluation.

7.4. Summary of Recommendations

A sound evaluation protocol for graph generative models should combine multiple metric families rather than rely on a single score. Descriptor-based metrics, learned-feature metrics, classifier-based metrics, and intrinsic-quality metrics such as validity, uniqueness, and novelty capture different aspects of generation quality. When the goal is to examine benchmark consistency, raw scores should be accompanied by tolerance-adjusted ranks, because small numerical differences may not reflect meaningful differences in generation quality. Authors should also report dataset splits, preprocessing rules, sample budgets, random seeds, metric settings, uncertainty estimates when possible, and any filtering or correction applied to generated graphs.

For fair comparison across model families, evaluation should make clear whether results are copied from original papers, rerun from public code, or obtained under a unified benchmark. This is especially important for comparing discrete-time and continuous-time generators, since they may allocate computation differently between training, sampling, and post-processing. Raw metric values should still be provided in an appendix, benchmark artifact, or repository so that rank-based summaries remain reproducible.

8. Applications of Time-Dependent Graph Generation

Time-dependent graph generation is useful for constrained relational objects, especially in scientific design. In these settings, generation is used not only for unconditional sampling, but also for conditional design, optimization, completion, representation learning, and simulation.

Molecular Design, Molecular Optimization, and Structure-Based Drug Design. Molecules are atom–bond graphs with strict constraints, including valency, drug-likeness, novelty, synthesizability, geometric stability, and target-specific binding. Diffusion models are therefore widely used in de novo drug design because they can model complex molecular distributions while supporting conditioning on properties [1].

For 3D molecule and conformation generation, ConfGF, GeoDiff, and Torsional Diffusion respectively model molecular conformation gradients, E(3)-equivariant diffusion, and torsion-space diffusion, while EDM jointly generates atom features and coordinates in an E(3)-equivariant framework [43,51,94,113]. Recent methods further vary the architecture, state space, and probability path:

 doi:10.20944/preprints202607.1019.v1

EQGAT-diff studies E(3)-equivariant design choices; HierDiff uses coarse-to-fine fragment-to-atom diffusion; GEOLDM and Latent 3D Graph Diffusion move generation into latent spaces; NExT-Mol couples SELFIES-based molecular language modeling with 3D diffusion; and GOAT uses geometric optimal transport and flow matching for faster joint generation of atom features and coordinates [41,59,73,88,112,116]. It shows how molecular generation motivates better denoisers, state spaces, probability paths, and symmetry-aware transport.

Time-dependent models also support optimization, multimodal conditioning, representation learning, and controllable molecular design. MOOD performs score-based out-of-distribution molecular generation with property guidance; UniGEM unifies molecular generation and property prediction; LDMol and DiffMS condition generation on text and mass spectra; and MoleculeSDE and SubGDiff use SDE- or diffusion-based objectives for molecular representation learning and downstream prediction [5,14,29,60,70,117]. DiffSBDD, DiffBP, and TargetDiff adapt equivariant diffusion to protein-aware 3D ligand generation; DiffLinker completes linkers or missing fragments; DiffDock treats docking pose prediction as diffusion over ligand transformations; FLAG generates ligands fragment by fragment using structural motifs and pocket context; IRDIFF adds interaction-based retrieval guidance; VoxBind denoises voxel grids around pockets; and DECOMPOPT supports controllable tasks such as R-group optimization and scaffold hopping [19,36,45,46,66,87,92,118,120].

Protein, RNA, and Biomolecular Dynamics. Time-dependent generation methods are well suited to proteins because protein design involves spatial constraints, long-range residue dependencies, chirality, and symmetry. For backbone design, Genie, RFdiffusion, FoldFlow, Proteus, and Proteina use diffusion or flow-based generation to produce designable and controllable protein structures [8,31,67,104,108]. SMCDiff targets motif scaffolding, and DiffAb applies diffusion-based sequence–structure generation to antibody CDR design [74,98].

Other work models sequences, conformational ensembles, docking, and molecular motion. DPLM applies discrete diffusion language modeling to protein sequences; AlphaFlow combines AlphaFold-style prediction with flow matching for ensemble generation; CONFDIFF uses force-guided SE(3) diffusion for protein conformations; MDGEN models molecular dynamics trajectories; and energy-based flow matching connects flow-based generation with iterative refinement and energy minimization for docking and backbone generation [50,52,106,107,119]. Time-dependent generation is also entering RNA design: RNAFlow formulates protein-conditioned RNA sequence–structure design as a flow-matching problem that combines RNA inverse folding with pretrained protein–nucleic-acid structure prediction [83].

Materials, Periodic Structures, and Molecular Assembly. Materials generation introduces periodicity, lattice geometry, stability, symmetry, and modular building blocks. CDVAE incorporates periodic invariance and stability into diffusion-augmented variational crystal generation, while DiffCSP uses equivariant diffusion for crystal structure prediction from composition [49,110]. TGDMat extends this setting with text-guided joint diffusion over atom types, coordinates, and lattices, and MOFFlow applies Riemannian flow matching to metal–organic framework prediction by generating roto-translations of rigid building blocks together with lattice structure [22,55]. Molecular assembly provides a related geometric setting: AssembleFlow performs rigid flow matching with inertial frames, decomposing molecular motion into translation and rotation so that molecules remain internally rigid during generation [37].

Visual, Software, and Other Structured Graph Domains. Beyond scientific design, time-dependent graph generation is relevant to program graphs, social and information networks, traffic and infrastructure networks, and synthetic benchmark construction. Program source code can be represented by abstract syntax trees, control-flow graphs, data-flow graphs, program-dependence graphs, or API-usage graphs. Early structured code generators use grammar- or AST-based generation, Syntax-Directed VAE imposes grammar constraints for validity, and graph-based generative code models combine grammar-driven expansion with graph neural message passing over partially generated

 doi:10.20944/preprints202607.1019.v1

programs [9,21,75]. These works indicate broader opportunities for time-dependent graph generation beyond scientific design.

9. Open Challenges and Future Directions

Despite rapid progress, time-dependent graph generative models still face open challenges which are tightly coupled with the choice of graph state space, transport path, local dynamics, sampler, and domain-specific validity requirements.

Scalability. Scalability remains a central bottleneck for time-dependent graph generative models. Dense node–edge parameterizations, full-graph transformers, iterative denoising, numerical integration, and reverse jump simulation can be costly on large or highly attributed graphs. Earlier scalable generators such as GRAN and BiGG introduce blockwise generation, sparsity-aware factorization, and edge-set modeling [20,64]. Recent diffusion-oriented methods such as SparseDiff and SaGess further suggest that sparse edge subsets, subgraph sampling, and divide-and-conquer strategies are promising for time-dependent graph generation [65,90]. Future work should develop scalable transport and diffusion mechanisms that support sparse or local updates, and reduce the number of required integration or denoising steps while preserving global graph structure.

Validity, Constraints, and Global Structure. Many applications require hard constraints such as valency, type consistency, connectivity, acyclicity, planarity, periodicity, or logical rules. Yet many current methods enforce these constraints only indirectly through training data, architectural bias, post-processing, or rejection. A key direction is to embed validity into the generative process itself, using constrained state spaces, validity-preserving transitions, projector-guided sampling, structure-aware transport paths, or hybrid symbolic–neural mechanisms. For example, ConStruct and PRODIGY illustrate this direction in graph diffusion through constraint-preserving or projection-guided sampling [76,93]. Beyond explicit validity, models must better preserve global and implicit graph properties. Future work should incorporate graph-level critics, multiscale consistency losses, topology-aware paths, and structured transport spaces beyond raw graph variables.

Evaluation, Controllability, and Reliability. Evaluation is still weakly standardized. Studies differ in datasets, preprocessing, sample budgets, metrics, validity filters, and reporting conventions, making cross-paper comparison difficult. Recent analyses show that descriptor-based MMD metrics are sensitive to kernel choices, sample sizes, and selected graph statistics [84,97]; since MMD is a kernel two-sample statistic, its use should be grounded in the assumptions of kernel testing [35]. Future metrics should combine descriptor-based, learned-feature, classifier-based, and intrinsic measures such as validity, uniqueness, and novelty.

Reliability-sensitive applications also require better controllability and interpretability. Many generators remain difficult to diagnose or steer toward desired structural outcomes. Promising directions include conditional and goal-directed generation, controllable path distortion, uncertainty-aware sampling, and trajectory-level debugging tools.

Toward Unified and Application-Aware Frameworks. A broader direction is to move from family-specific improvements toward modular frameworks. Many methods differ mainly in how they instantiate time, state space, path construction, local dynamics, and sampling. This suggests frameworks where representation, coupling, transport geometry, validity mechanisms, and samplers can be varied systematically. Foundations from optimal transport, Schrödinger bridges, diffusion bridges, flow matching, and rectified flow provide useful tools for designing and comparing such paths [23,61,68,71,86,103]. Future progress should also be application-aware that impose different trade-offs among validity, diversity, scalability, controllability, and interpretability. The long-term goal is therefore not a single universal graph generator, but a clearer design language for matching temporal generative mechanisms to domain-specific graph requirements.

 doi:10.20944/preprints202607.1019.v1

10. Conclusion

This survey reviewed time-dependent graph generation through a temporal and transport-based lens. No single model family is uniformly preferable: discrete-state methods are attractive for validity, interpretability, and graph-edit semantics; flow matching offers a flexible transport-based alternative to diffusion but still depends on permutation-consistent paths, scalable representations, endpoint coupling, and explicit validity control; CTMC methods bridge continuous-time sampling and discrete graph states but require scalable rate approximations; and geometry-aware relaxed or latent methods are promising for global and hierarchical structure but must address decoding error and validity loss. Across these families, the central challenge is to balance validity, scalability, expressivity, and path quality.

The main unresolved problems are scalable graph transport, structural validity guarantees, permutation-consistent probability paths, discrete–continuous discretization error, principled path design, validity-aware continuous-time dynamics, scalable CTMC simulation, reliable evaluation, and theoretical guarantees linking dynamics to distributional fidelity and validity preservation. Progress on these questions would move the field from empirical graph generation toward principled, scalable, and validity-aware transport over structured combinatorial spaces.

Appendix A. Detailed Challenge-Oriented Assessment Tables

Tables A2 and A3 provide the full challenge-oriented assessment tables corresponding to the compact ratings matrices in Tables 3 and 5. Table A1 summarizes the notation used in these assessment tables.

Table A1. Notation used in the challenge-oriented assessment tables.

Notation	Meaning
C1	Representation ambiguity
C2	Complex dependencies
C3	Large output spaces
C4	Global or implicit graph properties
C5	Validity requirements
C6	Interpretability or reliability
S	Strong design-level support
M	Moderate design-level support
W	Weak design-level support
W/M, M/S	Intermediate support between adjacent rating levels

Table A2. Full challenge-oriented assessment of representative discrete-time graph generation methods. C1–C6 and S/M/W follow Table 1; slash labels such as W/M or M/S indicate intermediate support. This appendix table retains the rationales omitted from the compact main-body ratings matrix in Table 3.

Model	C1	C2	C3	C4	C5	C6	Main rationale
Direct graph-variable diffusion and denoising							
GraphGUIDE	W/M	M	W/M	M	M	S	Discrete edge states and interventions improve interpretability, but dense binary edge space and general validity remain limited.
EDP-GNN	S	M	W	W/M	W/M	W/M	Permutation-equivariant score modeling is strong, but Gaussian adjacency noise and thresholding weaken graph-space validity and interpretability.
DiGress	S	M/S	W/M	M	M	M	Joint categorical node–edge denoising preserves discrete graph variables, but dense edge-category prediction is costly and constraints are learned.
ConStruct	M	M	W/M	M/S	S	M/S	Edge-absorbing diffusion plus projection directly supports selected edge-deletion-invariant constraints, but projector design is property-specific.
EDGE	M	M	S	M	W/M	M	Empty-graph prior, active-node selection, and sparse edge addition target scalability, with degree guidance for structural statistics.
Transformed graph-representation diffusion							
GBD	M	M/S	M	M	W/M	W/M	Bounded beta diffusion fits mixed topology–attribute variables, but hard structural constraints are not intrinsic.
GGSD	M/S	M	M	S	W/M	W/M	Spectral variables encode global structure directly, but eigenvector ambiguity and graph decoding introduce a reconstruction gap.
Hybrid absorbing and autoregressive denoising							
GraphARM	M	M	M	M	M	W/M	Learned node absorption reduces fixed-order dependence, but generation still uses ordered adjacency reconstruction and scales with graph size.
LayerDAG	S	S	M	M/S	S	M	Layerwise DAG construction reduces topological-order ambiguity and enforces acyclicity, but the design is specialized to DAGs.
Iterative denoising and critic-guided refinement							
SID	S	M/S	M	M	M	M	Predict-clean–then–re-noise refinement may reduce error accumulation, while validity and structure still depend on the denoiser.
CID	S	M/S	M	M	M/S	M/S	Critic-guided selective re-noising provides a reliability signal, but hard structural guarantees still require additional mechanisms.

 doi:10.20944/preprints202607.1019.v1

Table A3. Full challenge-oriented assessment of representative continuous-time graph generation methods. C1–C6 and S/M/W follow Table 1; slash labels such as W/M or M/S indicate intermediate support. This appendix table retains the rationales omitted from the compact main-body ratings matrix in Table 5.

Model	C1	C2	C3	C4	C5	C6	Condensed justification
ODE-based continuous normalizing flows							
CGF	M	S	M	W/M	W/M	W/M	Graph-structured continuous message passing models local dependencies, but relaxed graph states require decoding and make validity and graph-edit interpretability indirect.
ModFlow	M/S	S	M	M	M	W/M	E(3)-equivariant coupled ODEs provide strong molecular and geometric inductive bias, but final atom-score decoding and chemical validity remain partly extrinsic.
Flow matching and categorical transport							
CatFlow	M/S	M	W/M	M	W/M	W/M	Relaxed categorical endpoint prediction gives a clear transport objective, but dense node–edge transport and final categorical decoding limit scalability and validity guarantees.
GGFlow	M/S	S	M/S	M	M	M	Sparse categorical paths, OT-informed endpoint pairing, and edge-augmented updates improve molecular transport, but hard chemical constraints are still not guaranteed by the path alone.
DeFoG	S	M/S	M/S	M	M	M	Discrete node–edge states and CTMC-style sampling preserve categorical graph semantics, while clean-marginal prediction and post-training sampler control improve flexibility.
Geometry-aware transport and interpolants							
BWFlow	M	S	M	S	W/M	M	GraphMRF and Laplacian Bures–Wasserstein paths directly improve graph-aware interpolation and global structure, but constraints remain mostly encouraged rather than enforced.
GGBall	M	M/S	M	S	W/M	W/M	Hyperbolic latent transport is well matched to hierarchical or modular structure, but final validity depends on the learned encoder–codebook–decoder pipeline.
Projection-guided and belief-space stochastic models							
PRODIGY	M	M	M	M/S	S	M	Sampling-time projection explicitly enforces selected constraints and improves controllability, but it inherits representation and dependency modeling from the base diffusion model.
GraphBSI	M/S	M/S	M	M	W/M	M	Continuous belief-space evolution refines categorical uncertainty over graph variables, but final graph validity still depends on reconstruction and quantization.
Discrete-state CTMC graph generation							
DisCo	S	M/S	M/S	M	M	M	CTMC jumps preserve discrete node and edge states and allow adjustable simulation, but practical rates are factorized and hard structural constraints require restricted jumps.
Bridge-based graph generation							
GruM	M	S	M/S	M/S	M	M	Endpoint-conditioned OU-bridge mixtures provide topology-aware terminal guidance, but relaxed states, stochastic simulation, and final quantization limit hard validity guarantees.

Appendix B. Detailed Cross-Family Taxonomy Tables

Tables A4 and A5 provide the detailed cross-family taxonomy tables referenced in Section 6.

Table A4. Representative continuous-time graph generative models organized by evolution law, state space, and intermediate-state construction. EDP-GNN is excluded because it is treated as a finite-noise-level discrete-time boundary case in Section 4.1.

Method	Evolution law	State space	Intermediate-state construction
CGF [24]	ODE-based continuous normalizing flow	Relaxed graph space	Continuous transport path induced by neural ODE dynamics
ModFlow [101]	ODE-based continuous normalizing flow	Relaxed graph space	Continuous transport path over coupled node trajectories
CatFlow [26]	ODE-based flow matching / variational velocity model	Relaxed categorical graph space	Coupling- or interpolant-based construction
GGFlow [44]	Discrete-state flow matching / conditional probability velocity	Discrete graph space	Coupling-based categorical interpolation with OT-informed endpoint pairing
DeFoG [89]	Discrete flow matching with CTMC-style sampling	Discrete graph space	Prescribed interpolation path with jump-process reverse denoising
BWFlow [48]	ODE-based flow matching	Relaxed or mixed graph space	Bures–Wasserstein interpolant-based construction
GGBall [11]	Riemannian ODE-based flow matching	Hyperbolic latent space	Geometry-aware geodesic interpolant
PRODIGY [93]	Reverse-time SDE with projection-guided correction	Relaxed graph space	Prescribed forward noising path with projection-corrected reverse updates

 doi:10.20944/preprints202607.1019.v1

Table A4. *Cont.*

Method	Evolution law	State space	Intermediate-state construction
GraphBSI [85]	Drift–diffusion process in belief space	Relaxed categorical belief space	Smooth belief-space interpolation induced by Bayesian updates
DisCo [114]	Factorized CTMC with learned reverse rates	Discrete categorical graph space	Jump-process path over discrete node and edge states
GruM [53]	Bridge SDE / OU-bridge mixture dynamics	Relaxed graph space	Bridge-based construction relative to prior dynamics and endpoint mixture

Table A5. Discrete-time graph generative models organized by evolution law, state space, and intermediate-state construction.

Method	Evolution law	State space	Intermediate-state construction
GraphGUIDE [99]	Bernoulli transition kernels / reverse denoising	Discrete graph space	Prescribed Bernoulli noising path
EDP-GNN [82]	Finite-noise-level score denoising / annealed Langevin sampling	Continuous relaxed adjacency space	Prescribed Gaussian noising path with final thresholding
ConStruct [76]	Transition kernels / reverse denoising with projector	Discrete graph space	Edge-absorbing noising path with constrained reverse refinement
EDP [39]	Transition kernels / reverse denoising	Discrete graph space	Prescribed discrete noising path to an Erdős–Rényi prior
DiGress [102]	Transition kernels / reverse denoising	Discrete attributed graph space	Prescribed discrete noising path with marginal-preserving transitions
EDGE [17]	Transition kernels / reverse denoising with active-node factorization	Discrete graph space	Edge-removal path to an empty-graph prior with degree-guided reverse refinement
GBD [72]	Reverse beta denoising transitions	Continuous bounded graph space	Prescribed multiplicative beta noising path with concentration modulation
GGSD [78]	Spectral reverse denoising transitions	Relaxed spectral graph space	Prescribed Gaussian noising path in Laplacian spectral space
GraphARM [57]	Absorbing diffusion / autoregressive reverse kernel	Discrete graph space	Node-absorbing path with reverse node-wise reconstruction
LayerDAG [62]	Autoregressive layer transitions with inner denoising kernels	Discrete DAG space	Layerwise transition path with inner discrete noising
SID / CID [4]	Predict-clean iterative refinement with optional critic-guided re-noising	Discrete attributed graph space	Clean prediction followed by uniform or selective re-noising

Appendix C. Mathematical Supplement for Time-Dependent Graph Generation

This appendix provides additional mathematical background for the time-dependent and transport-based graph generative models discussed throughout the survey, with a particular focus on probability-path construction, temporal dynamics, state-space representations, and evaluation principles. Rather than introducing new model families, it aims to unify the mathematical concepts underlying discrete-time and continuous-time graph generation within a common framework. Consistent with the survey's emphasis on rigorous and reproducible evaluation, we do not treat generated-sample visualizations or sampling trajectories as primary evidence of model quality, since they are often sensitive to graph layouts, example selection, random seeds, and differences in underlying state spaces. Instead, the illustrative benchmark in Section 7.2 is interpreted primarily through quantitative metrics, repeated-run behavior, robustness, and consistency across multiple evaluation criteria.

 doi:10.20944/preprints202607.1019.v1

Appendix C.1. Discrete-Time Markov Kernels and Graph Corruption Processes

A discrete-time time-dependent graph generator defines a finite sequence of states

$$z_0 \to z_1 \to \cdots \to z_T, \tag{A1}$$

where each z_t may be a graph, a partially masked graph, a relaxed graph variable, or a latent representation. A forward corruption, masking, or construction process is commonly written as

$$q(z_{1:T} \mid z_0) = \prod_{t=1}^{T} q_t(z_t \mid z_{t-1}), \tag{A2}$$

where q_t is a transition kernel. In a finite discrete state space $\mathcal{X}$, the marginal distribution evolves as

$$q_t(z_t) = \sum_{z_{t-1} \in \mathcal{X}} q_t(z_t \mid z_{t-1}) q_{t-1}(z_{t-1}). \tag{A3}$$

For continuous or relaxed graph variables, the summation is replaced by an integral over the corresponding state space.

The learned reverse process is usually parameterized as

$$p_\theta(z_{0:T}) = p_T(z_T) \prod_{t=1}^{T} p_\theta(z_{t-1} \mid z_t, t), \tag{A4}$$

where p_T is a simple reference distribution, such as a fully noisy, masked, empty, or marginal categorical graph distribution. In denoising models, the network may predict the previous state z_{t-1}, the clean graph z_0, or marginal node and edge categories. A generic denoising objective can be written as

$$\mathcal{L}_{\text{denoise}}(\theta) = \mathbb{E}_{t, z_0, z_t}\left[\ell\big(f_\theta(z_t, t), z_0\big)\right], \tag{A5}$$

where ℓ is chosen according to the state space, for example cross-entropy for categorical variables or squared error for continuous relaxations.

This formulation covers several discrete-time graph generation mechanisms. In a categorical diffusion model, q_t is a categorical transition kernel over node and edge types. In an absorbing model, q_t progressively moves graph variables into a mask or absorbing state. In an edge-removal model, the forward process deletes edges and the reverse process learns to add them back. In an iterative refinement model, the reverse update may repeatedly predict a clean graph and then re-noise uncertain components, rather than strictly following a conventional Markov denoising chain.

Appendix C.2. Continuous-Time Limits and CTMC Master Equations

For discrete graph states, continuous-time evolution can be described by a continuous-time Markov chain (CTMC). Let $\mathcal{X}$ denote a discrete graph state space. A CTMC is specified by time-dependent transition rates

$$Q_t(x, x') \geq 0, \qquad x' \neq x, \tag{A6}$$

with diagonal entries

$$Q_t(x, x) = -\sum_{x' \neq x} Q_t(x, x'). \tag{A7}$$

The marginal law p_t evolves according to the master equation

$$\frac{\mathrm{d}}{\mathrm{d}t} p_t(x) = \sum_{x' \neq x} p_t(x') Q_t(x', x) - p_t(x) \sum_{x' \neq x} Q_t(x, x'). \tag{A8}$$

 doi:10.20944/preprints202607.1019.v1

When $p_t(x) > 0$, the formal reverse-time rates have the form

$$\widetilde{Q}_t(x, x') = Q_t(x', x)\frac{p_t(x')}{p_t(x)}, \qquad x' \neq x. \tag{A9}$$

In graph generation, p_t is not known exactly. Practical CTMC-based models therefore approximate the reverse process using a neural predictor of clean node and edge marginals, a learned rate model, or another tractable parameterization. This makes CTMC-based graph generation attractive for discrete graphs because node and edge variables can remain categorical throughout sampling, while the sampling step size can still be adjusted after training.

For graph data, a full CTMC over all possible graphs is usually intractable because the number of states grows combinatorially with the number of nodes and edge categories. Many models therefore factorize the forward corruption process over nodes and edges, while using a graph neural network or graph transformer in the reverse direction to recover dependencies among graph variables.

Appendix C.3. ODE Flows, Continuity Equations, and CNF Likelihoods

Deterministic continuous-time transport models evolve samples by an ordinary differential equation

$$\frac{\mathrm{d}}{\mathrm{d}t}z_t = v_\theta(z_t, t), \qquad t \in [0, 1], \tag{A10}$$

where v_θ is a time-dependent velocity field. If $z_0 \sim p_0$, the ODE induces a family of marginal distributions $\{p_t\}_{t\in[0,1]}$ satisfying the continuity equation

$$\partial_t p_t(z) + \nabla_z \cdot (p_t(z)v_\theta(z, t)) = 0. \tag{A11}$$

For continuous normalizing flows, the log-density evolves according to the instantaneous change-of-variables formula

$$\frac{\mathrm{d}}{\mathrm{d}t}\log p_t(z_t) = -\mathrm{Tr}(\nabla_z v_\theta(z_t, t)). \tag{A12}$$

Thus, likelihood-based training requires evaluating or estimating the divergence of the velocity field. In graph generation, this formulation is most natural when z_t is a continuous graph relaxation or a latent graph representation, since exact ODE flows are not directly defined over unordered discrete graph states without additional relaxation, dequantization, or decoding mechanisms.

The deterministic nature of an ODE trajectory can be useful for efficient sampling and likelihood evaluation, but it also introduces a representation gap for discrete graphs. If the final output must be a binary adjacency matrix, a categorical attributed graph, or a chemically valid molecule, then the model must include a decoding, projection, or quantization step from the continuous state to the final graph object.

Appendix C.4. SDE Reverse Processes and Probability-Flow ODEs

Stochastic continuous-time diffusion models define a forward process

$$\mathrm{d}z_t = f(z_t, t)\,\mathrm{d}t + g(t)\,\mathrm{d}W_t, \tag{A13}$$

where f is the drift, g is the diffusion scale, and W_t is Brownian motion. Under standard regularity assumptions, the reverse-time process can be expressed using the score $\nabla_z \log p_t(z)$:

$$\mathrm{d}z_t = \Big[f(z_t, t) - g(t)^2\nabla_z \log p_t(z_t)\Big]\mathrm{d}t + g(t)\,\mathrm{d}\overline{W}_t, \tag{A14}$$

where $\overline{W}_t$ denotes reverse-time Brownian motion and the equation is interpreted backward in time. A neural score model $s_\theta(z_t, t)$ is trained to approximate $\nabla_z \log p_t(z_t)$.

 doi:10.20944/preprints202607.1019.v1

A common denoising score-matching objective has the form

$$\mathcal{L}_{\mathrm{DSM}}(\theta) = \mathbb{E}_{t,z_0,z_t}\left[\left\| s_\theta(z_t,t) - \nabla_{z_t}\log q_{t|0}(z_t \mid z_0)\right\|_2^2\right], \tag{A15}$$

where $q_{t|0}$ is the perturbation kernel from clean data to the noisy state at time t.

The same marginal distributions can also be generated by the probability-flow ODE

$$\frac{\mathrm{d}}{\mathrm{d}t}z_t = f(z_t,t) - \frac{1}{2}g(t)^2 s_\theta(z_t,t). \tag{A16}$$

This connection separates the stochastic reverse sampler from a deterministic ODE sampler. For graph generation, however, the state is often a continuous relaxation, a logit-space belief, or a latent variable. If the target output is a discrete graph, then projection, quantization, categorical decoding, or a discrete-state formulation is still required.

Appendix C.5. Flow Matching and Conditional Flow Matching

Flow matching learns a velocity field by matching local transport targets along a prescribed probability path. Given a source sample $z_0 \sim p_0$ and a target sample $z_1 \sim p_{\mathrm{data}}$, a simple conditional path may be written as

$$z_t = (1-t)z_0 + tz_1, \qquad t \in [0,1]. \tag{A17}$$

The corresponding conditional velocity is

$$u_t(z_t \mid z_0, z_1) = z_1 - z_0. \tag{A18}$$

A generic flow-matching objective is then

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{t,z_t}\left[\|v_\theta(z_t,t) - u_t(z_t)\|_2^2\right]. \tag{A19}$$

For graph generation, the main modeling question is not only how to learn v_θ, but also how to define a meaningful path. Different methods may use Euclidean interpolation, categorical interpolation, optimal-transport pairings, geometry-aware graph interpolation, endpoint prediction, or latent-space geodesic interpolation. For discrete graph states, flow-matching-style training may also be combined with CTMC-style sampling, where the model predicts clean categorical marginals and the sampler converts them into reverse jump rates.

Conditional flow matching can be written as a regression problem over conditional paths:

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \mathbb{E}_{t,z_0,z_1,z_t}\left[\|v_\theta(z_t,t) - u_t(z_t \mid z_0,z_1)\|_2^2\right]. \tag{A20}$$

This formulation is useful when the conditional path is easy to sample even if the marginal velocity field is not available in closed form.

Appendix C.6. Bridge-Based Formulations and Endpoint Conditioning

Bridge-based generative models emphasize stochastic processes conditioned on endpoint distributions. Let $\mathbb{R}$ be a reference path measure and let $\mathbb{P}$ be a learned path measure whose marginals satisfy

$$\mathbb{P}_0 = p_{\mathrm{ref}}, \qquad \mathbb{P}_1 = p_{\mathrm{data}}. \tag{A21}$$

A Schrödinger-bridge-type formulation can be written abstractly as

$$\min_{\mathbb{P}} \mathrm{KL}(\mathbb{P} \,\|\, \mathbb{R}) \quad \text{subject to} \quad \mathbb{P}_0 = p_{\mathrm{ref}}, \quad \mathbb{P}_1 = p_{\mathrm{data}}. \tag{A22}$$

 doi:10.20944/preprints202607.1019.v1

In graph generation, bridge-based models are useful because endpoint structure is often highly informative. A model may learn to predict a terminal graph, a clean graph, or a mixture over possible terminal graphs from an intermediate state. The learned bridge drift can then guide the stochastic process toward a graph-like endpoint. This viewpoint is especially relevant when local perturbations alone do not capture global graph constraints such as connectivity, community structure, planarity, or chemical validity.

For an Ornstein–Uhlenbeck bridge, intermediate states can often be sampled from tractable conditional marginals during training. This makes it possible to train a terminal-graph predictor without simulating the full bridge process at every optimization step. Sampling then simulates the learned bridge dynamics from the reference distribution to the data-like endpoint distribution.

Appendix D. Evaluation Metric Details

This section summarizes the main evaluation quantities used in graph generation benchmarks. The goal is to clarify what each class of metric measures and why multiple metrics are needed for a consistency-oriented benchmark.

Appendix D.1. Descriptor-Based MMD Metrics

Descriptor-based maximum mean discrepancy compares generated and reference graph samples through graph statistics. Let $\psi(G)$ be a graph descriptor, such as a degree histogram, clustering coefficient histogram, orbit-count vector, spectral statistic, or community-related statistic. Given reference samples $\{G_i\}_{i=1}^{m}$ and generated samples $\{\widehat{G}_j\}_{j=1}^{n}$, the squared MMD is

$$\begin{aligned} \mathrm{MMD}^2 = &\frac{1}{m^2}\sum_{i=1}^{m}\sum_{i'=1}^{m} k\big(\psi(G_i),\psi(G_{i'})\big) \\ &+ \frac{1}{n^2}\sum_{j=1}^{n}\sum_{j'=1}^{n} k\big(\psi(\widehat{G}_j),\psi(\widehat{G}_{j'})\big) \\ &- \frac{2}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} k\big(\psi(G_i),\psi(\widehat{G}_j)\big), \end{aligned} \tag{A23}$$

where k is a positive-definite kernel.

Descriptor-based MMD metrics are interpretable, but each descriptor captures only part of graph structure. Degree MMD emphasizes local connectivity, clustering MMD emphasizes triangle-related closure, orbit MMD captures small graphlet patterns, and spectral MMD reflects eigenvalue-based structural information. Disagreement among these metrics should therefore be expected rather than treated as an error.

Appendix D.2. Learned-Feature Distributional Metrics

Learned-feature metrics compare graph distributions after embedding graphs into a representation space. Let $h(G) \in \mathbb{R}^d$ denote the graph embedding produced by a graph encoder, and let $\mu_{\mathrm{ref}}, \Sigma_{\mathrm{ref}}$ and $\mu_{\mathrm{gen}}, \Sigma_{\mathrm{gen}}$ be the empirical means and covariances of $\{h(G) : G \in \mathcal{G}_{\mathrm{ref}}\}$ and $\{h(G) : G \in \mathcal{G}_{\mathrm{gen}}\}$, respectively. A common Gaussian approximation compares these embedded distributions by:

$$\mathrm{FD} = \|\mu_{\mathrm{ref}} - \mu_{\mathrm{gen}}\|_2^2 + \mathrm{Tr}\Big(\Sigma_{\mathrm{ref}} + \Sigma_{\mathrm{gen}} - 2(\Sigma_{\mathrm{ref}}\Sigma_{\mathrm{gen}})^{1/2}\Big). \tag{A24}$$

Here μ_{ref} and Σ_{ref} are the empirical mean and covariance of reference graph embeddings, and μ_{gen} and Σ_{gen} are the corresponding quantities for generated graph embeddings.

These metrics can capture richer structural patterns than hand-designed descriptors, but they depend on the encoder, its training data, and the relevance of the learned representation to the target graph domain. Therefore, learned-feature metrics should be reported together with descriptor-based and task-specific metrics rather than used as a single replacement for them.

 doi:10.20944/preprints202607.1019.v1

Appendix D.3. Classifier-Based Two-Sample Metrics

Classifier-based metrics train a discriminator to distinguish reference graphs from generated graphs. Let $D_\phi(G)$ denote a classifier output estimating the probability that G comes from the reference distribution. A standard binary classification objective is

$$\max_{\phi} \mathbb{E}_{G\sim p_{\text{ref}}}\big[\log D_\phi(G)\big] + \mathbb{E}_{\widehat{G}\sim p_\theta}\Big[\log\big(1 - D_\phi(\widehat{G})\big)\Big]. \tag{A25}$$

If the classifier performs near chance, the generated distribution is difficult to distinguish from the reference distribution under the classifier's feature class. If it performs very well, the generated and reference distributions are separable.

Such metrics are useful as flexible two-sample tests, but they depend strongly on classifier architecture, training protocol, sample size, and regularization. A classifier-based score should therefore be interpreted as a complementary distributional check rather than a definitive proof of sample quality.

Appendix D.4. Validity, Uniqueness, and Novelty

For domains with explicit constraints, graph generation is often evaluated using validity, uniqueness, and novelty:

$$\text{Validity} = \frac{\#\{\text{valid generated graphs}\}}{\#\{\text{generated graphs}\}}, \tag{A26}$$

$$\text{Uniqueness} = \frac{\#\{\text{unique valid generated graphs}\}}{\#\{\text{valid generated graphs}\}}, \tag{A27}$$

and

$$\text{Novelty} = \frac{\#\{\text{unique valid generated graphs not in training set}\}}{\#\{\text{unique valid generated graphs}\}}. \tag{A28}$$

These metrics are especially common in molecular graph generation. They should be interpreted together with distributional metrics because a model may generate valid graphs while still failing to match the target distribution. Conversely, a model may match some graph statistics while violating domain-specific constraints.

Appendix E. Benchmark Configuration and Reproducibility Details

This section specifies the experimental settings used in the illustrative benchmark.

Appendix E.1. Dataset and Sample Reporting

All datasets are prepared with random seed 42. For synthetic datasets, we generate 10,240 graphs and use an 80/10/10 train/validation/test split. Planar and SBM both use fixed-size graphs with 64 nodes. The SBM generator uses 4 blocks with within-block probability $p_{\text{in}} = 0.25$ and cross-block probability $p_{\text{out}} = 0.02$. For molecular datasets, QM9 is loaded from the PyG preprocessed archive, capped at 12,000 shuffled molecules, and split 80/10/10. ZINC uses the PyG ZINC subset with the official split, giving 10,000 training molecules, 1,000 validation molecules, and 1,000 test molecules.

Each model–dataset pair is trained and evaluated over three independent runs. For each run, the sampler generates 1,024 graphs and compares them against 1,024 reference graphs. The main benchmark tables report the average value over the three runs, while the supplementary tables additionally report the corresponding standard deviation. Descriptor metrics use the test split as reference, learned-feature MMD uses the training split as reference, and molecular metrics include atom/bond distribution MMD, validity, uniqueness, novelty, and valid-unique-novel rates.

 doi:10.20944/preprints202607.1019.v1

Table A6. Dataset preparation and sample counts used in the illustrative benchmark. Splits are computed from the benchmark configuration; ZINC follows the official PyG subset split.

Dataset	Type	Graphs used	Train	Val.	Test	Key preparation settings
Planar	Synthetic topology	10,240	8,192	1,024	1,024	64 nodes; planar constraint; undirected; relabeled; self-loops removed.
SBM	Synthetic topology	10,240	8,192	1,024	1,024	64 nodes; 4 blocks; $p_{\text{in}} = 0.25$, $p_{\text{out}} = 0.02$; undirected; relabeled.
QM9	Molecular	12,000	9,600	1,200	1,200	PyG preprocessed archive; shuffled; node/edge features and targets kept; positions omitted.
ZINC	Molecular	12,000	10,000	1,000	1,000	PyG ZINC subset; official splits; node/edge features and targets kept; no shuffle.

Appendix E.2. Model-Specific Hyperparameters and Compute Budget

Table A7 summarizes the main model hyperparameters specified in the benchmark configuration. When a wrapper does not override an architecture field from the public upstream implementation, the table reports that the upstream default is used rather than restating an implementation-specific default. All models use batch size 32 and 100 training epochs in the benchmark configuration, but they differ in architecture settings, diffusion or transport schedules, and sampling budgets.

Table A7. Model-specific hyperparameters used in the benchmark. Values are taken from the benchmark configuration files; fields marked as upstream defaults are not overridden by the wrapper configuration.

Model	Training configuration	Architecture / representation configuration	Diffusion, transport, and sampling configuration
ConStruct	Epochs 100; batch 32; sample batch 32; learning rate 2×10^{-4}; weight decay 10^{-12}; validation every 10 epochs; train split used for node counts.	5 graph-transformer layers; hidden MLP dims $(X, E, y) = (256, 128, 128)$; hidden dims $(d_X, d_E, d_y) = (256, 64, 64)$; 8 heads; dropout 0.1; structural/eigen/cycle features enabled, but disabled for molecular datasets.	Absorbing-edge transition; diffusion steps 500; sampling steps 500; $\nu_X = \nu_C = \nu_E = \nu_y = 1$; loss weights $(1, 2, 5, 0)$.
DiGress	Epochs 100; batch 32; sample batch 32; learning rate 2×10^{-4}; workers 4; GPU flag 1.	No architecture overrides in wrapper config; upstream DiGress architecture defaults are used. QM9 keeps hydrogens (`remove_h=false`); generic molecular domain features disabled by default.	Discrete diffusion sampling metadata: 500 steps; temperature 1.0; finite guard checked every 50 steps.
DisCo	Epochs 100; batch 32; sample batch 32; learning rate 2×10^{-4}; weight decay 5×10^{-12}; edge loss weight 5.0.	Graph-transformer backbone; 5 layers; $n_{\text{dim}} = 128$; 8 heads; dropout 0.1; cycle/eigen/global features enabled, but structural features disabled for molecular datasets.	Marginal CTMC; $t_{\text{min}} = 0.01$; 50 sampling steps; $\beta = 2.0$, $\alpha = 0.8$; safe tau-leaping backend; sampler rate clip 1000.
EDP-GNN	Epochs 100; batch 32; sample batch 32; test batch 32; learning rate 10^{-3}; LR decay 0.999; weight decay 0.	GIN score network; hidden list $(16, 16, 16, 16)$; feature widths $(16, 16, 16, 16, 16)$; channels $(2, 4, 4, 4, 2)$; dropout 0; one stack; per-graph feature normalization; explicit masks preserve isolated nodes.	Score noise levels $\sigma \in \{0.1, 0.2, 0.4, 0.6, 0.8, 1.6\}$; Langevin sampler with 1000 steps; $\epsilon = 0.5$; gradient step size 0.01; fixed node number.
GraphGUIDE	Epochs 100; batch 32; sample batch 32; learning rate 10^{-3}; weight decay 0; clip grad 1.0.	GAT backbone; 4 GNN layers; hidden dim. 256; GAT hidden dim. 32; 8 heads; time embedding 256; spectrum dim. 5; random templates, max 2048.	Bernoulli zero-skip diffuser; $t_{\text{max}} = 1000$; diffuser parameters $a = 100$, $b = 10$.
GruM	Epochs 100; batch 32; sample batch 32; train $\epsilon = 0.002$ AdamW; learning rate 2×10^{-4}; weight decay 10^{-12}; EMA 0.999; clip grad 1.0; $\lambda_{\text{train}} = 5.0$;.	Upstream dataset-specific base config unless overridden; explicit masks preserve isolated nodes; features (eig1, eig2); feature scale 10.0; feature normalization enabled; permutation mixing enabled.	1000 diffusion scales; Euler predictor; no corrector; sample $\epsilon = 0.002$; TV kernel; quantization threshold 0.5;EMA used for sampling; noise removal enabled; QM9 constrained post-processing enabled.

 doi:10.20944/preprints202607.1019.v1

All experiments were run on a Linux server with two NVIDIA RTX A6000 GPUs, each with approximately 48 GB GPU memory. The software stack used Python 3.10.12, PyTorch 1.12.1+cu113, CUDA 11.3, cuDNN 8302, GCC 11.4.0, and Linux kernel 5.15.0. Table A8 records the configured training and sampling budgets and leaves measured wall-clock and memory values to be filled after the final benchmark run.

The benchmark codebase is publicly available and includes the dataset preparation scripts, model wrappers, configuration files, sampling scripts, and evaluation pipeline used for the illustrative benchmark [81].

Table A8. Compute budget for the illustrative benchmark. All runs used the same Linux server with two NVIDIA RTX A6000 GPUs, Python 3.10.12, PyTorch 1.12.1+cu113, CUDA 11.3, and cuDNN 8302.

Dataset	Model	Runs	Epochs	Batch	Sampling budget	Training time	Sampling time / 128 graphs	Peak memory
Planar	ConStruct	3	100	32	500 diffusion / sampling steps	1.97h	7.12m	5.46 GiB
Planar	DiGress	3	100	32	500 diffusion steps	1.85h	3.28m	5.73 GiB
Planar	DisCo	3	100	32	50 CTMC sampling steps	4.27h	20.92s	6.11 GiB
Planar	EDP-GNN	3	100	32	1000 Langevin steps	1.21h	5.37m	7.23 GiB
Planar	GraphGUIDE	3	100	32	$t_{\max} = 1000$	1.10h	3.84m	5.81 GiB
Planar	GruM	3	100	32	1000 scales; Euler sampler	3.26h	17.47m	8.35 GiB
QM9	ConStruct	3	100	32	500 diffusion / sampling steps	50.94m	48.78s	3.31 GiB
QM9	DiGress	3	100	32	500 diffusion steps	1.03h	43.86s	3.68 GiB
QM9	DisCo	3	100	32	50 CTMC sampling steps	1.26h	4.59s	3.59 GiB
QM9	GruM	3	100	32	1000 scales; Euler sampler	2.17h	2.90m	2.75 GiB
SBM	ConStruct	3	100	32	500 diffusion / sampling steps	1.99h	7.20m	5.46 GiB
SBM	DiGress	3	100	32	500 diffusion steps	1.84h	3.28m	5.73 GiB
SBM	DisCo	3	100	32	50 CTMC sampling steps	3.90h	20.79s	5.91 GiB
SBM	EDP-GNN	3	100	32	1000 Langevin steps	1.20h	5.71m	6.83 GiB
SBM	GraphGUIDE	3	100	32	$t_{\max} = 1000$	1.10h	3.79m	5.55 GiB
SBM	GruM	3	100	32	1000 scales; Euler sampler	2.52h	6.52m	8.47 GiB
ZINC	ConStruct	3	100	32	500 diffusion / sampling steps	1.38h	1.62m	5.12 GiB
ZINC	DiGress	3	100	32	500 diffusion steps	54.79m	1.19m	3.57 GiB
ZINC	DisCo	3	100	32	50 CTMC sampling steps	2.41h	8.49s	4.37 GiB
ZINC	GruM	3	100	32	1000 scales; Euler sampler	2.07h	3.06m	3.64 GiB

Appendix E.3. Benchmark Result Tables with Uncertainty

Tables A9 and A10 report the mean and standard deviation of the three independent runs. These tables complement the main benchmark tables, which report only the rank value.

Table A9. Illustrative comparison on synthetic graph benchmarks with mean and standard deviation over three independent runs. Lower is better for MMD, feature-space MMD, and PGS-JS.

Dataset	Model	Degree MMD	Clustering MMD	Orbit MMD	Spectral MMD	Feature-space MMD	PGS-JS ↓
Planar	ConStruct	0.0019 ± 0.0002	0.0435 ± 0.0076	0.0002 ± 0.0001	0.0006 ± 0.0001	0.6950 ± 0.0002	0.5356 ± 0.0348
Planar	DiGress	0.0041 ± 0.0014	0.1438 ± 0.0334	0.0018 ± 0.0005	0.0159 ± 0.0015	0.1014 ± 0.0038	0.9353 ± 0.0129
Planar	DisCo	0.0101 ± 0.0058	0.2132 ± 0.1157	0.0069 ± 0.0027	0.0231 ± 0.0044	0.6951 ± 0.0005	0.9329 ± 0.012
Planar	EDP-GNN	0.0163 ± 0.0117	1.8152 ± 0.0093	1.1326 ± 0.0268	0.7904 ± 0.0479	0.6842 ± 0.0024	0.9999 ± 0.0000
Planar	GraphGUIDE	1.0127 ± 0.4109	1.2111 ± 0.1443	1.3905 ± 0.0426	1.3709 ± 0.3219	0.2359 ± 0.0025	0.9999 ± 0.0000
Planar	GruM	0.0054 ± 0.0012	0.0839 ± 0.0158	0.0014 ± 0.0006	0.0079 ± 0.0017	0.6795 ± 0.0002	0.8695 ± 0.0275
SBM	ConStruct	0.0002 ± 0.0001	0.0070 ± 0.0036	0.0007 ± 0.0005	0.0009 ± 0.0004	0.6922 ± 0.0001	0.0472 ± 0.0436
SBM	DiGress	0.0001 ± 0.0001	0.0119 ± 0.0105	0.0014 ± 0.0009	0.0015 ± 0.0005	0.1053 ± 0.0011	0.1659 ± 0.0572
SBM	DisCo	0.0003 ± 0.0002	0.0198 ± 0.0156	0.0024 ± 0.0022	0.0017 ± 0.0004	0.6919 ± 0.0002	0.1022 ± 0.0636
SBM	EDP-GNN	0.2605 ± 0.0000	1.5748 ± 0.0019	1.7417 ± 0.0834	1.6733 ± 0.0572	0.7871 ± 0.0245	1.0000 ± 0.0000
SBM	GraphGUIDE	0.5709 ± 0.3239	0.9103 ± 0.0929	1.1358 ± 0.1895	0.8828 ± 0.3108	0.2042 ± 0.0111	0.9994 ± 0.0005
SBM	GruM	0.0032 ± 0.0012	0.0302 ± 0.0109	0.0059 ± 0.0020	0.0150 ± 0.0054	0.6751 ± 0.0002	0.2952 ± 0.0469

 doi:10.20944/preprints202607.1019.v1

Table A10. Illustrative comparison on molecular benchmarks with mean and standard deviation over three independent runs. Lower is better for atom-type MMD, bond-type MMD, feature-space MMD, and PGS-JS; higher is better for validity, uniqueness, and novelty

Dataset	Model	Atom-type MMD	Bond-type MMD	Validity ↑	Uniqueness ↑	Novelty ↑	Feature-space MMD	PGS-JS ↓
QM9	ConStruct	0.0015 ± 0.0004	0.0013 ± 0.0009	0.9202 ± 0.0044	0.9996 ± 0.0005	0.9844 ± 0.0020	0.7658 ± 0.0016	0.4452 ± 0.0316
QM9	DiGress	0.0021 ± 0.0004	0.0022 ± 0.0012	0.9349 ± 0.0047	0.9997 ± 0.0005	0.9854 ± 0.0060	0.7658 ± 0.0007	0.5045 ± 0.0140
QM9	DisCo	0.0031 ± 0.0013	0.0016 ± 0.0007	0.8571 ± 0.0140	0.9992 ± 0.0005	0.9799 ± 0.0005	0.7667 ± 0.0019	0.3871 ± 0.0293
QM9	GruM	0.0005 ± 0.0006	0.0013 ± 0.0015	1.0000 ± 0.0000	0.9997 ± 0.0005	0.9863 ± 0.0016	0.7835 ± 0.0010	0.5514 ± 0.0084
ZINC	ConStruct	0.0025 ± 0.0011	0.0164 ± 0.0088	0.3288 ± 0.2157	1.0000 ± 0.0000	1.0000 ± 0.0000	0.6885 ± 0.0026	0.8119 ± 0.0238
ZINC	DiGress	0.0077 ± 0.0057	0.0130 ± 0.0102	0.3057 ± 0.1688	1.0000 ± 0.0000	1.0000 ± 0.0000	0.6962 ± 0.0027	0.8865 ± 0.0110
ZINC	DisCo	0.0035 ± 0.0016	0.0075 ± 0.0093	0.6452 ± 0.0096	1.0000 ± 0.0000	1.0000 ± 0.0000	0.6915 ± 0.0029	0.8506 ± 0.01701
ZINC	GruM	0.0004 ± 0.0005	1.0808 ± 0.0018	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	0.1791 ± 0.0102	0.6994 ± 0.0094

Appendix F. Limitations of the Illustrative Benchmark

The benchmark is intended to illustrate evaluation practice and metric consistency rather than provide a definitive ranking of graph generative models. Its scope is limited to Planar, SBM, QM9, and ZINC, covering fixed-size synthetic topology, community structure, and small molecular graphs. It does not cover dynamic graphs, heterogeneous graphs, directed graphs, very large sparse graphs, or protein-scale molecular structures.

The evaluated models also make different design assumptions, including featureless graph topology, attributed molecular generation, constrained generation, continuous relaxations, CTMC dynamics, and bridge-based generation. Consequently, performance on one benchmark setting may not represent a model's strongest intended use case. The reported metrics should be interpreted jointly: topological MMDs, learned-feature distances, validity, uniqueness, and novelty capture different aspects of generated-graph quality and are not interchangeable. Scores may further vary with random seeds, training budget, checkpoint selection, sampling steps, solver choices, and post-processing.

References

1. Amira Alakhdar, Barnabás Póczos, and Newell Washburn. 2024. Diffusion Models in De Novo Drug Design. *Journal of Chemical Information and Modeling* 64, 19 (2024), 7238–7256. https://doi.org/10.1021/acs.jcim.4c01107
2. Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021. Structured Denoising Diffusion Models in Discrete State-Spaces. In *Advances in Neural Information Processing Systems*, Vol. 34. 17981–17993.
3. Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *Science* 286, 5439 (1999), 509–512.
4. Yoann Boget. 2025. Simple and critical iterative denoising: A recasting of discrete diffusion in graph generation. *arXiv preprint arXiv:2503.21592* (2025).
5. Montgomery Bohde, Mrunali Manjrekar, Runzhong Wang, Shuiwang Ji, and Connor W. Coley. 2025. DiffMS: Diffusion Generation of Molecules Conditioned on Mass Spectra. In *Proceedings of the 42nd International Conference on Machine Learning*.
6. Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. 2018. NetGAN: Generating Graphs via Random Walks. In *International Conference on Machine Learning*. 610–619.
7. Béla Bollobás. 2001. *Random Graphs* (2 ed.). Cambridge University Press. https://doi.org/10.1017/CBO9780511814068

 doi:10.20944/preprints202607.1019.v1

8. Avishek Bose, Tara Akhound-Sadegh, Guillaume Huguet, Kilian Fatras, Jarrid Rector-Brooks, Cheng-Hao Liu, Andrei Cristian Nica, Maksym Korablyov, Michael Bronstein, and Alexander Tong. 2024. SE(3) Stochastic Flow Matching for Protein Backbone Generation. In *International Conference on Learning Representations*.
9. Marc Brockschmidt, Miltiadis Allamanis, Alexander Gaunt, and Oleksandr Polozov. 2019. Generative Code Modeling with Graphs. In *International Conference on Learning Representations*.
10. Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. 2021. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv preprint arXiv:2104.13478* (2021).
11. Tianci Bu, Chuanrui Wang, Hao Ma, Haoren Zheng, Xin Lu, and Tailin Wu. 2025. GGBall: Graph Generative Model on Poincaré Ball. *arXiv preprint arXiv:2506.07198* (2025).
12. Andrew Campbell, Joe Benton, Valentin De Bortoli, George Deligiannidis, and Arnaud Doucet. 2022. Score-based Continuous-time Discrete Diffusion Models. *arXiv preprint arXiv:2211.16750* (2022).
13. Andrew Campbell, Joe Benton, Valentin De Bortoli, Tom Rainforth, George Deligiannidis, and Arnaud Doucet. 2022. A Continuous Time Framework for Discrete Denoising Models. In *Advances in Neural Information Processing Systems*, Vol. 35. 28266–28279.
14. Jinho Chang and Jong Chul Ye. 2025. LDMol: A Text-to-Molecule Diffusion Model with Structurally Informative Latent Space Surpasses AR Models. In *Proceedings of the 42nd International Conference on Machine Learning*.
15. Hongyang Chen, Can Xu, Lingyu Zheng, Qiang Zhang, and Xuemin Lin. 2024. Diffusion-Based Graph Generative Methods. *IEEE Transactions on Knowledge and Data Engineering* (2024).
16. Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. 2018. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems*, Vol. 31.
17. Xiaohui Chen, Jiaxing He, Xu Han, and Li-Ping Liu. 2023. Efficient and degree-guided graph generation via discrete diffusion modeling. *arXiv preprint arXiv:2305.04111* (2023).
18. Fan R. K. Chung. 1997. *Spectral Graph Theory*. CBMS Regional Conference Series in Mathematics, Vol. 92. American Mathematical Society.
19. Gabriele Corso, Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi Jaakkola. 2023. DiffDock: Diffusion Steps, Twists, and Turns for Molecular Docking. In *International Conference on Learning Representations*.
20. Hanjun Dai, Azade Nazi, Yujia Li, Bo Dai, and Dale Schuurmans. 2020. Scalable Deep Generative Modeling for Sparse Graphs. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*. PMLR, 2302–2312.
21. Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. 2018. Syntax-Directed Variational Autoencoder for Structured Data. In *International Conference on Learning Representations*.
22. Kishalay Das, Subhojyoti Khastagir, Pawan Goyal, Seung-Cheol Lee, Satadeep Bhattacharjee, and Niloy Ganguly. 2025. Periodic Materials Generation using Text-Guided Joint Diffusion Model. In *International Conference on Learning Representations*.
23. Valentin De Bortoli, James Thornton, Jeremy Heng, and Arnaud Doucet. 2021. Diffusion Schrödinger Bridge with Applications to Score-Based Generative Modeling. In *Advances in Neural Information Processing Systems*, Vol. 34. 17695–17709.
24. Zhiwei Deng, Megha Nawhal, Lili Meng, and Greg Mori. 2019. Continuous graph flow. *arXiv preprint arXiv:1908.02436* (2019).
25. Reinhard Diestel. 2017. *Graph Theory* (5 ed.). Graduate Texts in Mathematics, Vol. 173. Springer. https://doi.org/10.1007/978-3-662-53622-3
26. Floor Eijkelboom, Grigory Bartosh, Christian Andersson Naesseth, Max Welling, and Jan-Willem van de Meent. 2024. Variational flow matching for graph generation. *Advances in Neural Information Processing Systems* 37 (2024), 11735–11764.
27. Paul Erdős and Alfréd Rényi. 1959. On random graphs I. *Publicationes Mathematicae* 6 (1959), 290–297.
28. Faezeh Faez, Yassaman Ommi, Mahdieh Soleymani Baghshah, and Hamid R Rabiee. 2021. Deep graph generators: A survey. *IEEE Access* 9 (2021), 106675–106702.
29. Shikun Feng, Yuyan Ni, Yan Lu, Zhi-Ming Ma, Wei-Ying Ma, and Yanyan Lan. 2025. UniGEM: A Unified Approach to Generation and Property Prediction for Molecules. In *International Conference on Learning Representations*.
30. Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky T. Q. Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. 2024. Discrete Flow Matching. In *Advances in Neural Information Processing Systems*. Spotlight.

 doi:10.20944/preprints202607.1019.v1

31. Tomas Geffner, Kieran Didi, Zuobai Zhang, Danny Reidenbach, Zhonglin Cao, Jason Yim, Mario Geiger, Christian Dallago, Emine Kucukbenli, Arash Vahdat, and Karsten Kreis. 2025. Proteína: Scaling Flow-based Protein Structure Generative Models. In *International Conference on Learning Representations*.
32. Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*. PMLR, 1263–1272.
33. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, Vol. 27. 2672–2680.
34. Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. 2019. FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models. In *International Conference on Learning Representations*.
35. Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. A Kernel Two-Sample Test. *Journal of Machine Learning Research* 13, 25 (2012), 723–773. https://www.jmlr.org/papers/v13/gretton12a.html
36. Jiaqi Guan, Wesley Wei Qian, Xingang Peng, Yufeng Su, Jian Peng, and Jianzhu Ma. 2023. 3D Equivariant Diffusion for Target-Aware Molecule Generation and Affinity Prediction. In *International Conference on Learning Representations*.
37. Hongyu Guo, Yoshua Bengio, and Shengchao Liu. 2025. AssembleFlow: Rigid Flow Matching with Inertial Frames for Molecular Assembly. In *International Conference on Learning Representations*.
38. Xiaojie Guo and Liang Zhao. 2022. A systematic survey on deep generative models for graph generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 5 (2022), 5370–5390.
39. Kilian Konstantin Haefeli, Karolis Martinkus, Nathanaël Perraudin, and Roger Wattenhofer. 2022. Diffusion Models for Graphs Benefit From Discrete State Spaces. *arXiv preprint arXiv:2210.01549* (2022).
40. Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, Vol. 33. 6840–6851.
41. Haokai Hong, Wanyu Lin, and Kay Chen Tan. 2025. Accelerating 3D Molecule Generation via Jointly Geometric Optimal Transport. In *International Conference on Learning Representations*.
42. Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. 2021. Argmax Flows and Multinomial Diffusion: Learning Categorical Distributions. In *Advances in Neural Information Processing Systems*, Vol. 34. 12454–12465.
43. Emiel Hoogeboom, Victor Garcia Satorras, Clément Vignac, and Max Welling. 2022. Equivariant Diffusion for Molecule Generation in 3D. In *International Conference on Machine Learning*.
44. Xiaoyang Hou, Tian Zhu, Milong Ren, Dongbo Bu, Xin Gao, Chunming Zhang, and Shiwei Sun. 2025. GGFlow: A Graph Flow Matching Method with Efficient Optimal Transport. *Transactions on Machine Learning Research* (2025).
45. Zhilin Huang, Ling Yang, Xiangxin Zhou, Chujun Qin, Yijie Yu, Xiawu Zheng, Zikun Zhou, Wentao Zhang, Yu Wang, and Wenming Yang. 2024. Interaction-based Retrieval-augmented Diffusion Models for Protein-specific 3D Molecule Generation. In *International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*.
46. Ilia Igashov, Hannes Stärk, Clément Vignac, Arne Schneuing, Victor Garcia Satorras, Pascal Frossard, Max Welling, Michael Bronstein, and Bruno Correia. 2022. Equivariant 3D-Conditional Diffusion Model for Molecular Linker Design. In *NeurIPS Workshop on New Frontiers in Graph Learning*.
47. John J. Irwin and Brian K. Shoichet. 2005. ZINC–A Free Database of Commercially Available Compounds for Virtual Screening. *Journal of Chemical Information and Modeling* 45, 1 (2005), 177–182. https://doi.org/10.1021/ci049714+
48. Keyue Jiang, Jiahao Cui, Xiaowen Dong, and Laura Toni. 2025. Bures-Wasserstein Flow Matching for Graph Generation. *arXiv preprint arXiv:2506.14020* (2025).
49. Rui Jiao, Wenbing Huang, Peijia Lin, Jiaqi Han, Pin Chen, Yutong Lu, and Yang Liu. 2023. Crystal Structure Prediction by Joint Equivariant Diffusion. In *Advances in Neural Information Processing Systems*.
50. Bowen Jing, Bonnie Berger, and Tommi Jaakkola. 2024. AlphaFold Meets Flow Matching for Generating Protein Ensembles. In *International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*.

 doi:10.20944/preprints202607.1019.v1

51. Bowen Jing, Gabriele Corso, Jeffrey Chang, Regina Barzilay, and Tommi Jaakkola. 2022. Torsional Diffusion for Molecular Conformer Generation. In *Advances in Neural Information Processing Systems*.
52. Bowen Jing, Hannes Stärk, Tommi Jaakkola, and Bonnie Berger. 2024. Generative Modeling of Molecular Dynamics Trajectories. In *Advances in Neural Information Processing Systems*.
53. Jaehyeong Jo, Dongki Kim, and Sung Ju Hwang. 2023. Graph generation with diffusion mixture. *arXiv preprint arXiv:2302.03596* (2023).
54. Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. 2022. Score-based Generative Modeling of Graphs via the System of Stochastic Differential Equations. In *International Conference on Machine Learning*. arXiv:2202.02514.
55. Nayoung Kim, Seongsu Kim, Minsu Kim, Jinkyoo Park, and Sungsoo Ahn. 2025. MOFFlow: Flow Matching for Structure Prediction of Metal-Organic Frameworks. In *International Conference on Learning Representations*.
56. Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*.
57. Lingkai Kong, Jiaming Cui, Haotian Sun, Yuchen Zhuang, B. Aditya Prakash, and Chao Zhang. 2023. Autoregressive diffusion model for graph generation. In *OpenReview*.
58. Markus Krimmel, Philip Hartout, Karsten Borgwardt, and Dexiong Chen. 2025. PolyGraph Discrepancy: a classifier-based metric for graph generation. *arXiv preprint arXiv:2510.06122* (2025).
59. Tuan Le, Julian Cremer, Frank Noé, Djork-Arne Clevert, and Kristof Schütt. 2024. Navigating the Design Space of Equivariant Diffusion-Based Generative Models for De Novo 3D Molecule Generation. In *International Conference on Learning Representations*.
60. Seul Lee, Jaehyeong Jo, and Sung Ju Hwang. 2023. Exploring Chemical Space with Score-based Out-of-distribution Generation. In *International Conference on Machine Learning*.
61. Christian Léonard. 2014. A Survey of the Schrödinger Problem and Some of its Connections with Optimal Transport. *Discrete and Continuous Dynamical Systems - A* 34, 4 (2014), 1533–1574. https://doi.org/10.3934/dcds.2014.34.1533
62. Mufei Li, Viraj Shitole, Eli Chien, Changhai Man, Zhaodong Wang, Srinivas Sridharan, Ying Zhang, Tushar Krishna, and Pan Li. 2024. LayerDAG: A layerwise autoregressive diffusion model for directed acyclic graph generation. *arXiv preprint arXiv:2411.02322* (2024).
63. Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. 2018. Learning Deep Generative Models of Graphs. In *International Conference on Machine Learning*. 2837–2846.
64. Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Charlie Nash, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard S. Zemel. 2019. Efficient Graph Generation with Graph Recurrent Attention Networks. In *Advances in Neural Information Processing Systems*, Vol. 32.
65. Stratis Limnios, Praveen Selvaraj, Mihai Cucuringu, Carsten Maple, Gesine Reinert, and Andrew Elliott. 2023. SaGess: Sampling Graph Denoising Diffusion Model for Scalable Graph Generation. *arXiv preprint arXiv:2306.16827* (2023).
66. Haitao Lin, Yufei Huang, Meng Liu, Xuan Li, Shuiwang Ji, and Stan Z. Li. 2022. DiffBP: Generative Diffusion of 3D Molecules for Target Protein Binding. In *International Conference on Learning Representations Workshop*.
67. Yeqing Lin and Mohammed AlQuraishi. 2023. Generating Novel, Designable, and Diverse Protein Structures by Equivariantly Diffusing Oriented Residue Clouds. In *International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 202)*.
68. Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. 2023. Flow Matching for Generative Modeling. In *International Conference on Learning Representations*.
69. Chengyi Liu, Wenqi Fan, Yunqing Liu, Jiatong Li, Hang Li, Hui Liu, Jiliang Tang, and Qing Li. 2023. Generative diffusion models on graphs: Methods and applications. *arXiv preprint arXiv:2302.02591* (2023).
70. Shengchao Liu, Weitao Du, Zhiming Ma, Hongyu Guo, and Jian Tang. 2023. A Group Symmetric Stochastic Differential Equation Model for Molecule Multi-modal Pretraining. In *International Conference on Machine Learning*.
71. Xingchao Liu, Chengyue Gong, and Qiang Liu. 2023. Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow. In *International Conference on Learning Representations*.
72. Xinyang Liu, Yilin He, Bo Chen, and Mingyuan Zhou. 2024. Advancing graph generation through beta diffusion. *arXiv preprint arXiv:2406.09357* (2024).
73. Zhiyuan Liu, Yanchen Luo, Han Huang, Enzhi Zhang, Sihang Li, Junfeng Fang, Yaorui Shi, Xiang Wang, Kenji Kawaguchi, and Tat-Seng Chua. 2025. NExT-Mol: 3D Diffusion Meets 1D Language Modeling for 3D Molecule Generation. In *International Conference on Learning Representations*.

 doi:10.20944/preprints202607.1019.v1

74. Shitong Luo, Yufeng Su, Xingang Peng, Sheng Wang, Jian Peng, and Jianzhu Ma. 2022. Antigen-Specific Antibody Design and Optimization with Diffusion-Based Generative Models for Protein Structures. In *Advances in Neural Information Processing Systems*.
75. Chris J. Maddison and Daniel Tarlow. 2014. Structured Generative Models of Natural Source Code. In *Proceedings of the 31st International Conference on Machine Learning*.
76. Manuel Madeira, Clément Vignac, Dorina Thanou, and Pascal Frossard. 2024. Generative Modelling of Structurally Constrained Graphs. *Advances in Neural Information Processing Systems* 37 (2024), 137218–137262.
77. Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. 2019. Invariant and Equivariant Graph Networks. In *International Conference on Learning Representations*.
78. Giorgia Minello, Alessandro Bicciato, Luca Rossi, Andrea Torsello, and Luca Cosmo. 2024. Generating graphs via spectral diffusion. *arXiv preprint arXiv:2402.18974* (2024).
79. Mark Newman. 2010. *Networks: An Introduction*. Oxford University Press.
80. M. E. J. Newman. 2003. The structure and function of complex networks. *SIAM Rev.* 45, 2 (2003), 167–256.
81. Quang Nguyen. 2026. *Benchmark Survey: Codebase for Graph Generative Model Benchmarking*. https://doi.org/10.5281/zenodo.20520103
82. Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. 2020. Permutation Invariant Graph Generation via Score-Based Generative Modeling. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 108)*. PMLR, 4474–4484.
83. Divya Nori and Wengong Jin. 2024. RNAFlow: RNA Structure and Sequence Design via Inverse Folding-Based Flow Matching. In *International Conference on Machine Learning*.
84. Leslie O'Bray, Max Horn, Bastian Rieck, and Karsten Borgwardt. 2022. Evaluation Metrics for Graph Generative Models: Problems, Pitfalls, and Practical Solutions. In *International Conference on Learning Representations*.
85. Ole Petersen, Marcel Kollovieh, Marten Lienen, and Stephan Günnemann. 2025. Discrete Bayesian Sample Inference for Graph Generation. *arXiv preprint arXiv:2511.03015* (2025).
86. Gabriel Peyré and Marco Cuturi. 2019. Computational Optimal Transport. *Foundations and Trends in Machine Learning* 11, 5–6 (2019), 355–607. https://doi.org/10.1561/2200000073
87. Pedro O. Pinheiro, Arian Jamasb, Omar Mahmood, Vishnu Sresht, and Saeed Saremi. 2024. Structure-based Drug Design by Denoising Voxel Grids. In *International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*.
88. Bo Qiang, Yuxuan Song, Minkai Xu, Jingjing Gong, Bowen Gao, Hao Zhou, Weiying Ma, and Yanyan Lan. 2023. Coarse-to-Fine: a Hierarchical Diffusion Model for Molecule Generation in 3D. In *International Conference on Machine Learning*.
89. Yiming Qin, Manuel Madeira, Dorina Thanou, and Pascal Frossard. 2024. Defog: Discrete flow matching for graph generation. *arXiv preprint arXiv:2410.04263* (2024).
90. Yiming Qin, Clément Vignac, and Pascal Frossard. 2025. SparseDiff: Sparse Discrete Diffusion for Scalable Graph Generation. *Transactions on Machine Learning Research* (2025).
91. Raghunathan Ramakrishnan, Pavlo O. Dral, Matthias Rupp, and O. Anatole von Lilienfeld. 2014. Quantum Chemistry Structures and Properties of 134 Kilo Molecules. *Scientific Data* 1 (2014), 140022. https://doi.org/10.1038/sdata.2014.22
92. Arne Schneuing, Yuanqi Du, Charles Harris, Arian Jamasb, Ilia Igashov, Weitao Du, Tom Blundell, Pietro Lio, Carla Gomes, Max Welling, Michael Bronstein, and Bruno Correia. 2022. Structure-Based Drug Design with Equivariant Diffusion Models. In *NeurIPS Workshop on AI for Science*.
93. Kartik Sharma, Srijan Kumar, and Rakshit Trivedi. 2024. Diffuse, Sample, Project: Plug-And-Play Controllable Graph Generation. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. PMLR.
94. Chence Shi, Shitong Luo, Minkai Xu, and Jian Tang. 2021. Learning Gradient Fields for Molecular Conformation Generation. In *International Conference on Machine Learning*.
95. Hamed Shirzad, Kaveh Hassani, and Danica J. Sutherland. 2022. Evaluating Graph Generative Models with Contrastively Learned Features. In *Advances in Neural Information Processing Systems (NeurIPS)*.
96. Martin Simonovsky and Nikos Komodakis. 2018. GraphVAE: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*. Springer, 412–422.
97. Rylee Thompson, Boris Knyazev, Elahe Ghalebi, Jungtaek Kim, and Graham W. Taylor. 2022. On Evaluation Metrics for Graph Generative Models. In *International Conference on Learning Representations*.

 doi:10.20944/preprints202607.1019.v1

98. Brian L. Trippe, Jason Yim, Doug Tischer, Tamara Broderick, David Baker, Regina Barzilay, and Tommi Jaakkola. 2023. Diffusion Probabilistic Modeling of Protein Backbones in 3D for the Motif-Scaffolding Problem. In *International Conference on Learning Representations*.
99. Alex M Tseng, Nathaniel Diamant, Tommaso Biancalani, and Gabriele Scalia. 2023. GraphGUIDE: Interpretable and controllable conditional graph generation with discrete Bernoulli diffusion. *arXiv preprint arXiv:2302.03790* (2023).
100. Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. 2016. Neural Autoregressive Distribution Estimation. *Journal of Machine Learning Research* 17, 205 (2016), 1–37.
101. Yogesh Verma, Samuel Kaski, Markus Heinonen, and Vikas Garg. 2022. Modular flows: Differential molecular generation. *Advances in Neural Information Processing Systems* 35 (2022), 12409–12421.
102. Clément Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. 2022. DiGress: Discrete Denoising Diffusion for Graph Generation. *arXiv preprint arXiv:2209.14734* (2022).
103. Cédric Villani. 2009. *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften, Vol. 338. Springer. https://doi.org/10.1007/978-3-540-71050-9
104. Chentong Wang, Yannan Qu, Zhangzhi Peng, Yukai Wang, Hongli Zhu, Dachuan Chen, and Longxing Cao. 2024. Proteus: Exploring Protein Structure Generation for Enhanced Designability and Efficiency. In *International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*.
105. Liang Wang, Chao Song, Zhiyuan Liu, Yu Rong, Qiang Liu, and Shu Wu. 2025. Diffusion models for molecules: A survey of methods and tasks. *arXiv preprint arXiv:2502.09511* (2025).
106. Xinyou Wang, Zaixiang Zheng, Fei Ye, Dongyu Xue, Shujian Huang, and Quanquan Gu. 2024. Diffusion Language Models Are Versatile Protein Learners. In *International Conference on Machine Learning*.
107. Yan Wang, Lihao Wang, Yuning Shen, Yiqun Wang, Huizhuo Yuan, Yue Wu, and Quanquan Gu. 2024. Protein Conformation Generation via Force-Guided SE(3) Diffusion Models. In *International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*.
108. Joseph L. Watson, David Juergens, Nathaniel R. Bennett, Brian L. Trippe, Jason Yim, Helen E. Eisenach, Woody Ahern, Andrew J. Borst, Robert J. Ragotte, Lukas F. Milles, et al. 2023. De novo design of protein structure and function with RFdiffusion. *Nature* (2023).
109. Duncan J. Watts and Steven H. Strogatz. 1998. Collective dynamics of 'small-world' networks. *Nature* 393, 6684 (1998), 440–442.
110. Tian Xie, Xiang Fu, Octavian-Eugen Ganea, Regina Barzilay, and Tommi Jaakkola. 2022. Crystal Diffusion Variational Autoencoder for Periodic Material Generation. In *International Conference on Learning Representations*.
111. Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.
112. Minkai Xu, Alexander S. Powers, Ron O. Dror, Stefano Ermon, and Jure Leskovec. 2023. Geometric Latent Diffusion Models for 3D Molecule Generation. In *International Conference on Machine Learning*.
113. Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. 2022. GeoDiff: A Geometric Diffusion Model for Molecular Conformation Generation. In *International Conference on Learning Representations*.
114. Zhe Xu, Ruizhong Qiu, Yuzhong Chen, Huiyuan Chen, Xiran Fan, Menghai Pan, Zhichen Zeng, Mahashweta Das, and Hanghang Tong. 2024. Discrete-state Continuous-time Diffusion for Graph Generation. In *Advances in Neural Information Processing Systems 37*. OpenReview poster, arXiv:2405.11416.
115. Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*. PMLR, 5708–5717.
116. Yuning You, Ruida Zhou, Jiwoong Park, Haotian Xu, Chao Tian, Zhangyang Wang, and Yang Shen. 2024. Latent 3D Graph Diffusion. In *International Conference on Learning Representations*.
117. Jiying Zhang, Zijing Liu, Yu Wang, and Yu Li. 2024. SubGDiff: A Subgraph Diffusion Model to Improve Molecular Representation Learning. *arXiv preprint arXiv:2405.05665* (2024).
118. Zaixi Zhang, Yaosen Min, Shuxin Zheng, and Qi Liu. 2023. Molecule Generation for Target Protein Binding with Structural Motifs. In *International Conference on Learning Representations*.
119. Wenyin Zhou, Christopher Iliffe Sprague, Vsevolod Viliuga, Matteo Tadiello, Arne Elofsson, and Hossein Azizpour. 2025. Energy-Based Flow Matching for Generating 3D Molecular Structure. In *International Conference on Machine Learning*.

 doi:10.20944/preprints202607.1019.v1

120. Xiangxin Zhou, Xiwei Cheng, Yuwei Yang, Yu Bao, Liang Wang, and Quanquan Gu. 2024. Controllable and Decomposed Diffusion Models for Structure-Based Molecular Optimization. In *International Conference on Learning Representations*.
121. Yanqiao Zhu, Yuanqi Du, Yinkai Wang, Yichen Xu, Jieyu Zhang, Qiang Liu, and Shu Wu. 2022. A survey on deep graph generation: Methods and applications. In *Learning on Graphs Conference*. PMLR, 47–1.