Conditional Diffusion Based on Discrete Graph Structures for Molecular Graph Generation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Learning the underlying distribution of molecular graphs and generating high-fidelity samples is a fundamental research problem in drug discovery and material science. However, accurately modelling distribution and rapidly generating novel molecular graphs remain crucial and challenging goals. To accomplish these goals, we propose a novel Conditional Diffusion model based on discrete Graph Structures (CDGS) for molecular graph generation. Specifically, we construct a forward graph diffusion process on both graph structures and inherent features through stochastic differential equations (SDE) and derive discrete graph structures as the condition for reverse generative processes. We present a specialised hybrid graph noise prediction model that extracts the global context and the local node-edge dependency from intermediate graph states. We further utilize ordinary differential equation (ODE) solvers for efficient graph sampling, based on the semi-linear structure of the probability flow ODE. Experiments on diverse datasets validate the framework effectiveness. The proposed method, in particular, still generates high-quality molecular graphs in a limited number of steps.

1 Introduction

Dating back to the early works of Erdős Rényi random graphs [1], graph generation has been extensively studied for applications in biology, chemistry, and social science. Recent graph generative models make great progress in graph distribution learning by exploiting the capacity of neural networks. Models for molecular graph generation are notable for their success in representing molecule structures and restricting molecule search space. For the sampling process, graph generative models can construct molecular graphs step-by-step with decision sequences, known as autoregressive generation [2–5], whereas one-shot generation builds all graph components at once [6–8]. Recently, diffusion-based models have been applied effectively to one-shot molecular graph generation [9], highlighting the advantages of flexible model architectures and graph permutation invariant distribution modelling.

However, current diffusion models for molecular graphs still suffer from generation quality and sampling speed issues. In [9], the distribution of generated graphs faces an obvious distance with the true distribution of datasets. Furthermore, extra Langevin correction steps [10] are heavily relied on to diminish approximation errors in their generation process, largely increasing computation cost and implying insufficient expressive ability of models. We hypothesize that it is due to the focus on real-number graph formulation while neglecting the discrete graph structures. The straightforward design of score estimate networks may also fail to consider complex factors such as local chemical valency rules, global structure patterns, and even the atom type portion.

In this work, we propose a novel Conditional Diffusion model based on discrete Graph Structures (CDGS) for molecular graph generation. We develop a simple yet effective method to incorporate

Under review at the NeurIPS 2022 Workshop on Score-Based Methods. Do not distribute.

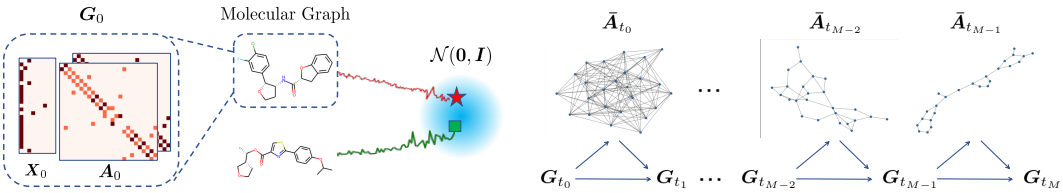


Figure 1: **(Left)** Forward diffusion process that perturbs molecular graphs towards a known prior distribution. A graph $\boldsymbol{G}_0$ is denoted by a node feature matrix $\boldsymbol{X}_0$ and a two-channel edge matrix $\boldsymbol{A}_0$ for edge types and existence. **(Right)** Discretized reverse generative process with discrete graph structure conditioning.

discrete graph structures without the special discrete state space. Along with variables for node and edge features, additional one-bit discrete variables are added to indicate the potential existence of edges. We convert them to real numbers and determine the quantization threshold. In our diffusion framework, the continuous forward process is applied directly to edge existence variables, but for the reverse process, discrete graph structures are decoded first and serve as the condition for each sampling step. We further develop a hybrid graph noise prediction model composed of standard message passing layers on discrete graphs and attention-based message passing layers on fully connected graphs. We employ stochastic differential equations (SDEs) to describe the graph diffusion process and promote fast sampling based on the associated probability flow ordinary differential equation (ODE). Specifically, we introduce fast ODE solvers for few-step graph sampling utilizing the semi-linear structure of probability flow ODE. Exploiting these ODE solvers, we also construct a useful pipeline for similarity-constrained molecule optimization, based on latent space determined by the parameterized ODE and gradient guidance from the graph property predictor.

2 Methodology

2.1 Conditional Graph Diffusion

The first step in constructing diffusion probabilistic models [11, 12, 10, 13] is to define a forward process that perturbs data with a sequence of noise until the output distribution becomes a known prior distribution. Assuming a continuous random variable $\boldsymbol{x}_0 \in \mathbb{R}^d$ and a well-defined forward process $\{\boldsymbol{x}_t\}_{t\in[0,T]}$, we have

$$q_{0t}(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t|\alpha_t\boldsymbol{x}_0, \sigma_t^2\boldsymbol{I}) \ , \tag{1}$$

where $\alpha_t, \sigma_t \in \mathbb{R}^+$ are time-dependant differentiable functions. α_t and σ_t are usually chosen to ensure that $q_T(\boldsymbol{x}_T) \approx \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ with the decreasing signal-to-noise ratio α_t^2/σ_t^2. By learning to reverse such a process, the diffusion model generates new samples from the prior distribution.

Formulating graphs as high-dimensional variables $\boldsymbol{G} \in \mathbb{R}^{N\times F} \times \mathbb{R}^{N\times N}$ composed of N node features and the weighted adjacency matrix is a simple method to apply diffusion models to the graph domain [9]. In this formulation, the discrete graph structures are ignored throughout the process. We argue that discrete graph structures, including patterns like rings and stars, can provide important clues for node-edge dependency modelling and graph denoising, and that the edge existence matrix should be separated from the edge type matrix. We propose to utilize a one-bit discrete variable representing the existence of a possible edge, forming $\bar{\boldsymbol{A}} \in \{0,1\}^{N\times N}$ for the whole graph. Instead of designing special discrete state spaces for discrete variables like [14, 15], we turn bits into real numbers and determine a quantization threshold. Thus, we can conveniently apply continuous diffusion process to these variables and decode them with quantization back to discrete graph structure $\bar{\boldsymbol{A}}_t$ for $t \in [0, T]$. The discrete graph structures can be plugged into the reverse process and function as conditions.

In addition to the corresponding $\bar{\boldsymbol{A}}$, we redefine the graph $\boldsymbol{G}$ by node features $\boldsymbol{X} \in \mathbb{R}^{N\times F}$ and edge information $\boldsymbol{A} \in \mathbb{R}^{2\times N\times N}$ (one channel for edge existence and the other for edge types). The forward diffusion process for graphs shown in Figure 1 can be described by the stochastic differential equation (SDE) sharing the same transition distribution in Eq. 1 [13] with $t \in [0, T]$ as

$$\mathrm{d}\boldsymbol{G}_t = f(t)\boldsymbol{G}_t\mathrm{d}t + g(t)\mathrm{d}\boldsymbol{w}_t \ , \tag{2}$$

where $f(t) = \frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t}$ is the drift coefficient, $g^2(t) = \frac{\mathrm{d}\sigma_t^2}{\mathrm{d}t} - 2\frac{\mathrm{d}\log\alpha_t}{\mathrm{d}t}\sigma_t^2$ is the diffusion coefficient, and $\boldsymbol{w}_t$ is a standard Wiener process. The reverse-time SDE from time T to 0 [10] equivalent to Eq. 2 is denoted as:

$$\mathrm{d}\boldsymbol{G}_t = [f(t)\boldsymbol{G}_t - g^2(t)\nabla_{\boldsymbol{G}}\log q_t(\boldsymbol{G}_t)]\mathrm{d}_t + g(t)\mathrm{d}\bar{\boldsymbol{w}}_t\ , \tag{3}$$

where $\nabla_{\boldsymbol{G}}\log q_t(\boldsymbol{G}_t)$ is the graph score function and $\bar{\boldsymbol{w}}_t$ is the reverse-time standard Wiener process. To alleviate the expensive computation of high-dimensional graph scores, we further split the reverse-time SDE into two parts, sharing the drift and diffusion coefficient as

$$\begin{cases} \mathrm{d}\boldsymbol{X}_t = [f(t)\boldsymbol{X}_t - g^2(t)\nabla_{\boldsymbol{X}}\log q_t(\boldsymbol{X}_t, \boldsymbol{A}_t)]\mathrm{d}_t + g(t)\mathrm{d}\bar{\boldsymbol{w}}_t^1 \\ \mathrm{d}\boldsymbol{A}_t = [f(t)\boldsymbol{A}_t - g^2(t)\nabla_{\boldsymbol{A}}\log q_t(\boldsymbol{X}_t, \boldsymbol{A}_t)]\mathrm{d}_t + g(t)\mathrm{d}\bar{\boldsymbol{w}}_t^2 \end{cases}. \tag{4}$$

We use a neural network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{G}_t, \bar{\boldsymbol{A}}_t, t)$ with discrete graph structure conditioning to parameterize the σ_t-scaled partial scores in Eq. 4, where the node output of the neural network is denoted by $\boldsymbol{\epsilon}_{\boldsymbol{\theta},\boldsymbol{X}}(\boldsymbol{G}_t, \bar{\boldsymbol{A}}_t, t)$ estimating $-\sigma_t\nabla_{\boldsymbol{X}}\log q_t(\boldsymbol{X}_t, \boldsymbol{A}_t)$, and the edge output is denoted by $\boldsymbol{\epsilon}_{\boldsymbol{\theta},\boldsymbol{A}}(\boldsymbol{G}_t, \bar{\boldsymbol{A}}_t, t)$ estimating $-\sigma_t\nabla_{\boldsymbol{A}}\log q_t(\boldsymbol{X}_t, \boldsymbol{A}_t)$. The model is optimized by

$$\min_{\boldsymbol{\theta}} \mathbb{E}_t\{w(t)\mathbb{E}_{\boldsymbol{G}_0}\mathbb{E}_{\boldsymbol{G}_t|\boldsymbol{G}_0}[||\boldsymbol{\epsilon}_{\boldsymbol{\theta},\boldsymbol{X}}(\boldsymbol{G}_t, \bar{\boldsymbol{A}}_t, t) - \boldsymbol{\epsilon}_{\boldsymbol{X}}||_2^2 + ||\boldsymbol{\epsilon}_{\boldsymbol{\theta},\boldsymbol{A}}(\boldsymbol{G}_t, \bar{\boldsymbol{A}}_t, t) - \boldsymbol{\epsilon}_{\boldsymbol{A}}||_2^2]\}\ , \tag{5}$$

where $w(t)$ is a given positive weighting function, $\boldsymbol{\epsilon}_{\boldsymbol{X}}$ and $\boldsymbol{\epsilon}_{\boldsymbol{A}}$ are the sampled Gaussian noise, and $\boldsymbol{G}_t = (\alpha_t\boldsymbol{X}_0 + \sigma_t\boldsymbol{\epsilon}_{\boldsymbol{X}}, \alpha_t\boldsymbol{A}_0 + \sigma_t\boldsymbol{\epsilon}_{\boldsymbol{A}})$. With the optimized $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ and numerical solvers discretizing the SDE trajectory, shown in the right of Figure 1, new graph samples can be generated.

2.2 Graph Noise Prediction Model

Since $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{G}_t, \bar{\boldsymbol{A}}_t, t)$ can be considered to predict the noise that is added to the original graph data, we refer to it as the graph noise prediction model. The design of noise prediction models plays a key role in diffusion-based generation, but it is still an open problem for the graph domain. In the case of molecular graphs, the model should focus on local node-edge dependence for chemical valency rules and also attempt to recover global graph patterns like edge sparsity, frequent ring subgraphs, and even atom type distribution.

To meet these challenges, we propose a hybrid message passing block (HMPB) consisting of two different kinds of message passing layers. One is a standard message passing layer like GINE [16] to aggregate local neighbor node-edge features, relying on the decoded discrete graph structures. The other one is a fully-connected attention-based message passing layer to focus on global information extraction and transmission. We denote the node and edge update process in the l-th HMPB as

$$\begin{aligned} \boldsymbol{H}^{l+1}, \boldsymbol{E}^{l+1} &= \mathrm{HMPB}^l(\boldsymbol{H}^l, \boldsymbol{E}^l, \bar{\boldsymbol{A}}), \\ \text{with} \quad \boldsymbol{M}^{l+1} &= \mathrm{GINE}^l(\boldsymbol{H}^l, \boldsymbol{E}^l, \bar{\boldsymbol{A}}) + \mathrm{ATTN}^l(\boldsymbol{H}^l, \boldsymbol{E}^l), \\ \boldsymbol{H}^{l+1} &= \mathrm{FFN}_0^l(\boldsymbol{M}^{l+1}), \\ \boldsymbol{E}^{l+1} &= \mathrm{FFN}_1^l(\boldsymbol{M}^{l+1} + (\boldsymbol{M}^{l+1})^\top), \end{aligned} \tag{6}$$

where $\boldsymbol{H}^l, \boldsymbol{E}^l$ are node and edge inputs from the last block; ATTN^l is the full-connected attention layer; FFN^l is Feed Forward Network composed of the multilayer perceptron (MLP) and normalization layers. Here, the time t and residual connections are omitted for clarity. In particular, different from [17–19], our attention layer takes edge features as the gate for both the message and dot-product calculation to thoroughly interact with node features and bias the message passing. The key attention mechanism is denoted by

$$\mathrm{softmax}(\frac{(\tanh(\phi_0(e)) \cdot Q)K^\top}{\sqrt{d}})(\tanh(\phi_1(e)) \cdot V), \tag{7}$$

where Q, K, V are projected from node features, e is the corresponding edge feature, ϕ_0 and ϕ_1 are learnable projections, and $\tanh$ is the activation layer.

For the initial features $\boldsymbol{H}^0$ and $\boldsymbol{E}^0$, we not only consider $\boldsymbol{X}_t$ and $\boldsymbol{A}_t$, but also extract structural encodings and relative positional encodings from $\bar{A}_t$. Using the m-step random walk matrix from the discrete adjacency matrix, we adopt the arrival probability vector as node features and obtain the shortest-path distance from the same matrix as edge features. Time information is also added to the initial features with the sinusoidal position embedding [20]. The final node and edge representations are respectively input to MLPs for graph noise prediction.

For the sampling process, we provide the details on our ODE samplers for graphs (GDPMS) in Appendix. We also introduce the solvers with gradient guidance for similarity-constrained optimization.

Table 1: Generation performance on ZINC250k **(Up)** and QM9 **(Down)**. The novelty metric on QM9 dataset denoted with $\star$ is debatable due to its contradiction with distribution learning.

	Method	VALID w/o check (%) ↑	NSPDK ↓	FCD ↓	VALID (%) ↑	UNIQUE (%) ↑	NOVEL (%) ↑
	Train	-	*5.91e-5*	*0.985*	-	-	-
Autoreg.	GraphAF	68.00	0.044	16.289	100.00	99.10	100.00
	GraphAF+FC	68.47	0.044	16.023	100.00	98.64	99.99
	GraphDF	89.03	0.176	34.202	100.00	99.16	100.00
	GraphDF+FC	90.61	0.177	33.546	100.00	99.63	100.00
One-shot	MoFlow	63.11	0.046	20.931	100.00	99.99	100.00
	GraphCNF	96.35	0.021	13.532	100.00	99.98	100.00
	EDP-GNN	82.97	0.049	16.737	100.00	99.79	100.00
	GraphEBM	5.29	0.212	35.471	99.96	98.79	100.00
	GDSS	97.01	0.019	14.656	100.00	99.64	100.00
	GDSS-EM	15.97	0.075	24.310	100.00	100.00	100.00
	GDSS-VP-EM	33.01	0.048	24.471	100.00	100.00	100.00
	CDGS-EM	**98.13**	**7.03e-4**	**2.069**	100.00	99.99	99.99
	CDGS-GDPMS-200	96.19	0.001	3.037	100.00	99.98	99.99
	CDGS-GDPMS-50	95.56	0.002	3.567	100.00	99.98	99.99
	CDGS-GDPMS-30	93.49	0.003	4.498	100.00	99.99	99.99

	Method	VALID w/o check (%) ↑	NSPDK ↓	FCD ↓	VALID (%) ↑	UNIQUE (%) ↑	*NOVEL (%)* $\star$
	Train	-	*1.36e-4*	*0.057*	-	-	-
Autoreg.	GraphAF	67.00	0.020	5.268	100.00	94.51	88.83
	GraphAF+FC	74.43	0.021	5.625	100.00	88.64	86.59
	GraphDF	82.67	0.063	10.816	100.00	97.62	98.10
	GraphDF+FC	93.88	0.064	10.928	100.00	98.58	98.54
One-shot	MoFlow	91.36	0.017	4.467	100.00	98.65	94.72
	EDP-GNN	47.52	0.005	2.680	100.00	99.25	86.58
	GraphEBM	8.22	0.030	6.143	100.00	97.90	97.01
	GDSS	95.72	0.003	2.900	100.00	98.46	86.27
	GDSS-EM	66.01	0.016	5.112	100.00	90.05	94.24
	GDSS-VP-EM	86.02	0.013	4.588	100.00	89.03	88.63
	CDGS-EM	**99.68**	**3.08e-4**	**0.200**	100.00	96.83	69.62
	CDGS-GDPMS-200	99.54	3.68e-4	0.269	100.00	97.20	72.52
	CDGS-GDPMS-50	99.47	3.85e-4	0.289	100.00	97.27	72.38
	CDGS-GDPMS-30	99.18	4.13e-4	0.326	100.00	97.42	72.52

3 Experiment

We compare our CDGS with several autoregressive and one-shot molecular graph generative models, including **GraphAF** [4], **GraphDF** [5], **MoFlow** [6], **GraphCNF** [7], **EDP-GNN** [21], **GraphEBM** [8], and **GDSS** [9]. **GraphAF+FC** and **GraphDF+FC** are the modified versions considering formal charges for fair comparison. **GDSS-EM** is the result sampled with the EM solver, and **GDSS-VP-EM** is retrained with VPSDE, sharing the same SDE parameters with our model.

The molecular graph generation quality benchmark results on ZINC250k and QM9 are reported in Table 1. In the first three non-trivial metrics across two different molecule datasets, CDGS with the EM solver markedly outperforms state-of-the-art molecular graph generative models. The high validity rate before valency checking shows that CDGS learns the chemical valency rule successfully and avoids unrealistically frequent valency correction. Furthermore, with much lower NSPDK and FCD values, CDGS learns the underlying distribution more faithfully in both graph and chemical space. CDGS achieves such performance without any Langevin correction steps in sampling, while previous diffusion-based GDSS drops off obviously with the pure EM solver. Using the same SDE parameters, the performance gap between GDSS-VP-EM and CDGS-EM further demonstrates the effectiveness of our framework design. Equipped with the 3rd-order GDPMS, our proposed model maintains excellent generation ability with limited NFE decreasing from 200 to 30.

We also point out that the novelty metric on the QM9 dataset is debatable because the QM9 dataset is almost an exhaustive list of molecules that adhere to a predetermined set of requirements [22, 23]. Therefore, a molecule that is thought to be novel violates the constraints, which means the model is unable to capture the dataset properties. This metric is kept for experiment completeness.

4 Conclusion

We present a novel conditional diffusion model for molecular graph generation that takes advantage of discrete graph structure conditioning and delicate graph noise prediction model design. Our model markedly outperforms existing molecular graph generative methods in both graph space and chemical space for distribution learning, and also performs well for efficient graph sampling.

References

[1] Paul Erdős, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[2] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay S. Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. In *NeurIPS*, pages 6412–6422, 2018.

[3] Wengong Jin, Regina Barzilay, and Tommi S. Jaakkola. Junction tree variational autoencoder for molecular graph generation. In Jennifer G. Dy and Andreas Krause, editors, *ICML*, pages 2328–2337, 2018.

[4] Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. In *ICLR*, 2020.

[5] Youzhi Luo, Keqiang Yan, and Shuiwang Ji. Graphdf: A discrete flow model for molecular graph generation. In *ICML*, pages 7192–7203, 2021.

[6] Chengxi Zang and Fei Wang. Moflow: an invertible flow model for generating molecular graphs. In *SIGKDD*, pages 617–626, 2020.

[7] Phillip Lippe and Efstratios Gavves. Categorical normalizing flows via continuous transformations. In *ICLR*, 2021.

[8] Meng Liu, Keqiang Yan, Bora Oztekin, and Shuiwang Ji. Graphebm: Molecular graph generation with energy-based models. *arXiv preprint arXiv:2102.00546*, 2021.

[9] Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. In *ICML*, pages 10362–10383, 2022.

[10] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *ICLR*, 2021.

[11] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, pages 2256–2265, 2015.

[12] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

[13] Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. In *NeurIPS*, 2021.

[14] Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. In *NeurIPS*, pages 12454–12465, 2021.

[15] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *NeurIPS*, pages 17981–17993, 2021.

[16] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay S. Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *ICLR*, 2020.

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

[18] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In *NeurIPS*, pages 28877–28888, 2021.

[19] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. In *NeurIPS*, volume 34, 2021.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[21] Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permutation invariant graph generation via score-based generative modeling. In *AISTATS*, pages 4474–4484, 2020.

[22] Clement Vignac and Pascal Frossard. Top-n: Equivariant set and graph generation without exchangeability. In *ICLR*, 2022.

[23] Emiel Hoogeboom, Victor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *ICML*, pages 8867–8887, 2022.

[24] Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. *arXiv preprint arXiv:2204.13902*, 2022.

[25] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps. *arXiv preprint arXiv:2206.00927*, 2022.

[26] Marlis Hochbruck and Alexander Ostermann. Explicit exponential runge-kutta methods for semilinear parabolic problems. *SIAM J. Numer. Anal.*, 43(3):1069–1090, 2005.

[27] Marlis Hochbruck and Alexander Ostermann. Exponential integrators. *Acta Numer.*, 19: 209–286, 2010.

[28] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[29] Matt J. Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *ICML*, pages 1945–1954, 2017.

[30] Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-directed variational autoencoder for structured data. In *ICLR*, 2018.

[31] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *ICANN*, pages 412–422. Springer, 2018.

[32] Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L. Gaunt. Constrained graph variational autoencoders for molecule design. In *NeurIPS 2018*, pages 7806–7815, 2018.

[33] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[34] Rim Assouel, Mohamed Ahmed, Marwin H. S. Segler, Amir Saffari, and Yoshua Bengio. Defactor: Differentiable edge factorization-based probabilistic graph generation. *arXiv preprint arXiv: 1811.09766*, 2018.

[35] Sungsoo Ahn, Binghong Chen, Tianzhe Wang, and Le Song. Spanning tree-based graph generation for molecules. In *ICLR*, 2022.

[36] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *NeurIPS*, volume 32, 2019.

[37] Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. Geodiff: A geometric diffusion model for molecular conformation generation. In *ICLR*, 2022.

[38] Bowen Jing, Gabriele Corso, Jeffrey Chang, Regina Barzilay, and Tommi S. Jaakkola. Torsional diffusion for molecular conformer generation. *arXiv preprint arXiv:2206.01729*, 2022.

[39] Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo numerical methods for diffusion models on manifolds. In *ICLR*, 2022.

[40] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018.

[41] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *J. Chem. Inf. Model.*, 50 (5):742–754, 2010.

[42] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *ESWC*, pages 593–607, 2018.

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

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

[45] Leslie O'Bray, Max Horn, Bastian Rieck, and Karsten Borgwardt. Evaluation metrics for graph generative models: Problems, pitfalls, and practical solutions. In *ICLR*, 2022.

[46] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *ICML*, pages 5708–5717, 2018.

[47] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard S. Zemel. Efficient graph generation with graph recurrent attention networks. In *NeurIPS*, pages 4257–4267, 2019.

[48] Rylee Thompson, Boris Knyazev, Elahe Ghalebi, Jungtaek Kim, and Graham W. Taylor. On evaluation metrics for graph generative models. In *ICLR*, 2022.

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

[50] Kristina Preuer, Philipp Renz, Thomas Unterthiner, Sepp Hochreiter, and Günter Klambauer. Fréchet chemnet distance: A metric for generative models for molecules in drug discovery. *J. Chem. Inf. Model.*, 58(9):1736–1741, 2018.

[51] John J. Irwin, Teague Sterling, Michael M. Mysinger, Erin S. Bolstad, and Ryan G. Coleman. ZINC: A free tool to discover chemistry for biology. *J. Chem. Inf. Model.*, 52(7):1757–1768, 2012.

[52] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 2014.

[53] Greg Landrum. Rdkit: Open-source cheminformatics software. 2016. URL http://www.rdkit.org.

A ODE Solvers for Few-step Graph Sampling

To generate graphs from the parameterized SDE in Eq. 4, it requires simulating the SDE trajectory with numerical solvers. The Euler-Maruyama (EM) solver is one of the simple and general solvers for SDEs. However, such a solver needs long steps (a.k.a., number of function evaluation (NFE)) to achieve convergence for the high-dimensional data space and fails to meet the fast sampling requirement. One promising fast sampling method is to use the mature black-box ODE solvers for the probability flow ODE [10] that shares the same marginal distribution at time t with the SDE. Accordingly, the parameterized probability flow ODE for graphs is defined as

$$\mathrm{d}\boldsymbol{G}_t/\mathrm{d}t = f(t)\boldsymbol{G}_t + \frac{g^2(t)}{2\sigma_t}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{G}_t, \bar{\boldsymbol{A}}_t, t)\ . \tag{8}$$

Recent works [24, 25] claim that the general black-box ODE solvers ignore the semi-linear structure of the probability flow ODE and introduce additional discretization errors. Therefore, new fast solvers are being developed to take advantage of the special structure of the probability flow ODE.

For our graph ODE in Eq. 8, we further extend fast solvers based on the semi-linear ODE structure to generate high-quality graphs within a few steps. By introducing $\lambda_t := \log(\alpha_t/\sigma_t)$ and its inverse function $t_\lambda(\cdot)$ that satisfies $t = t_\lambda(\lambda(t))$, we change the subscript t to λ and get $\hat{\boldsymbol{G}}_\lambda := \boldsymbol{G}_{t_\lambda(\lambda)}$, $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\hat{\boldsymbol{G}}_\lambda, \bar{\boldsymbol{A}}'_\lambda, \lambda) := \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\boldsymbol{G}_{t_\lambda(\lambda)}, \bar{\boldsymbol{A}}_{t_\lambda(\lambda)}, \lambda)$. We can derive the exact solution of the semi-linear probability flow ODE from time s to time t [25] as

$$\boldsymbol{G}_t = \frac{\alpha_t}{\alpha_s}\boldsymbol{G}_s - \alpha_t \int_{\lambda_s}^{\lambda_t} e^{-\lambda}\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}(\hat{\boldsymbol{G}}_\lambda, \bar{\boldsymbol{A}}'_\lambda, \lambda)\mathrm{d}\lambda\ . \tag{9}$$

With the analytical linear part, we only need to approximate the exponentially weighted integral of $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}$. This approximation can be achieved by various methods [26, 27], and we follow the derivation from [25] to obtain graph DPM-Solvers (GDPMS). Given the initial graph sampled from the prior distribution $\tilde{\boldsymbol{G}}_{t_0} := \boldsymbol{G}_T = (\boldsymbol{X}_T, \boldsymbol{A}_T)$ with the predefined time step schedules $\{t_i\}_{i=0}^M$, the sequence $\{\tilde{\boldsymbol{G}}_{t_i} = (\tilde{\boldsymbol{X}}_{t_i}, \tilde{\boldsymbol{A}}_{t_i})\}_{i=1}^M$ is calculated iteratively by the GDPMS of order 1 as follows:

$$\begin{cases} \tilde{\boldsymbol{X}}_{t_i} = \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{X}}_{t_{i-1}} - \gamma_i\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta},\boldsymbol{X}}(\tilde{\boldsymbol{G}}_{t_{i-1}}, \bar{\boldsymbol{A}}'_{t_{i-1}}, t_{i-1}) \\ \tilde{\boldsymbol{A}}_{t_i} = \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{A}}_{t_{i-1}} - \gamma_i\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta},\boldsymbol{A}}(\tilde{\boldsymbol{G}}_{t_{i-1}}, \bar{\boldsymbol{A}}'_{t_{i-1}}, t_{i-1}) \end{cases}, \tag{10}$$

where $\gamma_i = \sigma_{t_i}(e^{\lambda_{t_i} - \lambda_{t_{i-1}}} - 1)$, and discrete graph structure $\bar{\boldsymbol{A}}'_{t_{i-1}}$ is decoded from $\tilde{\boldsymbol{G}}_{t_{i-1}}$. The final graph sample is derived from $\tilde{\boldsymbol{G}}_{\boldsymbol{t_M}}$ with discretization.

B ODE-based Graph Optimization

Besides efficient sampling, the probability flow ODE offers latent representations for flexible data manipulation [10]. Based on our graph DPM-Solvers assisted by gradient guidance, we propose a useful optimization pipeline for the meaningful similarity-constrained molecule optimization task.

Specifically, we first train an extra time-dependent graph property predictor $\boldsymbol{R}_{\boldsymbol{\psi}}(\boldsymbol{G}_t, t)$ on noised graphs. Then we setup a solver for the parameterized ODE in Eq. 8 to map the initial molecular graphs at time 0 to the latent codes $\mathcal{G}_{t_\xi}$ at the time $t_\xi \in (0, T]$. Following the common optimization manipulation on latent space like [3, 6], we use the predictor to predict properties on the graph latent representation and lead the optimization towards molecules with desired properties through the gradient ascent, producing a latent graph sequence $\{\boldsymbol{\mathcal{G}}_{t_\xi}^k\}_{k=0}^K$. Instead of using the same ODE as in the forward encoding process, we introduce the gradient-guided ODE to further drive the sampling process to the high-property region during the decoding process from the latent space to the molecular graph space. The ODE with guidance can be modified from Eq. 8 as

$$\begin{cases} \mathrm{d}\boldsymbol{X}_t/\mathrm{d}t = f(t)\boldsymbol{X}_t + \frac{g^2(t)}{2\sigma_t}[\boldsymbol{\epsilon}_{\boldsymbol{\theta},\boldsymbol{X}} - r\sigma_t\nabla^*_{\boldsymbol{X}}\boldsymbol{R}_{\boldsymbol{\psi}}] \\ \mathrm{d}\boldsymbol{A}_t/\mathrm{d}t = f(t)\boldsymbol{A}_t + \frac{g^2(t)}{2\sigma_t}[\boldsymbol{\epsilon}_{\boldsymbol{\theta},\boldsymbol{A}} - r\sigma_t\nabla^*_{\boldsymbol{A}}\boldsymbol{R}_{\boldsymbol{\psi}}] \end{cases}, \tag{11}$$

where r is the guidance weight, ∇^* refers to the unit normalized gradients, and the input $(\boldsymbol{G}_t, \bar{\boldsymbol{A}}_t, t)$ for $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ and $(\boldsymbol{G}_t, t)$ for $\boldsymbol{R}_{\boldsymbol{\psi}}$ are omitted for simplicity. Notably, the GDPMS in Eq. 10 can still work for the gradient-guided ODE by constructing the $\hat{\boldsymbol{\epsilon}}_{\boldsymbol{\theta}}$ with the predictor gradients accordingly. The proposed pipeline can also be flexibly extended for multi-objective optimization by expanding the gradient guidance from multiple property prediction networks.

C Related Work

C.1 Molecule Generation

Early attempts for molecule generation introduce sequence-based generative models and represent molecules as SMILES strings [28–30]. Besides the challenge from long dependency modelling, these methods may exhibit low validity rates since the SMILES string does not ensure absolute validity. Therefore, graphs are more commonly used to represent molecule structures in recent studies. Various graph generative models have been proposed to construct graphs autoregressively or in a one-shot form, based on different types of generative models, including variational auto-encoders [31, 32], generative adversarial networks [33, 34], and normalizing flows [4, 5, 7, 6]. Compared to these models, our diffusion-based model advances in stable training and adaptable model architecture to consider the discrete graph structure for dependency modelling. In addition, [3, 35] adopt an effective tree-based graph formulation for molecules, while our method keeps the general graph settings and models permutation invariant distributions.

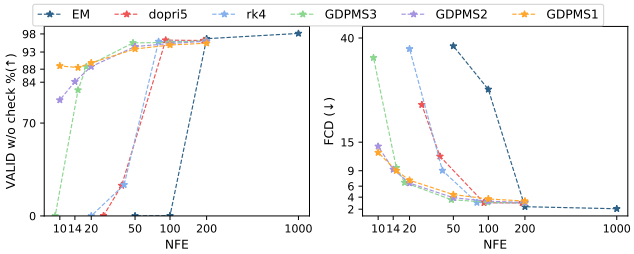
C.2 Diffusion Models

This new family of generative models [11, 12] correlated with score-based models [10, 36] has demonstrated great power in the generation of high-dimensional data such as images. For molecule science, in addition to molecular graph generation [9], diffusion models have also been applied to generate molecular conformations [37, 38] and 3D molecular structures [23]. Our framework greatly differs from the previous diffusion-based molecule generation in the conditional reverse process and the unified model design instead of separate models for nodes and edges. Moreover, we promote efficient molecular graph generation with training-free samplers, which is primarily investigated in the image domain [39, 24, 25].

D Additional Experiments

D.1 Fast Sampling

To explore fast and high-quality few-step molecular graph sampling, we compare the sampling quality of CDGS with different types of numerical solvers, including GDPMS with different orders, the EM solver, and black-box ODE solvers. For black-box ODE solvers, we pick out an adaptive-step and a fixed-step neural ODE solver implemented by [40], that is, Runge-Kutta of order 5 of Dormand-Prince-Shampine (dopri5) and Fourth-order Runge-Kutta with 3/8 rule (rk4). As shown in Figure 2, The EM solver fails to converge under 200 NFE, and black-box ODE solvers can obtain acceptable quality at around 50 NFE. The GDPMS displays clear superiority in the range below 50 NFE. Notably, the 1st-order GDPMS still generates reasonable molecular graphs with 10 NFE. For the running time comparison, CDGS equipped with GDPMS takes much less time compared to autoregressive GraphAF and GraphDF, and makes an obvious improvement towards GDSS. MoFlow spends the least time but fails to generate high-fidelity samples according to Table 1. In conclusion, benefiting from the framework design and the ODE solvers utilizing the semi-linear structure, we achieve great advancement in fast sampling for complex molecular graphs.



Method	GraphAF	GraphDF	MoFlow	GDSS	CDGS-50	CDGS-30	CDGS-10
Time (s)	$2.89e^2$	$3.19e^3$	1.54	$1.42e^2$	$3.79e^1$	$2.38e^1$	8.38

Figure 2: (Up) Few-step molecular graph sampling results for various numerical solvers. (Down) The wall-clock time taken to generate 512 molecular graphs.

D.1.1 Ablation Studies

We conduct ablation analysis on the ZINC250k dataset to verify the effectiveness of our framework. In Figure 3, we report the results using GDPMS with 50 NFE to alleviate the disturbance from the stochastic term in the EM solver, faithfully reflecting the model ability. Taking CDGS with 64 hidden dimensions (**64ch**) as reference, we first remove the discrete graph structure related components and remain with the attention part (**ATTN**), then further remove the edge existence variable (**W-ADJ**). The variant using GINE without attention layers is denoted as **GINE**. It can be observed that the attention part is crucial for distribution learning and that discrete graph structure greatly helps to capture the chemical valency rule. Without specialised discrete graph structure modelling, separating the edge existence in the formulation still makes contributions to molecule validity. In addition, it is necessary to increase hidden dimensions (**128ch**, **256ch**) for better handling the complexity of drug-like molecules on the ZINC250k dataset.

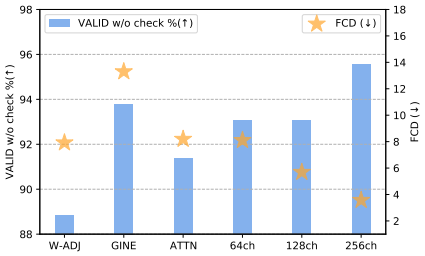


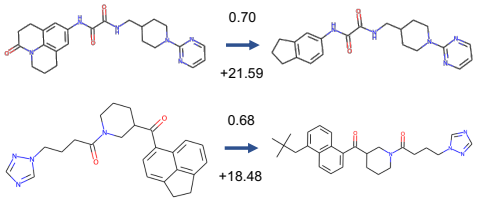
Figure 3: Ablation on ZINC250k.

D.1.2 Similarity-constrained Property Optimization

We also demonstrate the usefulness of our diffusion framework for the task of similarity-constrained property optimization. Following [4, 6], we select 800 molecules with low p-logP scores (*i.e.*, the octanol-water partition coefficients penalized by synthetic accessibility and number of long cycles) as the initial molecules for optimization. We aim to generate new molecules with a higher p-logP while keeping similarity to the original molecules with a threshold δ. The similarity metric is defined as Tanimoto similarity with Morgan fingerprints [41]. The property predictor is composed of 6 hybrid message passing blocks with RGCN [42] as the non-attention layer for differentiation. We pretrain the predictor on the ZINC250k dataset for 200 epochs. Each initial molecular graph is encoded into latent codes at the middle time $t_\xi = 0.3$ through the forward-time ODE solver. After 50 gradient ascent steps, all latent codes are decoded back to molecules with another gradient-guided reverse-time ODE solver. This procedure is repeated 20 times with a different number of atoms to search for the highest property molecule that satisfies the similarity constraint. Results for the similarity-constrained optimization are summarized in Table 2. **GraphAF-RL** is the representative method combined with reinforcement learning, **MoFlow** is a flow-based method, and **GraphEBM** is an energy-based method for molecule optimization. With the similarity constraint ($\delta > 0$), CDGS outperforms MoFlow and GraphEBM in terms of success rate and mean property improvement, showing competitive performance to the RL-based method. Since RL-based methods require heavy property evaluator calls, which is unrealistic in some optimization scenarios, our CDGS could serve as a useful supplement for drug discovery tasks.

Table 2: Similarity-constrained molecule property optimization performance. The values above and below arrows in visualizations denote similarity scores and improvements.

	GraphAF-RL		MoFlow	
δ	**Improvement**	**Success**	**Improvement**	**Success**
0.0	13.13±6.89	100%	8.61±5.44	99%
0.2	11.90±6.86	100%	7.06±5.04	97%
0.4	8.21±6.51	100%	4.71±4.55	86%
0.6	4.98±6.49	97%	2.10±2.86	58%
	GraphEBM		CDGS	
δ	**Improvement**	**Success**	**Improvement**	**Success**
0.0	15.75±7.40	99%	12.83±7.01	100%
0.2	8.40±6.38	94%	11.70±6.84	100%
0.4	4.95±5.90	79%	9.56±6.33	100%
0.6	3.15±5.08	45%	5.10±5.80	98%



D.2 Generic Graph Generation

D.2.1 Experimental Setup

To display the ability of graph structure distribution learning, we validate CDGS on four common generic graph datasets with various graph sizes and characteristics: (1) *Community-small*, 100 two-community graphs generated by the Erdős-Rényi model (E-R) [1] with $p = 0.7$, (2) *Ego-small*, 200 one-hop ego graphs extracted from Citeseer network [43], (3) *Enzymes*, 563 protein graphs with more

Table 3: Generation performance on generic graph datasets. The better results are indicated by a closer value with the performance of training graphs, and the best results are in bold.

	Community-small				Ego-small				Enzymes				Ego			
	$\lvert V\rvert_{max} = 20, \lvert E\rvert_{max} = 62$				$\lvert V\rvert_{max} = 17, \lvert E\rvert_{max} = 66$				$\lvert V\rvert_{max} = 125, \lvert E\rvert_{max} = 149$				$\lvert V\rvert_{max} = 399, \lvert E\rvert_{max} = 1071$			
	$\lvert V\rvert_{avg} \approx 15, \lvert E\rvert_{avg} \approx 36$				$\lvert V\rvert_{avg} \approx 6, \lvert E\rvert_{avg} \approx 9$				$\lvert V\rvert_{avg} \approx 33, \lvert E\rvert_{avg} \approx 63$				$\lvert V\rvert_{avg} \approx 145, \lvert E\rvert_{avg} \approx 335$			
	Deg.	Clus.	Spec.	GIN.	Deg.	Clus.	Spec.	GIN.	Deg.	Clus.	Spec.	GIN.	Deg.	Clus.	Spec.	GIN.
Train	*0.035*	*0.067*	*0.045*	*0.037*	*0.025*	*0.029*	*0.027*	*0.016*	*0.011*	*0.011*	*0.011*	*0.007*	*0.009*	*0.009*	*0.009*	*0.005*
ER	0.300	0.239	0.100	0.278	0.200	0.094	0.361	0.230	0.844	0.381	0.104	0.808	0.738	0.397	0.868	0.118
VGAE	0.391	0.257	0.095	0.360	0.146	0.046	0.249	0.089	0.811	0.514	0.153	0.716	0.873	1.210	0.935	0.520
GraphRNN	0.106	0.115	0.091	0.353	0.155	0.229	0.167	0.472	0.397	0.302	0.260	1.495	0.140	0.755	0.316	1.283
GraphRNN-U	0.410	0.297	0.103	0.970	0.471	0.416	0.398	0.915	0.932	1.000	0.367	1.263	1.413	1.097	1.110	1.317
GRAN	0.125	0.164	0.111	0.196	0.096	0.072	0.095	0.106	0.215	0.147	0.034	0.069	0.594	0.425	1.025	0.244
GRAN-U	0.106	0.127	0.083	0.164	0.155	0.229	0.167	0.094	0.343	0.122	0.041	0.242	0.099	0.170	0.179	0.128
EDP-GNN	0.100	0.140	0.085	0.125	0.026	0.032	0.037	0.031	0.120	0.644	0.070	0.119	0.553	0.605	0.374	0.295
GDSS	0.102	0.125	0.087	0.137	0.041	0.036	0.041	0.041	0.118	0.071	0.053	0.028	0.314	0.776	0.097	0.156
CDGS-EM	**0.052**	**0.080**	**0.064**	**0.062**	**0.025**	**0.031**	**0.033**	**0.025**	**0.048**	**0.070**	**0.033**	**0.024**	**0.036**	**0.075**	**0.026**	**0.026**
CDGS-GDPMS-30	0.100	0.121	0.084	0.120	0.116	0.064	0.141	0.052	0.140	0.127	0.041	0.040	0.157	0.109	0.153	0.064

than 10 nodes from BRENDA database [44], (4) *Ego*, 757 three-hop ego graphs also extracted from Citeseer network [43]. We use $8:2$ as the split ratio for train/test. We generate 1024 graph samples for evaluation on Community-small and Ego-small, and generate the same number of graphs as the test set on Enzymes and Ego. We follow the advice from [45] to evaluate the distribution of discrete graph structures. Three graph-level structure descriptor functions are selected: the degree distribution (**Deg.**), the clustering coefficient distribution (**Clus.**) and the Laplacian spectrum histograms (**Spec.**). We use MMD with the radial basis function kernel (RBF) to calculate the distance on features extracted by graph descriptors. To accurately evaluate distribution distance, different from [46, 47, 21] using a static smoothing hyperparameter for MMD, we provide a set of parameters and report the largest distance like [48]. We also consider a well-established comprehensive neural-based metric (**GIN.**) from [48].

D.2.2 Baselines

Apart from scored-based models (EDP-GNN and GDSS), we compare CDGS with a classical method (**ER** [1]), a VAE-based method (**VGAE** [49]), and two strong autoregressive graph generative models (**GraphRNN** [46], **GRAN** [47]). **GraphRNN-U** and **GRAN-U** are trained with uniform node orderings to alleviate the bias from specific ordering strategies.

D.2.3 Sampling Quality

Table 3 displays that, among four datasets, CDGS consistently achieves better performance than score-based models and autoregressive models. Especially for the large Ego dataset, CDGS still generates graphs with high fidelity while the diffusion-based GDSS fails in Deg. and Clus. metrics. The GDPMS is also supported for quick graph structure generation with acceptable quality. Thanks to the appropriate framework design and the emphasis on evolving discrete graph structures during the generative process, CDGS effectively captures the underlying distribution of graph topology.

E Experimental Details

E.1 Hyperparameters

The hyperparameters used for our CDGS in the experiments are provided in Table 5. In particular, we set the SDE to the default parameters of Variance Preserving SDE (VPSDE) without any sweeping, keeping the small signal-to-noise ratio at $\boldsymbol{G_T}$. Different from GDSS [9], we adopt the unified SDE setting for $\boldsymbol{X}$ and $\boldsymbol{A}$ and utilize the simple EM solver, avoiding complex parameter tuning.

E.2 Molecular Graph Generation

The dataset information is summarized in Table 4.

Table 4: Molecular dataset information.

Dataset	Number of molecules	Number of nodes	Number of node types	Number of edge types
ZINC250k	249,455	$6 \leq \lvert V \rvert \leq 38$	9	3
QM9	133,885	$1 \leq \lvert V \rvert \leq 9$	4	3

Table 5: Hyperparameters of CDGS used in graph generation experiments.

	Hyperparameter	ZINC250k	QM9	Community-small	Ego-small	Enzymes	Ego
Data	Edge initial scale	$[-1.0, 1.0]$	$[-1.0, 1.0]$	$[-1.0, 1.0]$	$[-1.0, 1.0]$	$[-1.0, 1.0]$	$[-1.0, 1.0]$
	Node initial scale	$[-0.5, 0.5]$	$[-0.5, 0.5]$	-	-	-	-
ϵ_θ	Number of message passing blocks	10	6	6	3	6	3
	Hidden dimension	256	64	64	64	64	64
	Number of attention heads	8	8	8	8	8	8
	Number of Random Walks	20	8	16	8	24	20
SDE	Type	VP	VP	VP	VP	VP	VP
	Number of EM sampling steps	1000	1000	1000	1000	1000	1000
	β_{min}	0.1	0.1	0.1	0.1	0.1	0.1
	β_{max}	20.0	20.0	20.0	20.0	20.0	20.0
Train	Optimizer	Adam	Adam	Adam	Adam	Adam	Adam
	Learning rate	1e-4	1e-4	1e-4	1e-4	1e-4	2e-4
	Batch size	64	128	64	64	48	8
	Number of training steps	1.25M	1.0M	1.0M	0.8M	1.0M	0.8M
	EMA	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999

E.3 Molecular Graph Generation Setup

E.3.1 Metrics

Fréchet ChemNet Distance (FCD) [50] calculates the distance between the reference molecule set and the generated set with the activations of the penultimate layer of ChemNet. Lower FCD values indicate higher similarity between the two distributions. **Neighborhood subgraph pairwise distance kernel (NSPDK)** is the distance measured by mean maximum discrepancy (MMD), which incorporates node and edge features along with the underlying graph structure. FCD and NSPDK, one from the perspective of molecules and the other from the perspective of graphs, are crucial for the evaluation of molecular graph distribution learning [9]. **VALID w/o check** is the percentage of valid molecules without post-hoc valency correction. Here, we follow the setting of [6, 9] to consider the formal charges for valency checking. We also report the results of three metrics that are used commonly but have obvious marginal effects, *i.e.*, the ratio of valid molecules (**VALID**), the ratio of unique molecules (**UNIQUE**), and the ratio of novel molecules with reference to the training set (**NOVEL**).

E.3.2 Implementation Details

We train and evaluate models on two molecule datasets, ZINC250k [51] and QM9 [52]. Before converting to graphs, all molecules are processed to the kekulized form using RDKit [53], where hydrogen atoms are removed and aromatic bonds are replaced by double bonds. We evaluate generation quality on $10,000$ generated molecules with the following widely used metrics.

For each molecule, we represent it with one-hot atom types $\{0,1\}^{N\times F}$, ordinal edge types $\{0,1,2,3\}^{N\times N}$ (*i.e.*, single, double, or triple bonds) and edge existence $\{0,1\}^{N\times N}$. We convert these variables to real numbers and obtain $\boldsymbol{G} = (\boldsymbol{X}, \boldsymbol{A})$. Scaling and shifting are also used to adjust the initial number scale, making them simpler for neural networks to process. As our method focuses on undirected graphs, we keep the adding noise and the output of edges symmetrical. We first sample the number of atoms from the probability mass function on the training graphs' atom number before the reverse generative process. After sampling through numerical solvers, we first move and shift the matrices back to their original scale and make quantization to obtain graph samples. We remain the biggest connected-subgraphs for those molecular graphs that are disconnected. The valency

correction procedure from [6] are adopted to further ensure molecular validity. As for baselines, we report the performance from [9], and re-sample or retrain GDSS with its official code.

E.4 Generic Graph Generation

E.4.1 Implementation Details

We directly use adjacency matrices $\{0,1\}^{N\times N}$ to represent generic graphs. We still convert variables to real numbers and adjust their scale. For the MMD metrics (Deg., Clus., and Spec.) used in graph structure distribution evaluation, we choose a efficient positive definite kernel function, *i.e.*, an RBF kernel with a smoothing parameter υ denoted as

$$k(x_i, x_j) = \exp(\frac{-||x_i - x_j||^2}{2\upsilon^2}). \tag{12}$$

It is important to choose υ to accurately measure the distribution distance. We report the largest MMD values using a set of υ parameters. 50 candidate $\log \upsilon$ values are selected evenly between $[10^{-5}, 10^5]$. We take 100 bins for the histogram conversion of clustering coefficient and 200 bins for the conversion of Laplacian spectrum.

As for the baselines, ER [1] is implemented by the edge probability estimated by maximum likelihood on training graphs. VGAE [49] is a variational auto-encoder implemented by a graph convolution network encoder and a simple MLP decoder with inner product computation for edge existence. For GraphRNN [46], GRAN [47], and EDPGNN [21], we utilize their official code to train the models with the same data split and generate graphs for evaluation.

F Algorithms of GDPM-Solvers

We provide the implementation details of fast ODE solvers of different orders for in Algorithm 1, 2, 3, mainly derived from [25]. The solvers can be equipped with the gradient guidance from time-dependent molecule property predictor conveniently like Algorithm 4.

Algorithm 1 Graph DPM-Solver 1

Require: initial graph $\boldsymbol{G_T} = (\boldsymbol{X_T}, \boldsymbol{A_T})$, time step schedule $\{t_i\}_{i=0}^M$, graph noise prediction model $\boldsymbol{\epsilon_\theta}$, quantized function $quantize(\cdot)$, post-processing function $post(\cdot)$

def GDPMS-1($\tilde{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{A}}_{t_{i-1}}, t_{i-1}, t_i$)
  $h_i \leftarrow \lambda_{t_i} - \lambda_{t_{i-1}}$
  $\bar{\boldsymbol{A}}'_{t_{i-1}} \leftarrow quantize(\tilde{\boldsymbol{A}}_{t_{i-1}})$
  $\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{t_{i-1}} \leftarrow \boldsymbol{\epsilon_\theta}((\tilde{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{A}}_{t_{i-1}}), \bar{\boldsymbol{A}}'_{t_{i-1}}, t_{i-1})$
  $\tilde{\boldsymbol{X}}_{t_i} \leftarrow \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{X}}_{t_{i-1}} - \sigma_{t_i}(e^{h_i} - 1)\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{t_{i-1}}$
  $\tilde{\boldsymbol{A}}_{t_i} \leftarrow \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{A}}_{t_{i-1}} - \sigma_{t_i}(e^{h_i} - 1)\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{t_{i-1}}$
  return $\tilde{\boldsymbol{X}}_{t_i}, \tilde{\boldsymbol{A}}_{t_i}$
$\tilde{\boldsymbol{X}}_{t_0}, \tilde{\boldsymbol{A}}_{t_0} \leftarrow \boldsymbol{X_T}, \boldsymbol{A_T}$
for $i \leftarrow 1$ to M **do**
  $\tilde{\boldsymbol{X}}_{t_i}, \tilde{\boldsymbol{A}}_{t_i} \leftarrow$ GDPMS-1($\tilde{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{A}}_{t_{i-1}}, t_{i-1}, t_i$)
return $post(\tilde{\boldsymbol{X}}_{t_M}, \tilde{\boldsymbol{A}}_{t_M})$

G Visualization

We visualize the reverse generative process on the QM9 dataset in Figure 4. We provides the visualization of generated graphs on different datasets: ZINC250k (in Figure 5), QM9 (in Figure 6), Enzymes (in Figure 7), Ego (in Figure 8), and Community-small (in Figure 9).

Algorithm 2 Graph DPM-Solver 2

Require: initial graph $\boldsymbol{G_T} = (\boldsymbol{X_T}, \boldsymbol{A_T})$, time step schedule $\{t_i\}_{i=0}^{M}$, graph noise prediction model $\boldsymbol{\epsilon_\theta}$, quantized function $quantize(\cdot)$, post-processing function $post(\cdot)$, $r_1 = 0.5$

def GDPMS-2($\tilde{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{A}}_{t_{i-1}}, t_{i-1}, t_i, r_1$)
  $h_i \leftarrow \lambda_{t_i} - \lambda_{t_{i-1}}$
  $s_i \leftarrow t_\lambda(\lambda_{t_{i-1}} + r_1 h_i)$
  $\bar{\boldsymbol{A}}'_{t_{i-1}} \leftarrow quantize(\tilde{\boldsymbol{A}}_{t_{i-1}})$
  $\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{t_{i-1}},\ \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{t_{i-1}} \leftarrow \boldsymbol{\epsilon_\theta}((\tilde{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{A}}_{t_{i-1}}), \bar{\boldsymbol{A}}'_{t_{i-1}}, t_{i-1})$
  $\boldsymbol{u}^{\boldsymbol{X}}_i \leftarrow \frac{\alpha_{s_i}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{X}}_{t_{i-1}} - \sigma_{s_i}(e^{r_1 h_i} - 1)\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{t_{i-1}}$
  $\boldsymbol{u}^{\boldsymbol{A}}_i \leftarrow \frac{\alpha_{s_i}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{A}}_{t_{i-1}} - \sigma_{s_i}(e^{r_1 h_i} - 1)\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{t_{i-1}}$
  $\boldsymbol{u}^{\bar{\boldsymbol{A}}}_i \leftarrow quantize(\boldsymbol{u}^{\boldsymbol{A}}_i)$
  $\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{s_i},\ \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{s_i} \leftarrow \boldsymbol{\epsilon_\theta}((\boldsymbol{u}^{\boldsymbol{X}}_i, \boldsymbol{u}^{\boldsymbol{A}}_i), \boldsymbol{u}^{\bar{\boldsymbol{A}}}_i, s_i)$
  $\tilde{\boldsymbol{X}}_{t_i} \leftarrow \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{X}}_{t_{i-1}} - \sigma_{t_i}(e^{h_i} - 1)\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{t_{i-1}} - \frac{\sigma_{t_i}}{2r_i}(e^{h_i} - 1)(\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{s_i} - \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{t_{i-1}})$
  $\tilde{\boldsymbol{A}}_{t_i} \leftarrow \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{A}}_{t_{i-1}} - \sigma_{t_i}(e^{h_i} - 1)\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{t_{i-1}} - \frac{\sigma_{t_i}}{2r_i}(e^{h_i} - 1)(\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{s_i} - \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{t_{i-1}})$
  return $\tilde{\boldsymbol{X}}_{t_i}, \tilde{\boldsymbol{A}}_{t_i}$
$\tilde{\boldsymbol{X}}_{t_0}, \tilde{\boldsymbol{A}}_{t_0} \leftarrow \boldsymbol{X_T}, \boldsymbol{A_T}$
for $i \leftarrow 1$ to M **do**
  $\tilde{\boldsymbol{X}}_{t_i}, \tilde{\boldsymbol{A}}_{t_i} \leftarrow$ GDPMS-2($\tilde{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{A}}_{t_{i-1}}, t_{i-1}, t_i, r_1$)
return $post(\tilde{\boldsymbol{X}}_{t_M}, \tilde{\boldsymbol{A}}_{t_M})$

Algorithm 3 Graph DPM-Solver 3

Require: initial graph $\boldsymbol{G_T} = (\boldsymbol{X_T}, \boldsymbol{A_T})$, time step schedule $\{t_i\}_{i=0}^{M}$, graph noise prediction model $\boldsymbol{\epsilon_\theta}$, quantized function $quantize(\cdot)$, post-processing function $post(\cdot)$, $r_1 = \frac{1}{3}$, $r_2 = \frac{2}{3}$

def GDPMS-3($\tilde{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{A}}_{t_{i-1}}, t_{i-1}, t_i, r_1, r_2$)
  $h_i \leftarrow \lambda_{t_i} - \lambda_{t_{i-1}}$
  $s_{2i-1} \leftarrow t_\lambda(\lambda_{t_{i-1}} + r_1 h_i), \quad s_{2i} \leftarrow t_\lambda(\lambda_{t_{i-1}} + r_2 h_i)$
  $\bar{\boldsymbol{A}}'_{t_{i-1}} \leftarrow quantize(\tilde{\boldsymbol{A}}_{t_{i-1}})$
  $\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{t_{i-1}},\ \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{t_{i-1}} \leftarrow \boldsymbol{\epsilon_\theta}((\tilde{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{A}}_{t_{i-1}}), \bar{\boldsymbol{A}}'_{t_{i-1}}, t_{i-1})$
  $\boldsymbol{u}^{\boldsymbol{X}}_{2i-1} \leftarrow \frac{\alpha_{s_{2i-1}}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{X}}_{t_{i-1}} - \sigma_{s_{2i-1}}(e^{r_1 h_i} - 1)\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{t_{i-1}}$
  $\boldsymbol{u}^{\boldsymbol{A}}_{2i-1} \leftarrow \frac{\alpha_{s_{2i-1}}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{A}}_{t_{i-1}} - \sigma_{s_{2i-1}}(e^{r_1 h_i} - 1)\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{t_{i-1}}$
  $\boldsymbol{u}^{\bar{\boldsymbol{A}}}_{2i-1} \leftarrow quantize(\boldsymbol{u}^{\boldsymbol{A}}_{2i-1})$
  $\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{s_{2i-1}},\ \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{s_{2i-1}} \leftarrow \boldsymbol{\epsilon_\theta}((\boldsymbol{u}^{\boldsymbol{X}}_{2i-1}, \boldsymbol{u}^{\boldsymbol{A}}_{2i-1}), \boldsymbol{u}^{\bar{\boldsymbol{A}}}_{2i-1}, s_{2i-1})$
  $\boldsymbol{D}^{\boldsymbol{X}}_{2i-1} \leftarrow \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{s_{2i-1}} - \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{t_{i-1}}, \quad \boldsymbol{D}^{\boldsymbol{A}}_{2i-1} \leftarrow \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{s_{2i-1}} - \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{t_{i-1}}$
  $\boldsymbol{u}^{\boldsymbol{X}}_{2i} \leftarrow \frac{\alpha_{s_{2i}}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{X}}_{t_{i-1}} - \sigma_{s_{2i}}(e^{r_2 h_i} - 1)\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{t_{i-1}} - \frac{\sigma_{s_{2i}} r_2}{r_1}(\frac{e^{r_2 h_i} - 1}{r_2 h_i} - 1)\boldsymbol{D}^{\boldsymbol{X}}_{2i-1}$
  $\boldsymbol{u}^{\boldsymbol{A}}_{2i} \leftarrow \frac{\alpha_{s_{2i}}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{A}}_{t_{i-1}} - \sigma_{s_{2i}}(e^{r_2 h_i} - 1)\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{t_{i-1}} - \frac{\sigma_{s_{2i}} r_2}{r_1}(\frac{e^{r_2 h_i} - 1}{r_2 h_i} - 1)\boldsymbol{D}^{\boldsymbol{A}}_{2i-1}$
  $\boldsymbol{u}^{\bar{\boldsymbol{A}}}_{2i} \leftarrow quantize(\boldsymbol{u}^{\boldsymbol{A}}_{2i})$
  $\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{s_{2i}},\ \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{s_{2i}} \leftarrow \boldsymbol{\epsilon_\theta}((\boldsymbol{u}^{\boldsymbol{X}}_{2i}, \boldsymbol{u}^{\boldsymbol{A}}_{2i}), \boldsymbol{u}^{\bar{\boldsymbol{A}}}_{2i}, s_{2i})$
  $\boldsymbol{D}^{\boldsymbol{X}}_{2i} \leftarrow \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{s_{2i}} - \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{t_{i-1}}, \quad \boldsymbol{D}^{\boldsymbol{A}}_{2i} \leftarrow \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{s_{2i}} - \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{t_{i-1}}$
  $\tilde{\boldsymbol{X}}_{t_i} \leftarrow \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{X}}_{t_{i-1}} - \sigma_{t_i}(e^{h_i} - 1)\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{t_{i-1}} - \frac{\sigma_{t_i}}{r_i}(\frac{e^{h_i} - 1}{h} - 1)\boldsymbol{D}^{\boldsymbol{X}}_{2i}$
  $\tilde{\boldsymbol{A}}_{t_i} \leftarrow \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}}\tilde{\boldsymbol{A}}_{t_{i-1}} - \sigma_{t_i}(e^{h_i} - 1)\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{t_{i-1}} - \frac{\sigma_{t_i}}{r_i}(\frac{e^{h_i} - 1}{h} - 1)\boldsymbol{D}^{\boldsymbol{A}}_{2i}$
  return $\tilde{\boldsymbol{X}}_{t_i}, \tilde{\boldsymbol{A}}_{t_i}$
$\tilde{\boldsymbol{X}}_{t_0}, \tilde{\boldsymbol{A}}_{t_0} \leftarrow \boldsymbol{X_T}, \boldsymbol{A_T}$
for $i \leftarrow 1$ to M **do**
  $\tilde{\boldsymbol{X}}_{t_i}, \tilde{\boldsymbol{A}}_{t_i} \leftarrow$ GDPMS-3($\tilde{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{A}}_{t_{i-1}}, t_{i-1}, t_i, r_1, r_2$)
return $post(\tilde{\boldsymbol{X}}_{t_M}, \tilde{\boldsymbol{A}}_{t_M})$

Algorithm 4 Graph DPM-Solver 1 with gradient guidance

Require: initial graph $\boldsymbol{G_T} = (\boldsymbol{X_T}, \boldsymbol{A_T})$, time step schedule $\{t_i\}_{i=0}^{M}$, graph noise prediction model $\boldsymbol{\epsilon_\theta}$, quantized function $quantize(\cdot)$, post-processing function $post(\cdot)$, property predictor $\boldsymbol{R_\psi}$, guidance weight r

def GDPMS-1-GUIDE($\tilde{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{A}}_{t_{i-1}}, t_{i-1}, t_i, r$)
  $h_i \leftarrow \lambda_{t_i} - \lambda_{t_{i-1}}$
  $\bar{\boldsymbol{A}}'_{t_{i-1}} \leftarrow quantize(\tilde{\boldsymbol{A}}_{t_{i-1}})$
  $\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{t_{i-1}} \leftarrow \boldsymbol{\epsilon_\theta}((\tilde{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{A}}_{t_{i-1}}), \bar{\boldsymbol{A}}'_{t_{i-1}}, t_{i-1})$
  $\boldsymbol{R}_{t_{i-1}} = \boldsymbol{R_\psi}((\tilde{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{A}}_{t_{i-1}}), t_{i-1})$
  $\tilde{\boldsymbol{X}}_{t_i} \leftarrow \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}} \tilde{\boldsymbol{X}}_{t_{i-1}} - \sigma_{t_i}(e^{h_i} - 1)(\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{X}}_{t_{i-1}} - r\sigma_{t_{i-1}} \nabla^*_{\boldsymbol{X}} \boldsymbol{R}_{t_{i-1}})$
  $\tilde{\boldsymbol{A}}_{t_i} \leftarrow \frac{\alpha_{t_i}}{\alpha_{t_{i-1}}} \tilde{\boldsymbol{A}}_{t_{i-1}} - \sigma_{t_i}(e^{h_i} - 1)(\tilde{\boldsymbol{\epsilon}}^{\boldsymbol{A}}_{t_{i-1}} - r\sigma_{t_{i-1}} \nabla^*_{\boldsymbol{A}} \boldsymbol{R}_{t_{i-1}}))$
  return $\tilde{\boldsymbol{X}}_{t_i}, \tilde{\boldsymbol{A}}_{t_i}$
$\tilde{\boldsymbol{X}}_{t_0}, \tilde{\boldsymbol{A}}_{t_0} \leftarrow \boldsymbol{X_T}, \boldsymbol{A_T}$
for $i \leftarrow 1$ to M **do**
  $\tilde{\boldsymbol{X}}_{t_i}, \tilde{\boldsymbol{A}}_{t_i} \leftarrow$ GDPMS-1-GUIDE($\tilde{\boldsymbol{X}}_{t_{i-1}}, \tilde{\boldsymbol{A}}_{t_{i-1}}, t_{i-1}, t_i, r$)
return $post(\tilde{\boldsymbol{X}}_{t_M}, \tilde{\boldsymbol{A}}_{t_M})$

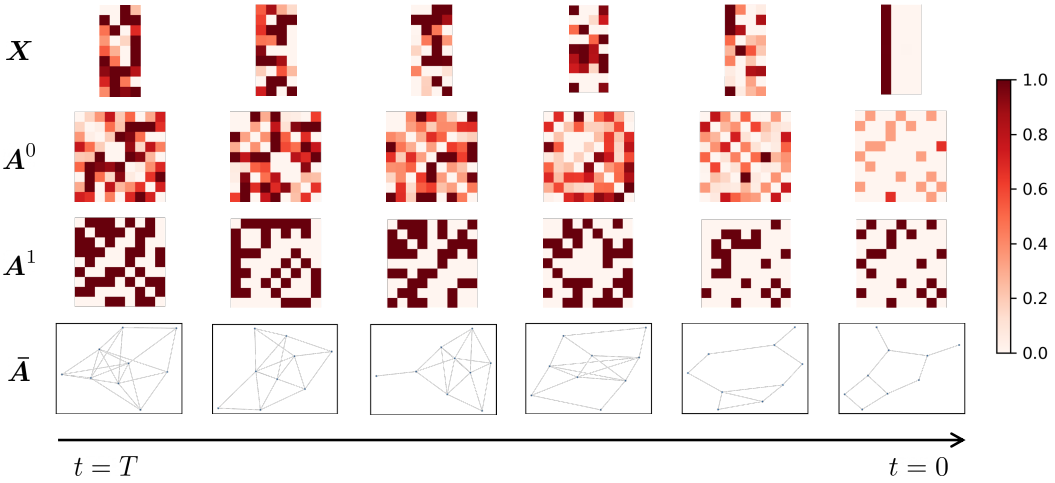


Figure 4: Molecular graph visualization at different steps in the reverse generative process from a model trained on QM9.

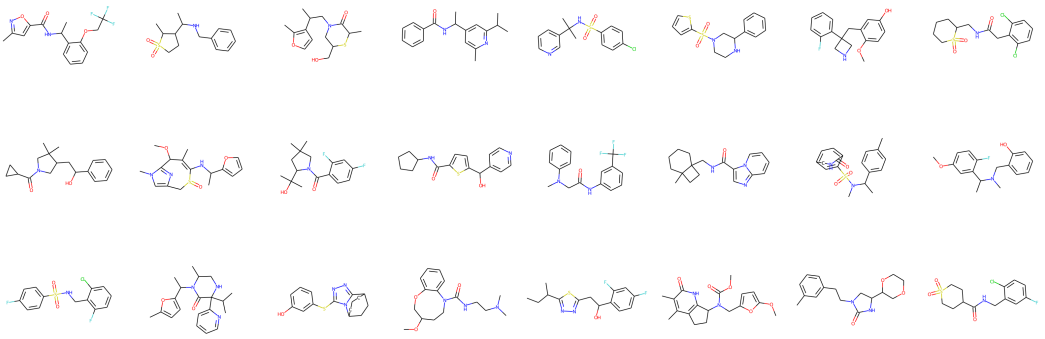


Figure 5: The generated samples from the model trained on the ZINC250k dataset.

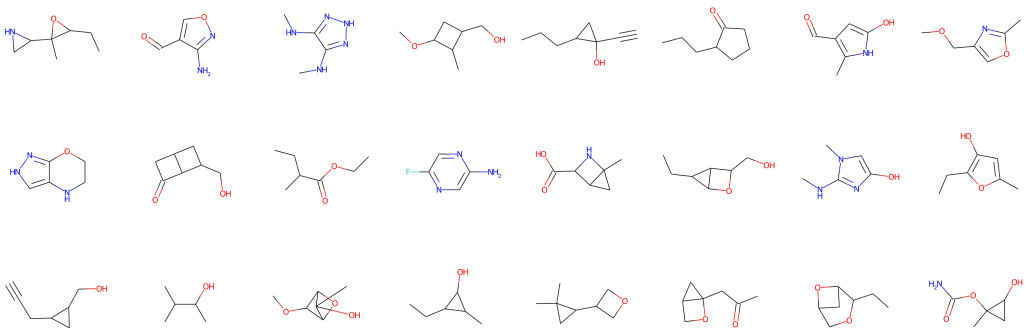


Figure 6: The generated samples from the model trained on the QM9 dataset.

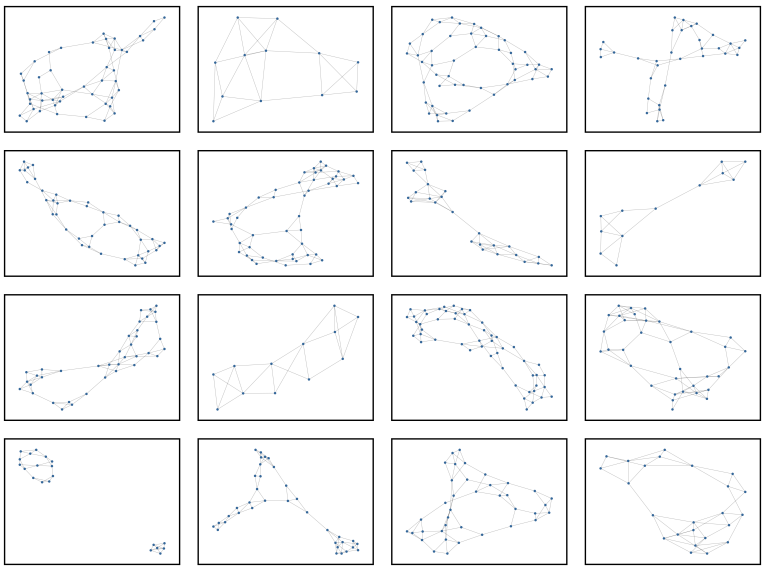


Figure 7: The generated samples from the model trained on the Enzymes dataset.

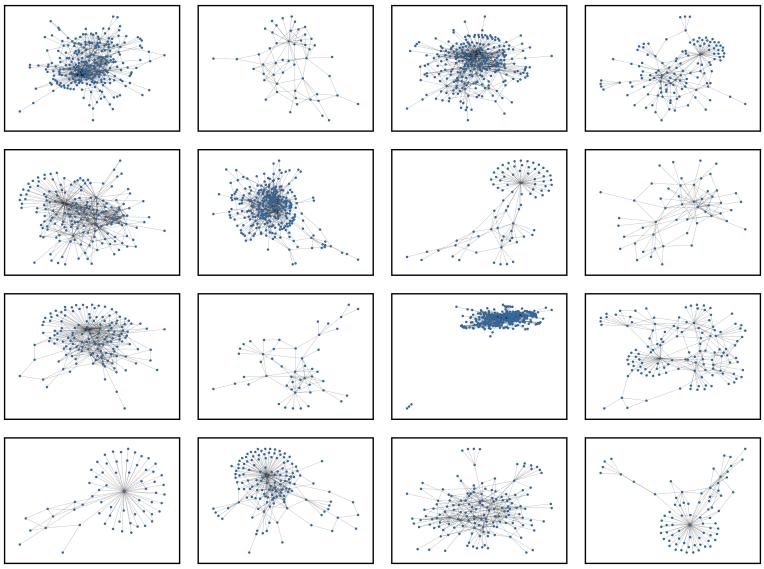


Figure 8: The generated samples from the model trained on the Ego dataset.

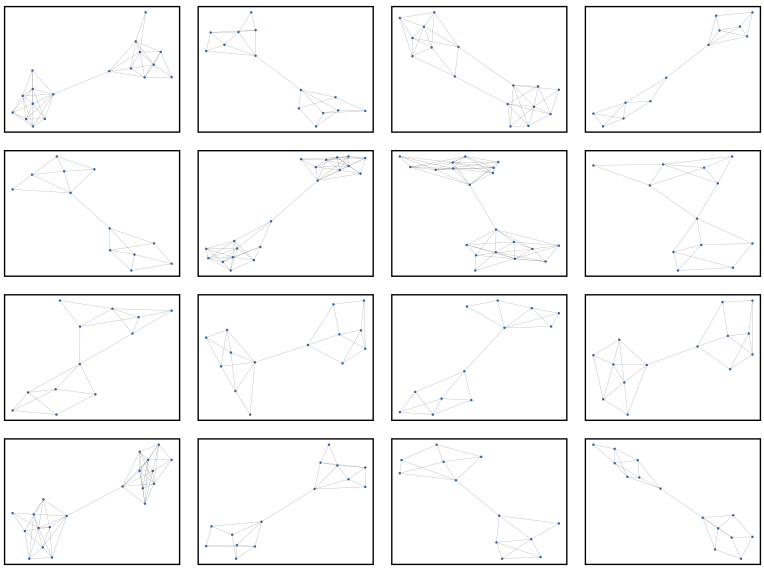


Figure 9: The generated samples from the model trained on the Community-small dataset.