Exploring the Design Space of Generative Diffusion Processes for Sparse Graphs

Anonymous Author(s)
Affiliation
Address
email

Abstract

We extend score-based generative diffusion processes (GDPs) to sparse graphs and other inherently discrete data, with a focus on scalability. GDPs apply diffusion to training samples, then learn a reverse process generating new samples out of noise. Previous work applying GDPs to discrete data effectively relax discrete variables to continuous ones. Our approach is different: we consider jump diffusion (*i.e.*, diffusion with punctual discontinuities) in $\mathbb{R}^d \times \mathcal{G}$ where $\mathcal{G}$ models discrete components of the data. We focus our attention on sparse graphs: our DISSOLVE process gradually breaks apart a graph $(V, E) \in \mathcal{G}$ in a certain number of distinct jump events. This confers significant advantages compared to GDPs that use less efficient representations and/or that destroy the graph information in a sudden manner. Gaussian kernels allow for efficient training with denoising score matching; standard GDP methods can be adapted with just an extra argument to the score function. We consider improvement opportunities for DISSOLVE and discuss necessary conditions to generalize to other kinds of inherently discrete data.

Generative diffusion processes (GDPs) [13] are a family of unsupervised methods that *pose* a forward diffusion process destroying the information contained in training data samples, then *learn* the reverse process (*i.e.*, backward in time) generating new samples out of noise. Although such techniques hinge heavily on the ability to continuously transform data instances, recent advances [1, 6, 9, 11, 12] extended the use of GDPs to domains involving discreteness (*e.g.*, text, graphs). In essence, the common strategy behind these works is to relax categorical variables to continuous ones so that a data sample may be represented as a point in real space. The present work challenges the scalability of that strategy for some classes of data – including large sparse graphs – and proposes an alternative where discrete information is explicitly part of a jump diffusion process' internal state.

For simplicity, we focus our attention on large undirected sparse graphs whose nodes and edges do not have any kind of inherent features. Given a training dataset composed of such graphs, our goal is to generate new graphs from the same distribution. At this stage, our main concerns are the time and space complexities for both training and inference: how scalable are GDPs for large sparse graphs? Our main contributions concerns two hurdles to such scalability: the sheer dimension of the real space in which diffusion takes place, hereafter called *representation complexity*; and the intricacies of "scenarios" that must be simultaneously considered, hereafter called *simultaneity complexity*. We are free to *pose* both how graphs are represented as well as what are the forward stochastic dynamics destroying training samples – the validity of this choice should be judged on how feasible it is to *learn* the corresponding backward process building graphs, and how scalable the method is overall.

As part of a tutorial-ish exposition, we consider different incarnations of graph GDP, building up toward more scalable ones. The first such GDP, BASE, captures the essential ideas behind Niu et al. [12] and Jo et al. [9]. Because it represents a graph as a continuous relaxation of its adjacency matrix, BASE's representational complexity scales as the square of the number of nodes which,

Submitted to Score-based Method Workshop at 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

for large sparse graphs, is highly inefficient. This motivates EXPLODE, which instead relaxes a sparse representation of the graph (*i.e.*, adjacency matrix in "COO" sparse format) and thus achieves much better representational complexity. However, learning the backward process in this sparse representation is subjected to simultaneity complexity issues. DISSOLVE eschews these problems by incrementally injecting small chunks of graph information into real space, instead of cramming it all in the initial condition. Although there are issues with DISSOLVE's time and space complexity as it is, we discuss viable paths towards improving it for reasonable classes of sparse graphs.

Mathematically, DISSOLVE is a jump diffusion process whose state at any given time is the joint of a point in real space and of a graph. During the time interval between two jumps, the graph component of this state remains fixed and diffusion takes place in real space: standard denoising score matching allows us to learn a score function, parameterized by the graph, that reverses this inter-jump diffusion. When a jump occurs, a chunk of information is removed from the graph and injected in real space: these jumps are posed so that they are simple to both detect and "undo" in the backward process. Overall, backward diffusion and backward jumps can be iteratively chained to generate new graphs. Appendices provide additional details and considers generalizing our core ideas beyond graphs.

Background on denoising score matching for score-based GDPs. We here consider GDPs from the score-based stochastic differential equation (SDE) perspective presented in Song et al. [14]. Our notation here distinguishes stochastic variables (capital letters) from the value they may take (lowercase letters); temporal dependency is indicated as a subscript t.

Let X be the stochastic variable associated with the data distribution: we have access to some training dataset providing a finite number of samples $x \sim X$, and our goal is to generate new samples $\hat{x} \sim X$. The first step is to *pose* a stochastic process $\{\mathbf{U}_t\}$ whose initial value $\mathbf{U}_0$ fully determine x but, for a sufficiently high time τ_{stop}, the mutual information $\mathrm{I}(X; \mathbf{U}_{\tau_{\text{stop}}})$ drops to zero. In practical terms, this means specifying both how the initial condition $\mathbf{U}_0$ may be obtained from x, as well as the SDE governing the forward evolution of the process. We usually make those choices such that we know a simple analytical expression for $p(\mathbf{U}_t = \mathbf{u}|X = x)$, the probability density to observe $\mathbf{u}$ at time t given that the input training sample was x (hereafter called *kernel*).

The next step is to *learn* how to reverse this stochastic process, *i.e.*, running it backward in time. It is known that forward SDE may be converted to a backward one provided that we can evaluate $\nabla_{\mathbf{u}} \ln p(\mathbf{U}_t = \mathbf{u})$ [14]. *A priori*, we may estimate $p(\mathbf{U}_t = \mathbf{u}) = \mathbb{E}_{x\sim X}\, p(\mathbf{U}_t = \mathbf{u}|X = x)$ by marginalizing the kernel over the training dataset, but this is not tractable in practice. Instead, we use the training dataset to learn the parameters $\boldsymbol{\theta}$ of the score function $\mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u})$ which itself approximates $\nabla_{\mathbf{u}} \ln p(\mathbf{U}_t = \mathbf{u})$ by minimizing the denoising score matching loss

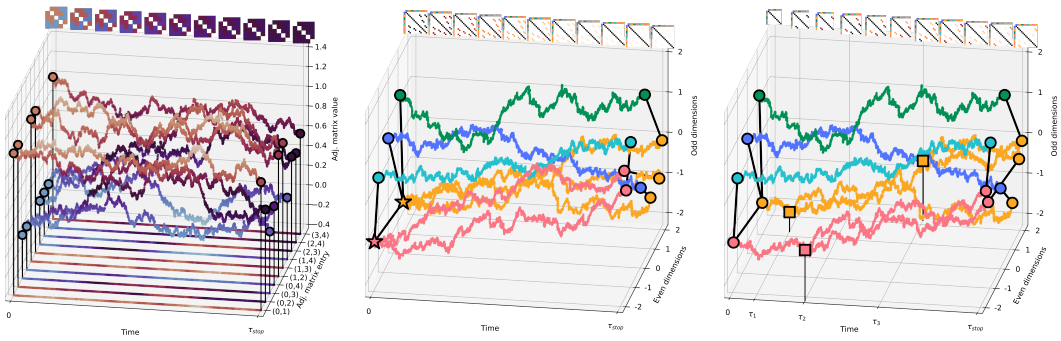
$$\mathcal{L}^{\boldsymbol{\theta}} = \mathbb{E}_{x\sim X}\, \mathbb{E}_{t\sim T}\, \lambda_t\, \mathbb{E}_{\mathbf{u}\sim \mathbf{U}_t|X=x} \left\| \mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u}) - \boldsymbol{\kappa}_t(\mathbf{u}|x) \right\|_2^2 , \tag{1}$$

where the *kernel score* $\boldsymbol{\kappa}_t(\mathbf{u}|x) = \nabla_{\mathbf{u}} p(\mathbf{U}_t = \mathbf{u}|X = x)$ is typically known analytically. Here λ_t is a positive weighting function for the expectation over $t \sim T$: as discussed in Song et al. [15], the choice of λ_t and of the distribution of the stochastic variable T are complementary levers toward the dual goal of managing the loss' training variance while balancing the errors at different times.

***Status quo*: the BASE process.** We now restrict our attention to the case where the training dataset is composed of undirected graphs without any kind of node nor edge features. More precisely, a sample $x \sim X$ takes the form $x = (V, E)$, where $V = \{0, 1, \cdots, N-1\}$ (*i.e.*, the first N contiguous non-negative integers) and $E \subseteq \{\{i, i'\}|i, i' \in V, i \neq i'\}$. We define the BASE GDP so that it represents such graphs as the continuous relaxation of their adjacency matrix, which is the main ideas behind Niu et al. [12] and Jo et al. [9] in the context of our present discussion.

Concretely, for a graph $(V, E) \sim X$ with N nodes, we pose the stochastic process $\{\mathbf{U}_t\}$ in $\mathbb{R}^{\frac{1}{2}N(N-1)}$ so that, for $i < i'$, the initial condition's entry $[\mathbf{U}_0]_{\frac{1}{2}i(i-1)+i'}$ is 1 if $\{i, i'\} \in E$ and 0 otherwise, which amounts to "unrolling" the upper-triangular part of the (symmetric) adjacency matrix. We should then pose the SDE specifying BASE's forward dynamics, then decide on the architecture of the neural network for the score function $\mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u})$ (preferably taking advantage of the permutation invariance inherent to graphs), then finally train this network by minimizing $\mathcal{L}^{\boldsymbol{\theta}}$ in Eq. (1).

However, for our present needs, it suffices to notice that BASE's representation complexity, $\mathcal{O}(N^2)$, is highly inefficient for large sparse graphs (*i.e.*, $|E| \ll N^2$). Indeed, the proportion of nonzero entries in $\mathbf{U}_0$ would tend toward zero as larger sparse graphs are considered, and the space and time



(a) BASE (b) EXPLODE with $\delta = 2$ (c) DISSOLVE with $\delta = 2$

Figure 1: **Sample path of graph GDPs for** $x = (\{0, 1, 2, 3, 4\}, \{\{0, 1\}, \{1, 2\}, \{1, 3\}, \{3, 4\}\})$**.** Each of these plots may be read left-to-right (forward time, destroying a training sample graph) or right-to-left (backward time, generating a graph). In each case, multiple curves are used to represent *a single multi-dimensional realization of the process*. On the top of each plot, the expectation over many such realizations is shown for the corresponding adjacency (first) or covariance (two others) matrix. (a) For BASE, the 10 entries in the upper triangular part of the adjacency matrix experience diffusion. After a certain time, one cannot say whether an edge was initially present or not. (b) In EXPLODE, the graph's 5 nodes are each randomly embedded in $\delta = 2$ dimensions, and 8 particles are initialized at the endpoints of the corresponding 4 edges. Particle trajectories quickly "explode" away from nodes with more than one edge (star markers). Taken individually, a particle's position carries no information whatsoever. With time, diffusion destroys the covariances among particles that initially encoded the graph structure. (c) In effect, DISSOLVE "chunks" EXPLODE's initial explosion into multiple "split" events (square markers). Each such split creates a copy of the splitting particle at the same location, and both particles split the original's edges – each taking at least one. The resulting covariance matrix has "less going on" than EXPLODE does at any particular time of the process. **Remark**: the limit where all splittings occur simultaneously at $t = 0$ exactly recovers EXPLODE.

complexity of the overall method would suffer accordingly. To be clear, the use of BASE-like GDPs may be very appropriate in situations involving small and/or dense graphs. The problem is that there are plenty of potential use cases involving large sparse graphs (*e.g.*, social networks, knowledge graphs, relational databases, computer code), which justify an alternative approach.

Naive fix: the EXPLODE process. Sparse data warrants sparse representations: instead of tracking a matrix whose entries are almost all zero, it may be better to just list where the nonzero entries are (*i.e.*, same idea as the "COO" sparse format). Indeed, the set of edges E can be represented as a two-columns table whose $|E|$ rows each specify the two nodes involved in the corresponding edge, *i.e.*, $V^{|E|\times 2}$. Each entry of that table is a categorical variable with $N = |V|$ possible values: how should we represent those in a manner that plays well with neural networks?

A categorical variable $i \in V$ could be represented as some variations of "one-hot encoding": a vector $\mathbb{R}^{|V|}$ whose i-th entry is 1, all the other ones are zero. However, representing each of our table's variables as such would give $\mathbb{R}^{|E|\times 2|V|}$. We could slightly improve that and come up with a "two-hot encoding" $\mathbb{R}^{|E|\times|V|}$, but our representation complexity would remain $\mathcal{O}(|E||V|)$. Since we can expect nontrivial graphs with $|E| \geq \mathcal{O}(|V|)$, we're back to a representation complexity of $\mathcal{O}(N^2)$ or worse: we must represent categorical variables in a sublinear manner to make any headway.

We thus settle on representing $i \in V$ as an embedding $\mathbf{v}^i \in \mathbb{R}^\delta$ randomly sampled from the δ-dimensional Normal (*i.e.*, $\mathbf{v}^i \sim \mathcal{N}(\mathbf{0}_\delta, \mathbb{I}_\delta)$ for all $i \in V$), with the restriction that δ should be sublinear in N. Such embeddings are common in NLP and other domains where many-valued categorical variables are involved, but one typically trains these embeddings after their random initialization. In sharp contrast, we actually sample a fresh set of embeddings $\{\mathbf{v}^i\}$ for each graph to be represented: we argue that, while it may make sense to learn an embedding specialized for the word "cat", there is here no clear benefit to learn one specialized for "node 42". Still, the Normal distribution was solely selected for its analytical convenience: it may turn out that different distributions would be more appropriate. Other potentially promising candidates include sparse codes, projections on hyperspheres, and equiangular tight frames.

Now fully committed to this embedding choice, we pose the EXPLODE GDP in terms of the aforementioned table $V^{|E|\times 2}$ represented as $\mathbb{R}^{|E|\times 2\delta}$ then "unrolled" as $\mathbb{R}^{2\delta|E|}$. After sampling $\{\mathbf{v}^i\}$, the initial condition $\mathbf{U}_0$ is obtained by concatenating $\mathbf{v}^i$ and $\mathbf{v}^{i'}$ for each edge $\{i, i'\} \in E$. Appendix A poses a specific forward SDE and details how the analytical solution for the kernel and kernel scores are obtained. For our present needs, it should suffice to get some intuition by considering Fig. 1b.

At any time t, the point $\mathbf{u} \sim \mathbf{U}_t$ may be understood as specifying the position in $\mathbb{R}^\delta$ of $|E|$ pairs of particles, each pair being joined by a line. At time $t = 0$, these lines between particle pairs effectively "draw" the graph in $\mathbb{R}^\delta$, and there will be as many particles at position $\mathbf{v}^i$ as the number of edges in which node i is involved (*i.e.*, i's degree). However, the diffusion process quickly cause these particles to "explode" away from $\mathbf{v}^i$ and each other: there is no actual repulsive force involved, it is just that each particle experiences different random events. After a sufficient amount of time, the particles effectively forgot about their origin, resulting in a random jumble of unrelated lines.

Following the standard GDP recipe, we could *a priori* learn the score function $\mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u})$ and use the resulting backward SDE to run the process backward down to $t = 0$, interpreting sufficiently-close particles as representing the same node. However, "learning the score function" here turns out to be a tall order, which we attribute to a phenomenon that we call "simultaneity complexity". In a nutshell, the forward process blows up the graph information all at once, so the reverse process similarly has to infer the graph structure – local and global – simultaneously.

This conjecture begs for the question "Why isn't that an issue with image GDPs, or even with BASE?" We here blame the fact that a particle's position in EXPLODE does not bear any meaning in itself, but only through its relation with others particles. In an image GDP, not only does a pixel's color has some inherent meaning, but that pixel also always shares the same 4 neighbours in the image lattice through the process: the pixel can look at its neighbours to get an idea about itself. Similarly, high values in BASE's adjacency matrix are indicators of the probable presence of an edge, and a row/column with many high values is an indicator of an high degree node. In EXPLODE, there is a "fog of war" that prevents particles to get an appropriate awareness of the situation until it is too late.

Enter jumps: the DISSOLVE process. If EXPLODE's problem truly is that each of its nodes simultaneously blow up, then it makes sense to pose a different process where the particles originating from the same node "stick together" for longer, departing at a more gradual pace. There are many ways one could achieve this "clinginess" (*e.g.*, particle-particle attractive interactions, "potential wells" trapping particles near node embedding $\mathbf{v}^i$), but such "soft" approaches forbid (or highly complicate) obtaining an analytical solution for the kernel, which would cause inefficient training. We thus "chunk down the explosion into multiple all-or-nothing discrete splitting events": the reader is strongly encouraged to process this informal intuition through Fig. 1c before continuing further.

Formally, DISSOLVE is a jump diffusion process whose full internal state $(\mathbf{u}, g)$ is the join of a continuous state $\mathbf{u} \in \mathbb{R}^{n\delta}$, tracking the position of n particles in $\mathbb{R}^\delta$, and of a discrete substate $g \in \mathcal{G}$, where $\mathcal{G}$ allows to explicitly represent a graph in its native discrete format $g = (V, E)$. For $(V, E) \sim X$, we pose the initial condition $(\mathbf{U}_0, G_0)$ so that $G_0 = (V, E)$ and $\mathbf{U}_0 \sim \mathcal{N}(\mathbf{0}_{|V|\delta}, \mathbb{I}_{|V|\delta})$ (corresponding to EXPLODE's random embeddings $\{\mathbf{v}^i\}$). We pose the forward SDE driving the continuous evolution of $\mathbf{U}_t$ in the interval between jump events to be the same one as for EXPLODE. At a jump event, a new node is added to the discrete substate's graph and a pre-existing node with more than one edge is randomly selected: both nodes split the original node's edges among themselves, each ending up with a minimum of 1 edge. A copy of the selected node's position is appended at the end of the continuous substate, effectively adding δ dimensions to it. See Appendix B for details.

Beyond DISSOLVE. Although DISSOLVE does decently in terms of representation and simultaneity complexities, this does not in itself guarantee good time and space complexity for GDP models based off it. In fact, if the score function were to consider all particle-particle covariances, we would quickly end back in $\mathcal{O}(N^2)$ (or more) territory. However, we see two complementary paths toward a more efficient successor to DISSOLVE: approximations involving locality-sensitive hashing (leveraging the fact that the score function for a particle should mostly depend on the neighbouring particles), and the addition of new birth–death mechanisms for the particles in the process (greatly reducing the number of particles present at any time in the process). This last point reflects the fact that the analytical tractability of DISSOLVE's kernel is not just a mere accident: Appendix C paves the way for a new breed of jump diffusion GDPs that need not be restricted to graphs.

References

[1] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021.

[2] Reinhard Diestel. Graph theory 3rd ed. *Graduate texts in mathematics*, 173:33, 2005.

[3] Tim Dockhorn, Arash Vahdat, and Karsten Kreis. Score-based generative modeling with critically-damped langevin diffusion. In *International Conference on Learning Representations*, 2021.

[4] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[5] Dwaraknath Gnaneshwar, Bharath Ramsundar, Dhairya Gandhi, Rachel Kurchin, and Venkatasubramanian Viswanathan. Score-based generative models for molecule generation. *arXiv preprint arXiv:2203.04698*, 2022.

[6] Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*, 34:12454–12465, 2021.

[7] Emiel Hoogeboom, Víctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *International Conference on Machine Learning*, pages 8867–8887. PMLR, 2022.

[8] Bowen Jing, Gabriele Corso, Renato Berlinghieri, and Tommi Jaakkola. Subspace diffusion generative models. *arXiv preprint arXiv:2205.01490*, 2022.

[9] Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. *arXiv preprint arXiv:2202.02514*, 2022.

[10] Shitong Luo, Chence Shi, Minkai Xu, and Jian Tang. Predicting molecular conformation via dynamic graph score matching. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 19784–19795. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper/2021/file/a45a1d12ee0fb7f1f872ab91da18f899-Paper.pdf`.

[11] Gautam Mittal, Jesse Engel, Curtis Hawthorne, and Ian Simon. Symbolic music generation with diffusion models. *arXiv preprint arXiv:2103.16091*, 2021.

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

[13] Yang Song. Generative modeling by estimating gradients of the data distribution, May 2021. URL `https://yang-song.net/blog/2021/score/`.

[14] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2020.

[15] Yang Song, Conor Durkan, Iain Murray, and Stefano Ermon. Maximum likelihood training of score-based diffusion models. *Advances in Neural Information Processing Systems*, 34:1415–1428, 2021.

[16] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[17] Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. Geodiff: A geometric diffusion model for molecular conformation generation. In *International Conference on Learning Representations*, 2021.

A Details for EXPLODE

This section specifies SDEs for EXPLODE and discuss of its forward analytical solution, providing both the associated kernel and kernel score.

Although basically any diffusion process could do, we here favour the so-called "variance-preserving" SDE [14] with unit noise, *i.e.*,

$$\mathrm{d}\mathbf{u}_t = -\tfrac{1}{2}\mathbf{u}_t \mathrm{d}t + \mathrm{d}\mathbf{w}_t \qquad \text{(forward)} \tag{2a}$$

$$\mathrm{d}\mathbf{u}_t = -\left(\tfrac{1}{2}\mathbf{u}_t + \mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u}_t)\right)\mathrm{d}t + \mathrm{d}\bar{\mathbf{w}}_t \quad , \qquad \text{(backward)} \tag{2b}$$

where $\mathbf{w}_t$ (resp. $\bar{\mathbf{w}}_t$) is the standard Wiener process for time flowing forward (resp. backward). For Gaussian distributions, this process can be solved analytically in the forward direction:

$$\text{if } \mathbf{U}_t \sim \mathcal{N}(\boldsymbol{\mu}_t, \Sigma_t), \text{ then } \mathbf{U}_{t'} \sim \mathcal{N}\left(\boldsymbol{\mu}_t \mathrm{e}^{-\frac{1}{2}(t'-t)}, \mathbb{I} + (\Sigma_t - \mathbb{I})\mathrm{e}^{-(t'-t)}\right) \text{ for } t' > t. \tag{3}$$

We now formalize how the embeddings $\{\mathbf{v}^i\}$ are assigned to the initial condition $\mathbf{U}_0$ for a training sample $x = (V, E)$. We must first decide on a particle-to-node assignment $\boldsymbol{\nu} = (\nu_0, \nu_1, \cdots, \nu_{(2|E|-1)})$ such that there exists a particle pairing E' satisfying $E = \{\{\nu_k, \nu_{k'}\} : \{k, k'\} \in E'\}$. For the small matrices depicted on the top of Fig. 1b, this assignment is $\boldsymbol{\nu} = (0, 1, 2, 3, 4, 1, 3, 1)$ (which may be related with the colors in these small matrix margins) and we can verify that $E' = \{\{0, 1\}, \{2, 5\}, \{3, 4\}, \{6, 7\}\}$ satisfies the above condition (which may be related with the lines drawn between particle pairs).

Given $\boldsymbol{\nu}$ and $\{\mathbf{v}^i\}$, the initial condition is $\mathbf{U}_0 = \mathbf{v}_{\nu_0} \frown \mathbf{v}_{\nu_1} \frown \cdots \frown \mathbf{v}_{\nu_{(2|E|-1)}}$, where "$\frown$" is the vector concatenation operator. Recalling that $\mathbf{v}^i \sim \mathcal{N}(\mathbf{0}_\delta, \mathbb{I}_\delta)$, we have $\mathbf{U}_0 \sim \mathcal{N}(\mathbf{0}_{2\delta|E|}, \Sigma_{0|x})$ with

$$\Sigma_{0|x} = \begin{bmatrix} \mathbb{I}_\delta & 1_{\nu_0=\nu_1}\mathbb{I}_\delta & 1_{\nu_0=\nu_2}\mathbb{I}_\delta & \cdots \\ 1_{\nu_1=\nu_0}\mathbb{I}_\delta & \mathbb{I}_\delta & 1_{\nu_1=\nu_2}\mathbb{I}_\delta & \cdots \\ 1_{\nu_2=\nu_0}\mathbb{I}_\delta & 1_{\nu_2=\nu_1}\mathbb{I}_\delta & \mathbb{I}_\delta & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \tag{4}$$

where $1_{\nu_k=\nu_{k'}}$ is the indicator function with value 1 if $\nu_k = \nu_{k'}$ and 0 otherwise. Plugging this initial condition into Eq. (3) allows us to efficiently sample the system's state at any time $t \geq 0$ using

$$\mathbf{U}_{t|X=x} \sim \mathcal{N}(\mathbf{0}_{2\delta|E|}, \Sigma_{t|x}) \text{ where } \Sigma_{t|x} = \begin{bmatrix} \mathbb{I}_\delta & 1_{\nu_0=\nu_1}\mathrm{e}^{-t}\mathbb{I}_\delta & 1_{\nu_0=\nu_2}\mathrm{e}^{-t}\mathbb{I}_\delta & \cdots \\ 1_{\nu_1=\nu_0}\mathrm{e}^{-t}\mathbb{I}_\delta & \mathbb{I}_\delta & 1_{\nu_1=\nu_2}\mathrm{e}^{-t}\mathbb{I}_\delta & \cdots \\ 1_{\nu_2=\nu_0}\mathrm{e}^{-t}\mathbb{I}_\delta & 1_{\nu_2=\nu_1}\mathrm{e}^{-t}\mathbb{I}_\delta & \mathbb{I}_\delta & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{5}$$

Stated otherwise, the kernel is the Gaussian $p(\mathbf{U}_t = \mathbf{u}|X = x) = \phi(\mathbf{u}; \mathbf{0}_{2\delta|E|}, \Sigma_{t|x})$ and the kernel score is $\boldsymbol{\kappa}_t(\mathbf{u}|x) = -\Sigma_{t|x}^{-1}\mathbf{u}$.

Note that all the above may be performed at pre-processing time: the training loop minimizing Eq. (1) may receive batches composed of samples $(t, \mathbf{u}, \boldsymbol{\kappa}_t(\mathbf{u}|x))$. Worker processes (with or without GPU acceleration) may prepare such batches in parallel from the main optimization process.

B Details for DISSOLVE

This section establishes some notations, provides the kernel and kernel score for the DISSOLVE process, discusses some adaptations (necessary and optional) for Eq. (1), hint at the functional form for the graph-dependent score function $\mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u}|g)$, and drafts the path toward improving DISSOLVE with an additional birth–death mechanisms for the particles.

Notations and solution. We introduce two hyperparameters, τ_{pairs} and τ_{stop}, so that $0 < \tau_{\text{pairs}} \leq \tau_{\text{stop}}$. For a graph $x = (V, E)$ without any singleton (*i.e.*, all its nodes have at least one edge), a total of $J = 2|E| - |V|$ jump events will occur before τ_{pairs}, the time at which it is guaranteed that all that remains in the discrete substate is a soup of disjoint particle pairs. As for τ_{stop}, it is the time at which

we end the process and assume that all information about the input x, except its number of edges $|E|$, has been destroyed from the full state: the time interval $\tau_{\text{stop}} - \tau_{\text{pairs}}$ allows the continuous part of state to thermalize. When generating graphs, we may sample $|E|$ by some external means (be it the training distribution or some interpolation of it) then initialize the backward process at τ_{stop} using $\mathbf{U}_{\tau_{\text{stop}}} = \mathcal{N}(\mathbf{0}_{2|E|}, \mathbb{I}_{2|E|})$ and setting the graph to a soup of $|E|$ pairs. Note that we present below variations that also destroy this information $|E|$.

We need to pose the distribution for the different times $0 \leq \tau_1 \leq \tau_2 \leq \cdots \leq \tau_J \leq \tau_{\text{pairs}}$ at which the J jumps occur. At the moment of writing these lines, our code uniformly samples J times in the interval $[0, \tau_{\text{pairs}})$, sorts them, then assigns the j-th one to τ_j. However, future investigations may reveal better choices. In any case, we use the convention $\tau_0 = 0$ and $\tau_{J+1} = \tau_{\text{stop}}$ so that if $\tau_j \leq t < \tau_{j+1}$, then j jumps occurred before time t.

We also need to pose the specific way in which the j-th jump transforms the graph $g_{j-1} = (V, E)$ (before the jump) into the graph $g_j = (V', E')$ (after the jump). Our current code first samples a node i from g_{j-1}, giving each node a weight of one less than its degree (minimum 0). Hence, a node of degree 3 is twice as likely as a node of degree 2 to be picked, and a node of degree 0 or 1 will never be picked. We then set $V' = V \cup i'$, where $i' = |V|$ is the new node. We note $E_i = \{\{i, i''\} : i'' \in V, \{i, i''\} \in E\}$ the set of edges in E that involve i. Now we define two disjoint sets by randomly selecting two edges from E_i and assigning one to each of those two sets, then we randomly distribute the remaining edges among the two sets according to a fair coin. We note one of those two sets E'_i, and we substitute all instances of i in the other set by i' before placing them in $E'_{i'}$. We finally set $E' = (E \setminus E_i) \cup E'_i \cup E'_{i'}$, which completes the specification of g_j. Again, future investigations may reveal better strategies.

We abide by the "càdlàg" convention (from the French "*continue à droite, limite à gauche*", meaning "right continuous with left limits") such that τ_j can be understood as the time *immediately after* the j-th jump (so $\mathbf{U}_{\tau_j}$ has already been subjected by the j-th jump), and we understand τ_j^- as the time *immediately before* the j-th jump, writing $\mathbf{U}_{\tau_j^-}$ the appropriate limit. If i is the particle being split by the j-th jump, then that jump simply concatenate that particle's position at the end of the continuous substate, taking $\mathbf{U}_{\tau_j^-}$ to $\mathbf{U}_{\tau_j} = \mathbf{U}_{\tau_j^-} \frown \left[\mathbf{U}_{\tau_j^-}\right]_{i\delta:(i+1)\delta}$.

This last relation, together with Eq. (3) and the initial condition $\mathbf{U}_0 \sim \mathcal{N}(\mathbf{0}_{|V|\delta}, \mathbb{I}_{|V|\delta})$, fully specify how to obtain a sample $\mathbf{U}_t$ at time $\tau_j \leq t < \tau_{j+1}$ for any initial graph $x = (V, E)$, provided we know when jumps occur and which node gets split in each of them. We formalize this type of information as the *discrete history* H, whose samples $h \sim H$ have the format $h = \big((\tau_1, g_1), (\tau_2, g_2), \cdots, (\tau_J, g_J)\big)$. We write $h_{:j} = \big((\tau_1, g_1), \cdots, (\tau_{j-1}, g_{j-1})\big)$ the part of the discrete history that, together with the discrete initial condition $g_0 = x$, suffices to sample $\mathbf{U}_t$ if $t < \tau_j$. In particular, notice that we may easily infer which node got split at the j'-th jump (with $j' < j$) by finding which of $g_{j'-1}$'s node lost at least one edge in $g_{j'}$.

In fact, for $\tau_{j-1} \leq t < \tau_j$, we can iteratively obtain the covariance matrix $\Sigma_{t|x,h} = \Sigma_{t|x,h_{:j}}$ (*i.e.*, with no dependency whatsoever in the j-th jump nor the following ones) for a given sample $x = (V, E)$ and discrete history h such that $\mathbf{U}_t$ may be sampled from the Gaussian $\mathcal{N}(\mathbf{0}_{(|V|+j)\delta}, \Sigma_{t|x,h})$. Concretely, the effect of the j-th jump splitting g_{j-1}'s node i can be inferred by considering an *Ansatz*:

$$
\begin{aligned}
\text{if } \Sigma_{\tau_j^-|x,h_{:j}} &= \begin{bmatrix} \mathbb{I}_\delta & c_{0,1}\mathbb{I}_\delta & \cdots & c_{0,|V|+j-1}\mathbb{I}_\delta \\ c_{1,0}\mathbb{I}_\delta & \mathbb{I}_\delta & \cdots & c_{1,|V|+j-1}\mathbb{I}_\delta \\ \vdots & \vdots & \ddots & \vdots \\ c_{|V|+j-1,0}\mathbb{I}_\delta & c_{|V|+j-1,1}\mathbb{I}_\delta & \cdots & \mathbb{I}_\delta \end{bmatrix}, \\
\text{then } \Sigma_{\tau_j|x,h_{:(j+1)}} &= \begin{bmatrix} \mathbb{I}_\delta & c_{0,1}\mathbb{I}_\delta & \cdots & c_{0,|V|+j-1}\mathbb{I}_\delta & c_{0,i}\mathbb{I}_\delta \\ c_{1,0}\mathbb{I}_\delta & \mathbb{I}_\delta & \cdots & c_{1,|V|+j-1}\mathbb{I}_\delta & c_{1,i}\mathbb{I}_\delta \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{|V|+j-1,0}\mathbb{I}_\delta & c_{|V|+j-1,1}\mathbb{I}_\delta & \cdots & \mathbb{I}_\delta & c_{|V|+j-1,i}\mathbb{I}_\delta \\ c_{i,0}\mathbb{I}_\delta & c_{i,1}\mathbb{I}_\delta & \cdots & c_{i,|V|+j-1}\mathbb{I}_\delta & \mathbb{I}_\delta \end{bmatrix}, \qquad (6)
\end{aligned}
$$

where $c_{i',i''} = c_{i'',i'}$ and $c_{i',i'} = 1$. We may verify that this *Ansatz* holds by recursion, starting with $\Sigma_{\tau_1^-|x,h_{:1}} = \mathbb{I}_{|V|\delta}$ (because the initial condition remains unaffected by Eq. (3)). In general, chaining

Eq. (3) and Eq. (6) provides the sought covariance matrix at any time. The reader may get some additional intuition by inspecting the small matrices on the top of Fig. 1c.

In particular, this provides the *history-conditioned kernel* $p(\mathbf{U}_t = \mathbf{u}|X = x, H = h) = \phi(\mathbf{0}, \Sigma_{t|x,h})$. The initial conditions and system's symmetry cause the means to always by zero. All of $\mathbf{U}_t$'s information about X is captured by its covariance matrix.

Adapting the loss. In standard GDP, the process' state at any time t is given by a continuous vector $\mathbf{U}_t$, and we learn the score function $\mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u}) \approx \nabla_{\mathbf{u}} \ln p(\mathbf{U}_t = \mathbf{u})$ by minimizing the loss Eq. (1), which involves the kernel score $\boldsymbol{\kappa}_t(\mathbf{u}|x) = \nabla_{\mathbf{u}} \ln p(\mathbf{U}_t = \mathbf{u}|X = x)$. Here the process' state at a time $\tau_j \leq t < \tau_{j+1}$ is the joint $(\mathbf{U}_t, G_j)$, involving an extra discrete substate G_j in addition to the usual continuous vector $\mathbf{U}_t$. We now consider how to generalize the score function to this case, as well as obtaining the denoising score matching loss generalizing Eq. (1).

It is important to keep in mind the circumstances in which we will need the score function: we wish to run the diffusion process backward from some time τ_{j+1} to time τ_j. More specifically, this operation takes as input the time τ_{j+1} (but **not** the value of j: we do not know how many jumps occur between time 0 and time τ_{j+1}) as well as the state $(\mathbf{u}, g)$ at that time (*i.e.*, $\mathbf{U}_{\tau_{j+1}^-} = \mathbf{u}$ and $G_j = g$). We do not know τ_j *a priori*, but we can infer it from the continuous substate: going backward from τ_{j+1}^-, we can detect τ_j as the first time $t < \tau_{j+1}$ such that there is a collision between two particles in $\mathbf{U}_t$ (*i.e.*, in practice, two particles get within some small distance ϵ of one another). Notice that the graph g stays the same over this whole flight: we can view g as a parameter of the diffusion process between the interval $\tau_j \leq t < \tau_{j+1}$. This motivates us to consider the *graph-conditioned score function* $\mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u}|g) = \nabla_{\mathbf{u}} \ln p(\mathbf{U}_t = \mathbf{u}|G_j = g)$ which, given the g parameterizing this interval, allows us to approximate the reverse stochastic process described above.

There are minor complications due to how this graph g appears as an entry of the discrete history H, but which one depends on T, the time at which we are presently minimizing the error (see the comment following Eq. (1)). To simplify the following, we define the stochastic variable $\widetilde{G}$ such that, given $H = \big(\cdots, (\tau_j, g_j), \cdots\big)$ and $T = t$, if $\tau_j \leq t\tau_{j+1}$, then $\widetilde{G} = g_j$. Despite the fact that we defined $\widetilde{G}$ in terms of H, we will consider the opposite dependency: we will sample discrete histories H that are compatible with given $\widetilde{G}$, X and T. Using the notation "$\llbracket t \rrbracket$" introduced later in Appendix C, we have $\widetilde{G} = G_{\llbracket t \rrbracket}$.

With these definitions in mind, we observe that

$$p\big(\mathbf{U}_t = \mathbf{u} \;\big|\; X = x, \widetilde{G} = g\big) = \mathbb{E}_{h \sim H|X=x,\widetilde{G}=g,T=t}\, p\big(\mathbf{U}_t = \mathbf{u} \;\big|\; X = x, H = h\big)\,, \tag{7}$$

where the probability on the right-hand side is known to be $\phi(\mathbf{0}, \Sigma_{t|x,h})$. This provides the *graph-conditioned kernel*

$$\begin{aligned}\boldsymbol{\kappa}_t(\mathbf{u}|x, g) &= \nabla_{\mathbf{u}} \ln p\big(\mathbf{U}_t = \mathbf{u} \;\big|\; X = x, \widetilde{G} = g\big) \\ &= \left(\frac{\mathbb{E}_{h \sim H|X=x,\widetilde{G}=g,T=t}\, \phi(\mathbf{u}; \mathbf{0}, \Sigma_{t|x,h})(-\Sigma_{t|x,h}^{-1})}{\mathbb{E}_{h \sim H|X=x,\widetilde{G}=g,T=t}\, \phi(\mathbf{u}; \mathbf{0}, \Sigma_{t|x,h})}\right)\mathbf{u}\,,\end{aligned} \tag{8}$$

which we may estimate by empirically sampling a certain number of discrete histories compatible with $X = x$, $\widetilde{G} = g$ and $T = t$. Again, note that Eq. (8) may be evaluated outside of the main training loop by some "dataloader" process. The corresponding denoising score matching loss thus becomes

$$\mathcal{L}^{\boldsymbol{\theta}} = \mathbb{E}_{x \sim X}\, \mathbb{E}_{t \sim T}\, \lambda_t\, \mathbb{E}_{g \sim \widetilde{G}|X=x,T=t}\, \mathbb{E}_{\mathbf{u} \sim \mathbf{U}_t|X=x,\widetilde{G}=g} \Big\| \mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u}|g) - \boldsymbol{\kappa}_t(\mathbf{u}|x, g) \Big\|_2^2\,. \tag{9}$$

Functional form for $\mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u}|g)$. Our current code represents $\mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u}|g)$ as a message passing neural network (MPNN) [4], which is a family of graph neural networks. Our MPNN proceeds in two steps: first obtain "features" for each particles, then use these features to estimate a matrix $S_t^{\theta}(\mathbf{u}|g)$ such that $\mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u}|g) = S_t^{\theta}(\mathbf{u}|g)\mathbf{u}$. Again, future investigations may reveal better strategies.

In the first step, we learn an MPNN whose main goal is to grant to each particle some features capturing the "role" played by the corresponding node in the graph g. We initialize each node's representation with the corresponding particle's distance from the origin as well as that node's degree.

Multiple rounds of message passing then propagate representations on the graph g: we append to each message the distance between the corresponding source and target particles, and aggregation/update may only depend on these messages as well as the receiving node's degree and distance from origin. The main learned representations thus depend on g but also on $\mathbf{u}$ through pairwise distances between nodes and/or origin: it is not clear yet to which extent, but we expect these limited dependencies on $\mathbf{u}$ to improve the MPNN's representations over the usual Weisfeiler–Lehman graph isomorphism bounds [16].

The second step is a another MPNN, initialized on the first one's output and with similar dependencies on distances in $\mathbf{u}$, except that this time messages are exchanged on a different graph. *A priori*, we would have to consider the complete graph where all nodes are neighbours to all other nodes, but there are strong indications that sparse interactions should suffice to get good approximations. Indeed, we argue that most of the contributions should come from the complement of g, and that this graph may be further filtered by only considering particles that are sufficiently close from each other.

The clear part of this picture is that the sought symmetric matrix $S_t^\theta(\mathbf{u}|g)$ has the same general structure of "weighted $\mathbb{I}_\delta$ blocks" as those seen Eq. (6), except that the main diagonal's weights need not be 1. The fact that such a matrix gives $\mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u}|g) = S_t^\theta(\mathbf{u}|g)\mathbf{u}$ follows from the observation that

$$p(\mathbf{U}_t|\widetilde{G} = g) = \mathbb{E}_{x\sim X}\, p\big(\mathbf{U}_t = \mathbf{u} \mid X = x, \widetilde{G} = g\big) \tag{10}$$

implies that $\mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u}|g)$ has a similar form as $\boldsymbol{\kappa}_t(\mathbf{u}|x, g)$ in Eq. (8).

Intuitively, each weight in that matrix captures to which extent pairs of particles are " attracted to each other" in the backward process. Due to the nature of the process, these weights should be sparsely populated: our second MPNN models these weights as a function of the messages exchanged on a graph that is similarly sparse. More experiments are needed on this front.

Injecting and destroying particles in DISSOLVE. In DISSOLVE, once a pair of two degree one particles has detached from the rest, we know that neither of these particles will ever be involved in a splitting event again in the forward process. Once the entries of the covariance matrix involving those two particles are sufficiently close to zero, there is no point in tracking those two particles anymore: we could amend the forward process such that the pair spontaneously disintegrate. If we can learn a function of t, $\mathbf{u}$ and g that predicts the global rate at which these disintegration take place, we can use that rate to spontaneously create pairs of particles in the backward process.

Note that this operation does destroy the information about the number of edges $|E|$ in the graph, which may be seen as a feature instead of a bug (because it removes the need to sample $|E|$ before starting the backward process at generation time). Moreover, if preserving the exact number of edges is desirable, then the number of destroyed edges may be tracked by incrementing a counter in an augmented discrete state $\mathcal{G}' = \mathcal{G} \times \mathbb{N}_0$.

Similarly, at the beginning of the forward process, we know that particles that have not yet been involved in a splitting event cannot be involved in any non-diagonal nonzero terms of the covariance matrix: we don't need to track these particles until they get involved in splitting events. We may thus further amend the forward process such that it injects particles on their first split, learn a function of t, $\mathbf{u}$ and g that predicts the probability that a split event was concomitant with an injection, and use that probability to assess, in the backward process, when to eject merging particles.

Further improvements could be made by crafting an "injection schedule" that determines which particles to inject first, together with a "splitting schedule" determining which particles to split, so that as few particles as possible are simultaneously "active" in the process. For graphs of low treewidth [2], a promising direction is to consider — in addition to 1. the particles not injected yet, 2. the particles currently active in the process and 3. the particles that were disintegrated — a fourth tier of particles that have been injected but are presently "shelved", to be made active again under some circumstances later. The tree decomposition of the graph could then inform good choices for "injection schedule", "splitting schedule" and new "shelving/unshelving schedules".

Of course, this long term programme repose on numerous assumptions that should first be verified in simpler contexts. However, we wish to highlight that this rich landscape hints at the existence of very efficient graph GDPs

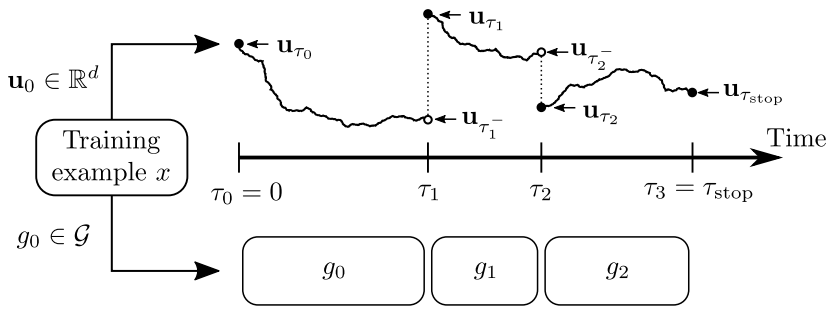
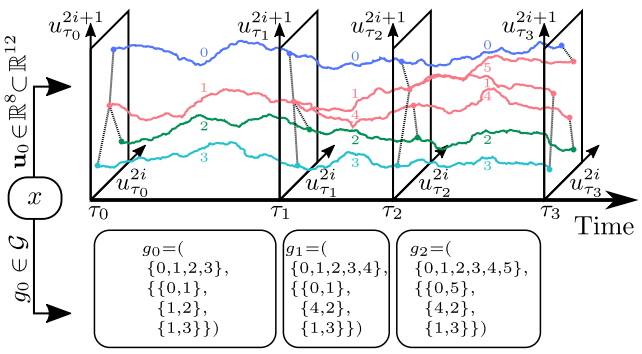
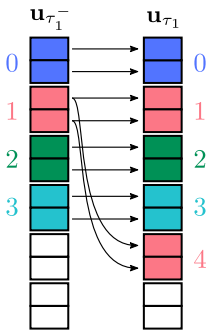


Figure 2: **Generic jump-diffusion process in $\mathbb{R}^d \times \mathcal{G}$.** This figure solely illustrates notations and core ideas for our GDP framework: *it is not meant to represent any specific process*. *Top:* A piecewise-continuous trajectory in the $\mathbb{R}^d$ subspace is drawn as one-dimensional for simplicity. *Bottom:* Discrete trajectories in the $\mathcal{G}$ subspace are constant between jumps.



(a) Specialization of Fig. 2 to DISSOLVE.



(b) Jump at τ_1 in Fig. 3a.

Figure 3: **A different take on Fig. 1c.** Here a smaller graph is considered, and curves are hand-drawn (Fig. 1 shows actual stochastic processes), which may slightly improve readability. (a) *Bottom:* An input graph x (here the "claw graph") is directly represented as $g_0 \in \mathcal{G}$. At the jump time τ_1, a new node 4 is created and one of 1's edge is given to it. A similar jump occurs at τ_2, resulting in the graph g_2 composed of 3 disjoint edges. *Top:* Each curve tracks the 2-dimensional position of a node (*i.e.*, $\delta = 2$): like Fig. 2, Fig. 3a shows a single trajectory. Since x has 4 nodes, there are initially only 4 curves, and $\mathbf{u}_0 \sim \mathcal{N}(\mathbf{0}_8, \mathbb{I}_8)$ lives in a $\mathbb{R}^8$ subspace of $\mathbb{R}^{12}$. Each jump adds a node, and thus 2 dimension to the continuous substate. (b) At the jump time τ_1, the new node 4 takes the same position as the node 1 (from which it got its edge).

C General framework for jump diffusion GDP

This section revisits core ideas discussed in the main text in the context of sparse graphs, here reframed in a much broader context. Our aim here is to be "self-contained but terse". Notations are compatible.

We note X the stochastic variable associated with some data distribution: we have access to a training dataset providing a finite number of samples $x \sim X$, and our goal is to generate new samples $\hat{x} \in X$. *A priori*, these samples can be anything.

We address this problem with a GDP. At a given time t, the process' *full state* corresponds to a point in the space $\mathbb{R}^d \times \mathcal{G}$, which we understand as the joint of a *continuous state* in the subspace $\mathbb{R}^d$ and a *discrete state* in the subspace $\mathcal{G}$. *A priori*, the elements of $\mathcal{G}$ may be anything.

The stochastic process $\mathbf{U}_t$, with samples $\mathbf{u} \sim \mathbf{U}_t$, tracks the evolution of the continuous state, which continuously obeys a specified SDE for $0 \leq t \leq \tau_{\text{stop}}$ except at a finite number of *jump times* $\tau_1, \cdots, \tau_J$. To simplify the notation, we pose $\tau_0 = 0$ and $\tau_{J+1} = \tau_{\text{stop}}$ so that $\tau_j \leq t < \tau_{j+1}$ means that j jumps have taken place before time t. The stochastic variable G_j, with samples $g_j \sim G_j$, tracks the discrete state after j jumps, which remains static in-between jumps. The

number of jumps J, the jump times $\tau_1, \cdots, \tau_J$, and the *stopping time* τ_{stop} may all differ between realizations of the jump-diffusion process: we write $\boldsymbol{\mathcal{T}}$ the stochastic variable such that $\boldsymbol{\tau} \sim \boldsymbol{\mathcal{T}}$ provides $\boldsymbol{\tau} = (0, \tau_1, \cdots, \tau_J, \tau_{\text{stop}})$. We write τ_j^- the time immediately before jump j, and we write $\llbracket t \rrbracket$ the number of jumps before time t (so $(\mathbf{u}_t, g_{\llbracket t \rrbracket})$ is the full state at time t). Figure 1 exemplifies some of these notations.

The initial condition $(\mathbf{U}_0, G_0)$ is obtained from a training example $x \sim X$ such that, given $(\mathbf{u}_0, g_0)$, we may perfectly recover x, and we may do so efficiently. A necessary condition is $\mathrm{I}(X; \mathbf{U}_0, G_0) = \mathrm{H}(X)$. Like in a standard GDP, the jump-diffusion process is chosen so that it gradually destroys the mutual information between the process' state and the training sample: we require that τ_{stop} is sufficiently high so that $\mathrm{I}(X; \mathbf{U}_{\tau_{\text{stop}}}, G_J)$ is negligible. We wish for the jumps to be simple to "undo" efficiently: a necessary condition is that the mutual information between the training sample and the full state is the same immediately after a jump than it was immediately before it, *i.e.*,

$$\mathrm{I}\big(X; \mathbf{U}_{\tau_j}, G_j\big) = \mathrm{I}\big(X; \mathbf{U}_{\tau_j^-}, G_{j-1}\big) \quad \forall\, 1 \leq j \leq J\ . \tag{11}$$

Our earlier requirement for the initial condition can similarly be viewed as "a jump from x to $(\mathbf{u}_0, g_0)$."

Intuitively, G_j may carry less information about x than G_{j-1} did, but the difference in informational content has been "pushed into $\mathbf{U}_{\tau_j}$" so that, overall, the jump preserved all the information about X. Conversely, any information about X missing from the full state has thus — like in standard GDP — been destroyed by the (continuous) diffusion process.

Therefore, we can directly generalize the standard GDP approach to generate new samples $\hat{x} \sim X$ provided that: 1. we can define a forward jump-diffusion process satisfying the above requirements; 2. we can learn the associated score estimation function $\mathbf{s}_t^{\boldsymbol{\theta}}(\mathbf{u}|g) \approx \nabla_{\mathbf{u}} \ln \mathbb{P}\left(\mathbf{U}_t = \mathbf{u} | G_{\llbracket t \rrbracket} = g\right)$; 3. we can efficiently sample τ_{stop} and $(\mathbf{U}_{\tau_{\text{stop}}}, G_J)$; 4. in the backward process, we can "detect" when a jump should occur and we know how to "undo" that jump.

D Relation with other work

The present work focuses on general purpose, efficient GDPs for sparse graphs. Besides Niu et al. [12] and Jo et al. [9], other work considered GDPs in a graph context, mainly in the molecular domain. However, these works often focus on generating conformations for fixed graphs [10, 17]. Other work [7] predict atom positions, then infer the graph structure from those positions, which is an approach that does not generalize to many domains of interest. Moreover, the use of application-specific "codes", *e.g.*, SELFIES for molecular data [5], may mitigate representational complexity issues.

More generally, the problem of "generating graphs" has been addressed an incredible number of times across fields such as computer science (with or without machine learning involved), mathematics, physics, sociology, engineering, *etc.* We here acknowledge this fact, but focus our coverage of the literature those most relevant to this work. The present work focuses general purpose, efficient generation for sparse graphs using GDPs.

Although they do not involve graphs, two more works deserve mention as they relate to the general framework presented in Appendix C. The first one, Jing et al. [8] considers an image GDP in which the stochastic process proceeds "the standard way" in pixel-space up to a certain point, at which it halves the resolution of the representation: in our language, this resolution halving corresponds to a jump. More generally, we could consider the system's state as a lattice specifying the spatial arrangement of the pixels (discrete information in $\mathcal{G}$), together with a vector $\mathbf{U}_t$ encoding the RGB representations of all pixels (without information about their location). The jump transforms the lattice into a new one, adapting $\mathbf{U}_t$ accordingly.

The second work, Dockhorn et al. [3], is relevant for two reasons. At a conceptual level, it decouples the sample $x \sim X$ from the initial condition $\mathbf{U}_0 = \mathbf{u}$: we need not have a bijective map between the input sample and the system's initial condition. Indeed, in absence of a discrete substate, it suffices to have $p(\mathbf{U}_0 = \mathbf{u} | X = x)$ such that $\mathrm{I}(X; \mathbf{U}_0) = \mathrm{H}(X)$ (as long as we know how to efficiently map back $\mathbf{U}_0$ to x). At a more practical level, our own methods could benefit from a critically-damped Langevin diffusion treatment, and the addition of "conjugate variables" has interesting applications to improve DISSOLVE (*e.g.*, grant momentum in opposite directions to splitting particles, give a different mass according to a particle's corresponding node degree).