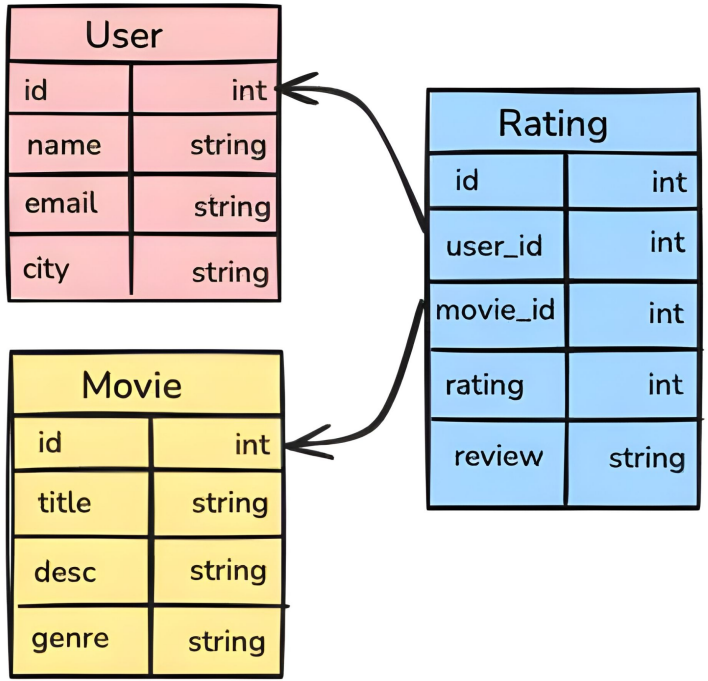
Graph-Conditional Flow Matching for Relational Data Generation

Davide Scassola, PhD candidate at University of Trieste
Sebastiano Saccani, Aindo
Luca Bortolussi, University of Trieste

The Problem

- Generating synthetic relational data
  - Multiple tables
  - Foreign-key relationships

- Why?
  - Privacy
  - Data augmentation



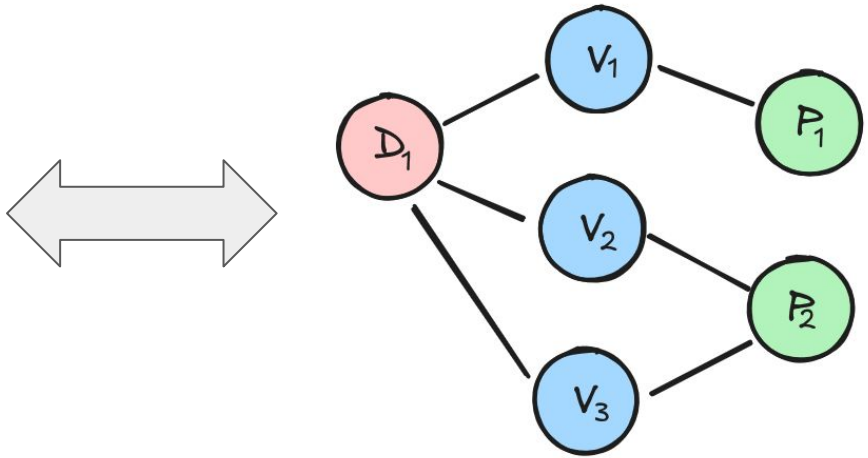
Relational datasets are (heterogeneous) graphs

- Rows ➜ Nodes
- Foreign-keys ➜ Edges

Visits			
ID	Dr	Pt	Features
1	1	1	…
1	1	2	…
1	1	2	…

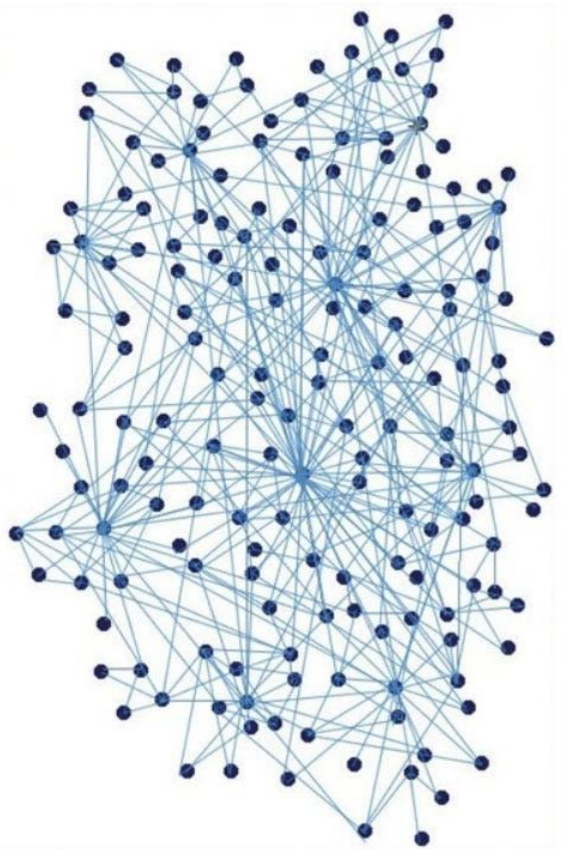
Doctors	
ID	Features
1	…

Patients	
ID	Features
1	…
2	…



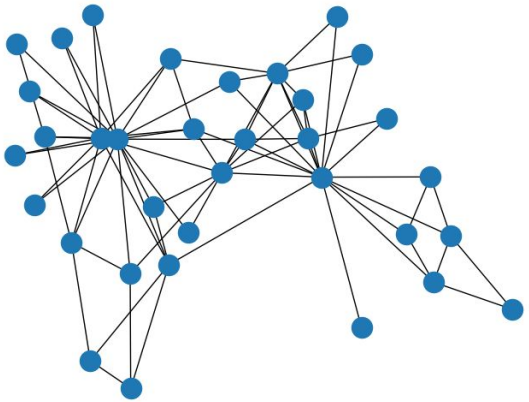
Challenges of Graph Generation

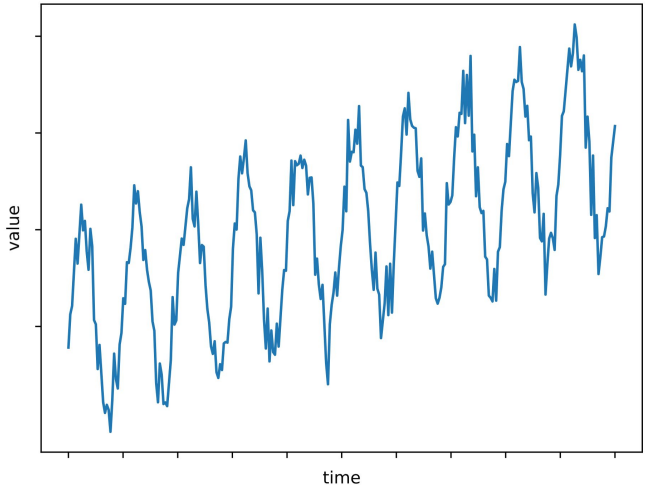
- Modelling correlations in connected nodes
- Generating the graph structure
- Large connected components (a connected component is a single sample!)
    - Large and variable dimensionality
    - Adjacency matrix generation (scales quadratically in the number of nodes)



iid samples?

- Selecting samples and train-test split are not trivial
- One can learn “within-sample” patterns





Current Methods

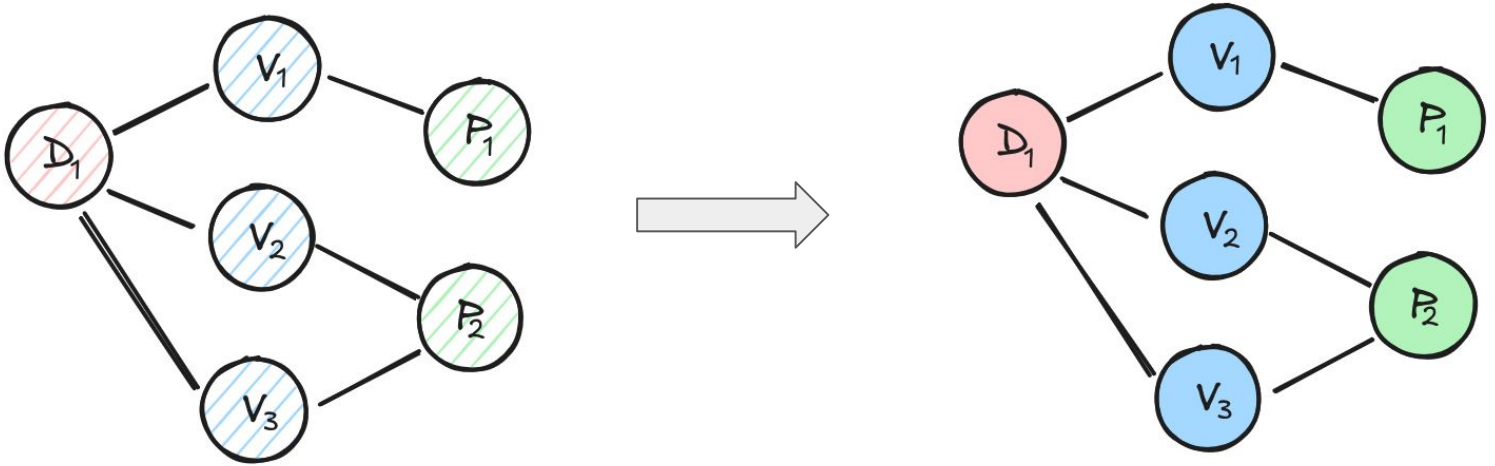
- Deep generative models for graphs:
  - Generation of a dense adjacency matrix ➜ they don’t scale
- Relational data generation methods:
  - Limited expressiveness (e.g., independence assumptions)
  - Limited compatibility

Guo, Xiaojie, and Liang Zhao. "A systematic survey on deep generative models for graph generation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.5 (2022): 5370-5390.

What we aim for

- **Expressiveness**
  - Any node depends on any other in the same connected component
- **Flexibility**
  - Support for any kind of graph
- **Scalability**
  - Scaling to large connected components (thousands of nodes)

Our Approach: Graph-conditional Generation



fk graph ~ p(fk graph)

(scalable model, or assume it is given)

features ~ **p(features | fk graph)**

(deep generative model)

Xu, Kai, et al. "Synthetic data generation of many-to-many datasets via random graph generation." The Eleventh International Conference on Learning Representations. 2022.

What Model?

- $p(\text{features} \mid \text{fk graph})$ should be expressive, flexible, scalable.
- We want to model whole connected components
- We use **flow matching**
- Why a diffusion-like model?
    - It easily scales to large dimensionalities

“Variational” Flow Matching

Continuous Normalizing Flows (ODEs) + diffusion-like training

- **Sampling**: solving an ODE

$$\frac{d}{dt}\varphi_t(x) = v_t(\varphi_t(x)) \text{ with initial conditions } \varphi_0(x) = x$$

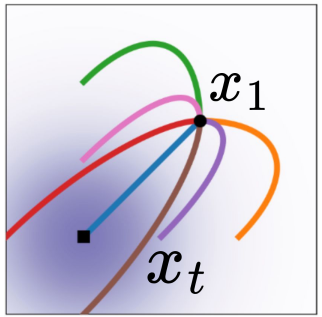
- **Training**: learning the vector field by denoising samples from a chosen conditional probability path

$$\mathcal{L}(\theta) = -\mathbb{E}_{t\sim\mathcal{U}[0,1],\, x_1\sim q_{\text{data}},\, x_t\sim p(x_t|x_1)} \left[\log q_\theta\left(x_1 \mid x_t\right)\right]$$

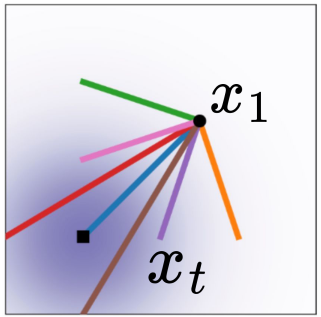
$$v_\theta(x_t) = \mathbb{E}_{x_1\sim q_\theta(x_1|x_t)} \left[u\left(x_t \mid x_1\right)\right]$$

Flow-matching vs Diffusion

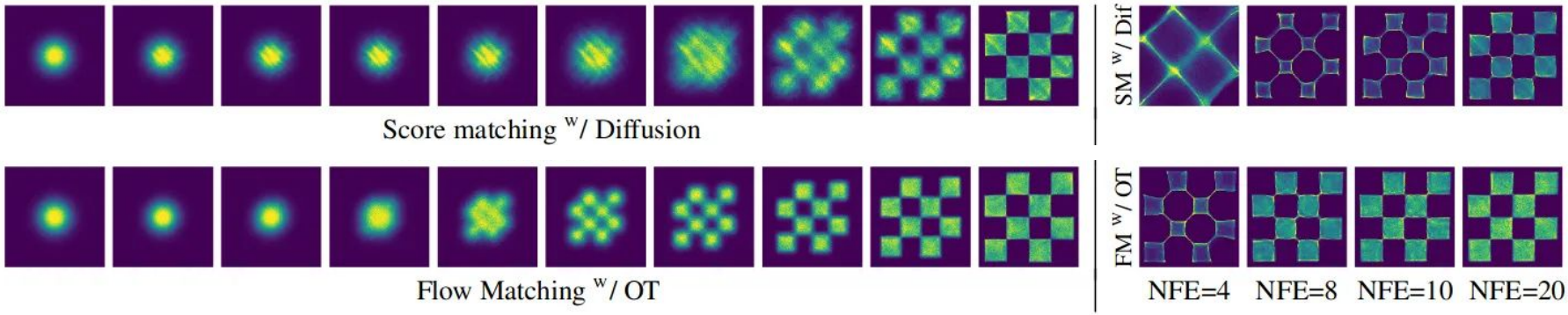
- You can choose linear paths, as the optimal transport map (OT)
- Map is deterministic (ODE)
- More efficient sampling and training



Diffusion



OT



Lipman, Yaron, et al. "Flow matching for generative modeling." *arXiv preprint arXiv:2210.02747* (2022).

Flow Matching for Relational Data

Training

- Define $p(\text{features}_t \mid \text{features}_1)$ using OT Gaussian noise, independent for each feature
- Learn a denoiser $q_\theta(\text{features}_1 \mid \text{features}_t, \text{fk-graph})$ by maximum likelihood

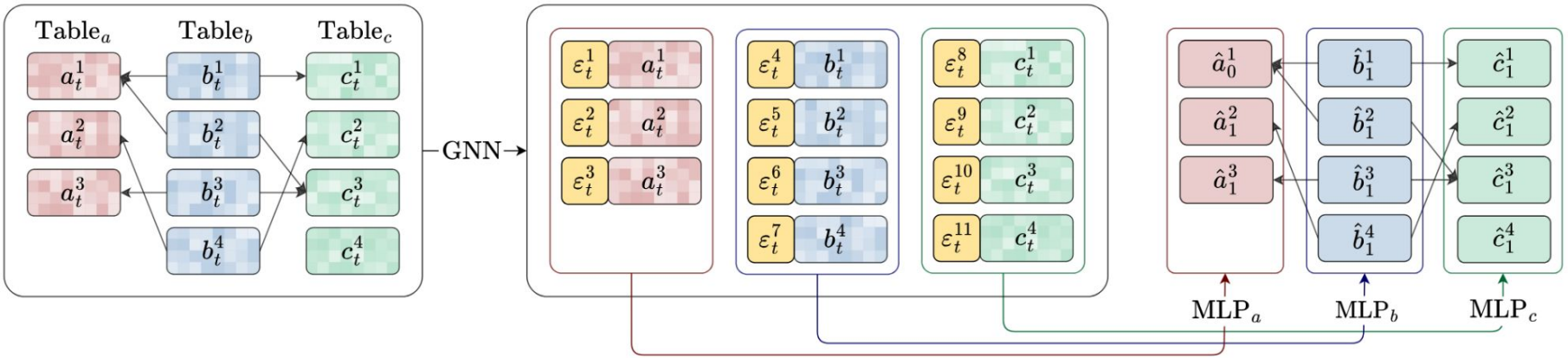
Sampling

1. Generate the foreign-key graph (we just re-sample connected components from the original graph)
2. Initialize every node with Gaussian noise
3. Solve the ODE using the learned velocity, parametrized by the learned denoiser (we use the Euler's method)

Denoiser Architecture

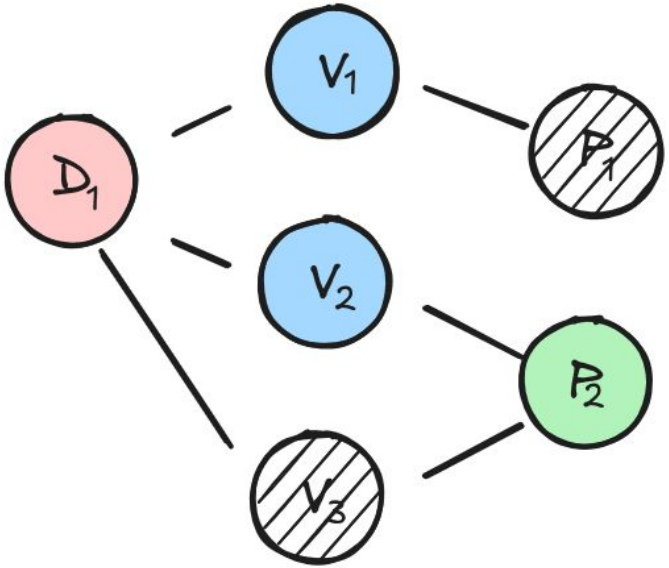
- A GNN computes node embeddings for each record
- MLPs use noisy records and node embeddings to predict the clean record

$$\mathbb{E}[p_\theta(x_1^i | \text{graph}_t)] = \text{MLP}_{\theta_2}(x_t^i, \text{GNN}_{\theta_1}(\text{graph}_t)^i)$$



Controlling Generalization Error

- We randomly split the nodes between train and validation nodes
- During training don’t compute loss for validation nodes
- Validation loss used for early stopping
- “Within sample” generalization



Results

- Metric: accuracy of an XGBoost classifier, that learned to distinguish real from generated rows
- Rows are enriched with aggregated informations from connected rows
- State-of-the-art performances

	AirBnB	Biodegradability	CORA	IMDB	Rossmann	Walmart
Ours	$\mathbf{0.58 \pm 0.03}$	$\mathbf{0.59 \pm 0.02}$	0.63 ± 0.02	$\mathbf{0.59 \pm 0.03}$	$\mathbf{0.51 \pm 0.01}$	$\mathbf{0.73 \pm 0.01}$
Ours (no GNN)	0.70 ± 0.005	0.86 ± 0.004	0.62 ± 0.004	0.89 ± 0.002	0.75 ± 0.01	0.91 ± 0.04
Hudovernik [20]	0.67 ± 0.003	0.83 ± 0.01	$\mathbf{0.60 \pm 0.01}$	0.64 ± 0.01	0.77 ± 0.01	0.79 ± 0.04
ClavaDDPM [36]	≈ 1	-	-	0.83 ± 0.004	0.86 ± 0.01	0.74 ± 0.05
RCTGAN [15]	0.98 ± 0.001	0.88 ± 0.01	0.73 ± 0.01	0.95 ± 0.002	0.88 ± 0.01	0.96 ± 0.02
REaLTabF. [43]	≈ 1	-	-	-	0.92 ± 0.01	≈ 1
SDV [38]	≈ 1	0.98 ± 0.01	≈ 1	-	0.98 ± 0.003	0.90 ± 0.03

Results

- **Privacy**: Despite copying/resampling the foreign-key graph, we didn't detect privacy leaks in the features (checking the Distance to Closest Record statistic)
- **Efficiency**: Training and generation are fast (on a NVIDIA RTX A5000)

Dataset Name	Running Time
AirBnB	10m 3s
Biodegradability	1m 6s
CORA	3m 10s
IMDB MovieLens	14m 25s
Rossmann	2m 57s
Walmart	1m 48s

Our Contribution

Expressiveness

- We model whole connected components (no independence assumptions) with flow matching + GNN
- Generalization achieved through: modular denoiser, GNN embedding size bottleneck, controlling “within sample” generalization

Flexibility

- GNN supports any kind of graph

Scalability

- Flow matching scales with large dimensionalities
- Avoid dealing with dense adjacency matrix; GNN scales since number of edges is proportional to number of nodes

Limitations

- Not a foreign-key graph generation method
- Input of the GNN is a whole connected component, it has to fit in memory
- Hyperparameter tuning for each dataset

Other people are working on this…

Joint Relational Database Generation via Graph-Conditional Diffusion Models

Mohamed Amine Ketata, David Lüdke, Leo Schwinn, Stephan Günnemann

School of Computation, Information and Technology & Munich Data Science Institute
Technical University of Munich, Germany
Correspondence to: a.ketata@tum.de

Abstract

Building generative models for relational databases (RDBs) is important for applications like privacy-preserving data release and augmenting real datasets. However, most prior work either focuses on single-table generation or relies on autoregressive factorizations that impose a fixed table order and generate tables sequentially. This

Thank you!

Relational Data Generation with Graph Neural Networks and Latent Diffusion Models, Table Representation Learning Workshop at NeurIPS 2024.

Hudovernik, Valter, Martin Jurkovič, and Erik Štrumbelj. "Benchmarking the Fidelity and Utility of Synthetic Relational Data." *arXiv preprint arXiv:2410.03411* (2024).

Lipman, Yaron, et al. "Flow matching for generative modeling." *arXiv preprint arXiv:2210.02747* (2022).

Xu, Kai, et al. "Synthetic Data Generation of Many-to-Many Datasets via Random Graph Generation." *The Eleventh International Conference on Learning Representations*. 2022.

Guo, Xiaojie, and Liang Zhao. "A systematic survey on deep generative models for graph generation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.5 (2022): 5370-5390.

Eijkelboom, Floor, et al. "Variational Flow Matching for Graph Generation." *arXiv preprint arXiv:2406.04843* (2024).

Jolicoeur-Martineau, Alexia, Kilian Fatras, and Tal Kachman. "Generating and imputing tabular data via diffusion and flow-based gradient-boosted trees." *International Conference on Artificial Intelligence and Statistics*. PMLR, 2024.

OT conditional flow

$$
\begin{aligned}
p(x_t \mid x_1) &= \mathcal{N}(tx_1, (1-t)I) \\
\varphi_t(x_0 \mid x_1) &= x_t = tx_1 + (1-t)x_0 \\
u_t(x_t \mid x_1) &= \frac{d}{dt}\varphi_t(x_0) = x_1 - x_0
\end{aligned}
$$

Dataset	Table	# Rows	# Features	Foreign Keys
AirBnB	users	10,000	15	–
	sessions	47,217	5	users
Biodegradability	molecule	328	3	–
	group	1,736	1	–
	atom	6,568	1	molecule
	gmember	6,647	–	atom, group
	bond	6,616	1	atom1, atom2
CORA	paper	2,708	1	–
	content	49,216	1	paper
	cites	5,429	–	paper1, paper2
IMDB MovieLens	users	6,039	3	–
	movies	3,832	4	–
	actors	98,690	2	–
	directors	2,201	2	–
	ratings	996,159	1	movie, user
	movies2actors	138,349	1	movie, actor
	movies2directors	4,141	1	movie, director
Rossmann	store	1,115	9	–
	historical	57,970	7	store
Walmart	stores	45	2	–
	features	225	11	store
	depts	15,047	4	store

2.2 The flow matching objective

Since direct access to the vector field v_t of a CNF generating the data is unavailable, we cannot directly match a parameterized velocity field v_t^θ against the ground truth v_t. The main idea of flow matching is instead to define the underlying probability path as a mixture of conditional "per-example" probability paths, that can be defined in a tractable way.

Let us denote by $p_t(\boldsymbol{x}|\boldsymbol{x}_1)$ a conditional probability path such that $p_0(\boldsymbol{x}|\boldsymbol{x}_1) = p_0(\boldsymbol{x})$ and $p_1(\boldsymbol{x}|\boldsymbol{x}_1)$ is a distribution concentrated around $\boldsymbol{x} = \boldsymbol{x}_1$, as $p_1(\boldsymbol{x}|\boldsymbol{x}_1) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{x}_1, \sigma^2 I)$ with σ small. We define the conditional velocity $u_t(\boldsymbol{x} \mid \boldsymbol{x}_1)$ as the velocity generating the conditional probability path $p_1(\boldsymbol{x}|\boldsymbol{x}_1)$. It is then possible to prove that the marginal velocity, defined as

$$u_t(\boldsymbol{x}) = \int u_t(\boldsymbol{x} \mid \boldsymbol{x}_1) p_t(\boldsymbol{x}_1|\boldsymbol{x}) d\boldsymbol{x}_1 = \int u_t(\boldsymbol{x} \mid \boldsymbol{x}_1) \frac{p_t(\boldsymbol{x} \mid \boldsymbol{x}_1) q(\boldsymbol{x}_1)}{p_t(\boldsymbol{x})} d\boldsymbol{x}_1$$

generates the marginal probability path

$$p_t(\boldsymbol{x}) = \int p_t(\boldsymbol{x} \mid \boldsymbol{x}_1) q(\boldsymbol{x}_1) d\boldsymbol{x}_1$$

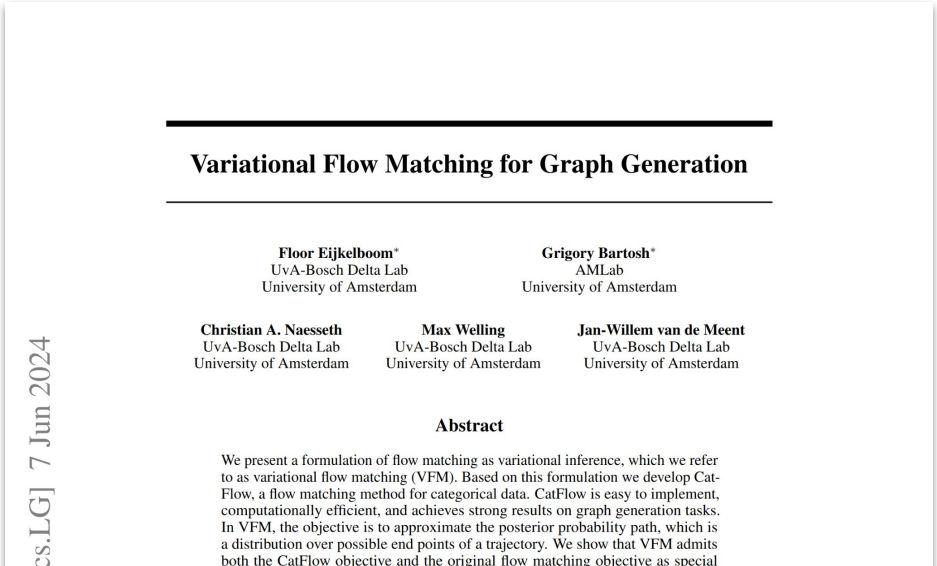
that, following the definition of conditional probability path, at $t = 1$ closely matches the target distribution $q(\boldsymbol{x})$. Moreover, it can be shown that a valid loss for learning the marginal velocity is the following:

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t \sim \mathcal{U}[0,1], \boldsymbol{x}_1 \sim q(\boldsymbol{x}_1), \boldsymbol{x} \sim p_t(\boldsymbol{x}|\boldsymbol{x}_1)} \| v_t^\theta(\boldsymbol{x}) - u_t(\boldsymbol{x} \mid \boldsymbol{x}_1) \|^2$$

known as the conditional flow matching objective, where a neural network learns a match the marginal velocity by matching conditional velocities.

Variational Parametrization

- Instead of learning the velocity x1 -x0 given xt by MSE, learn p(x1 | xt) by maximum likelihood and parametrize a distribution
- Improved learning for one-hot-encoded categorical variables



Variational Flow Matching for Graph Generation

Floor Eijkelboom*
UvA-Bosch Delta Lab
University of Amsterdam

Grigory Bartosh*
AMLab
University of Amsterdam

Christian A. Naesseth
UvA-Bosch Delta Lab
University of Amsterdam

Max Welling
UvA-Bosch Delta Lab
University of Amsterdam

Jan-Willem van de Meent
UvA-Bosch Delta Lab
University of Amsterdam

Abstract

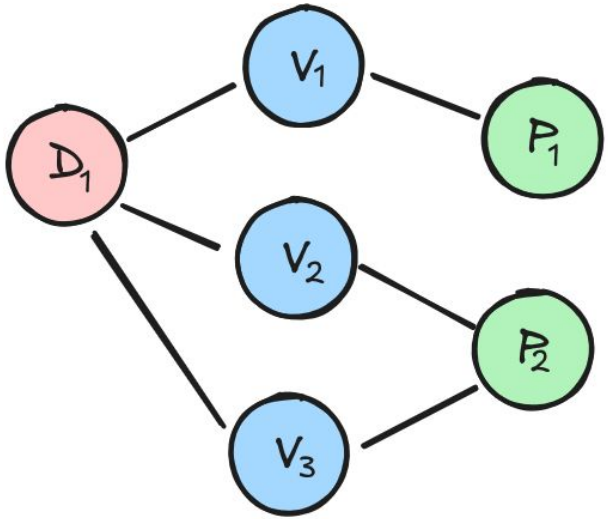
We present a formulation of flow matching as variational inference, which we refer to as variational flow matching (VFM). Based on this formulation we develop CatFlow, a flow matching method for categorical data. CatFlow is easy to implement, computationally efficient, and achieves strong results on graph generation tasks. In VFM, the objective is to approximate the posterior probability path, which is a distribution over possible end points of a trajectory. We show that VFM admits both the CatFlow objective and the original flow matching objective as special

First step: Tables -> Heterophilic Graph

Doctors	
Id: 1	features: $\mathbf{d}_1$

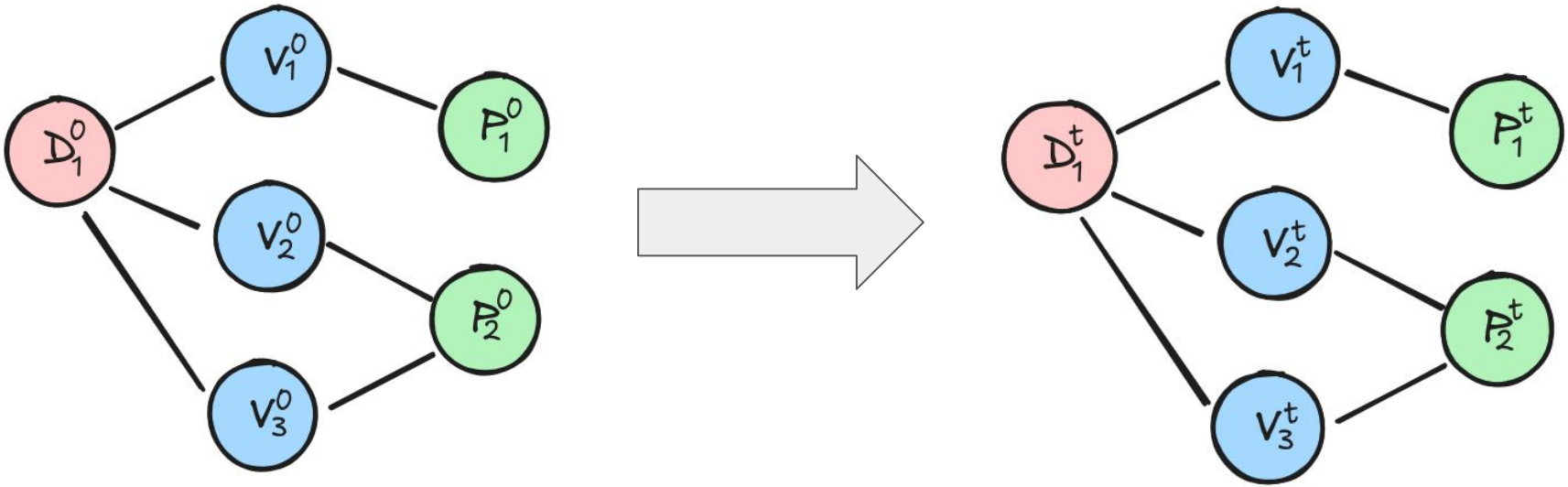
Patients	
Id: 1	features: $\mathbf{p}_1$
Id: 2	features: $\mathbf{p}_2$

Visits			
Id: 1	Dr: 1	Pt: 1	features: $\mathbf{v}_1$
Id: 2	Dr: 1	Pt: 2	features: $\mathbf{v}_2$
Id: 3	Dr: 1	Pt: 2	features: $\mathbf{v}_3$



Training: Graph diffusion

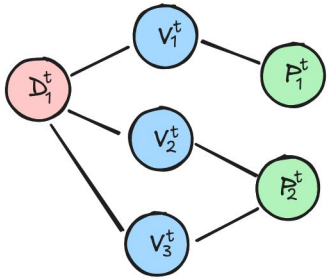
- Sample a noise level t from a uniform distribution [0 (no noise), 1 (only noise)]
- Sample the noisy version for each node/row with noise level t

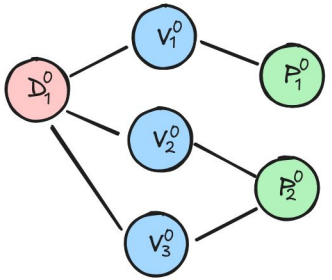


Training: Loss

- The denoiser takes the noisy graph as input, and tries to predict the original one
- Loss is MSE node-by-node for continuous features, cross-entropy for one-hot-encoded features

$$\text{Loss}(\ \text{NN}(\ \ ),\ \ )$$



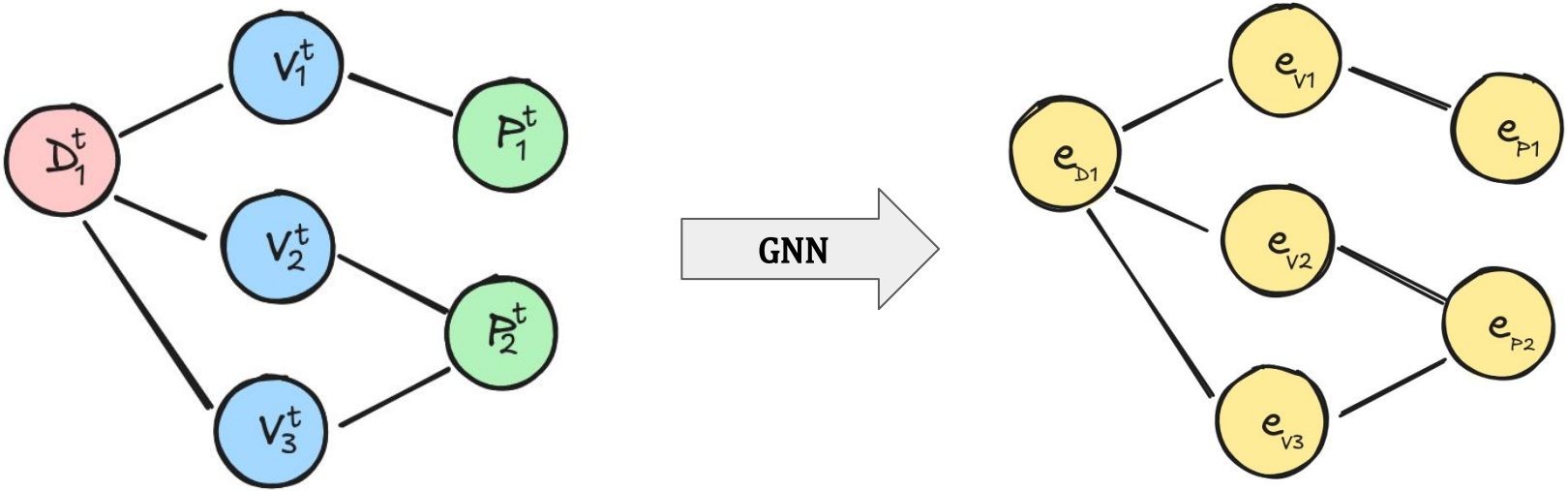


Denoised graph

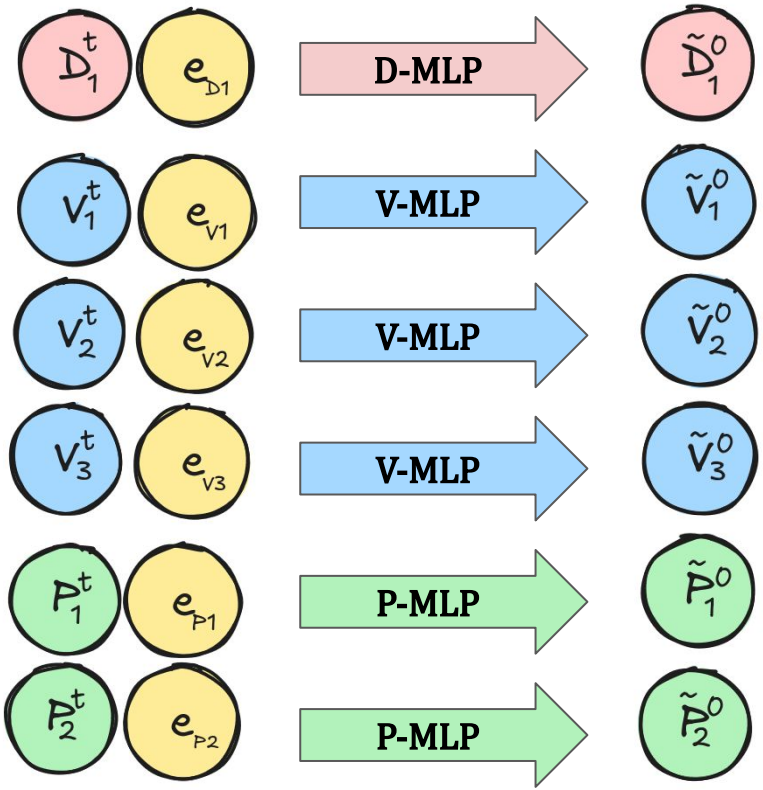
Original graph

Denoising architecture: GNN

- Use a GNN to compute node embeddings. It's the only moment where graph information is used.
- The GNN has to be "heterophilic"
- The larger size of the node embedding the more informative



Denoising architecture: row-wise denoisers



- For each node/row, use a denoiser specific for the table, conditioned on node embedding to incorporate neighbor information
- At the moment I'm using plain MLPs
- This part of the denoising can be done in parallel (also batched among same table nodes)

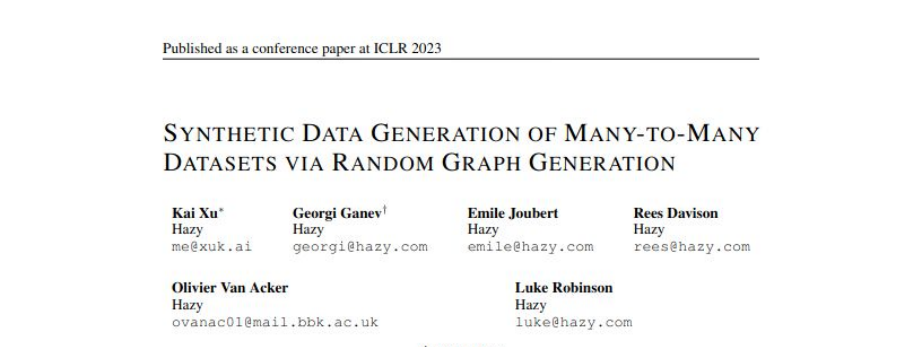
Hazy

$$p(\mathbb{L})p(\mathbb{U} \mid \mathbb{L})p(\mathbb{V} \mid \mathbb{U}, \mathbb{L})$$

- $p(\mathbb{L})$ (the edge model) requires to model the edges (i.e. the graph) unconditionally—this is the type of random graph models that usually studied in random graph theory.
- $p(\mathbb{U} \mid \mathbb{L})$ (the edge-conditional table model) requires to generate one of the table given the topology of edges. One way to achieve such conditioning is by using a node embedding to condition on. This is also related to node attributes generation/prediction in graph models.
- $p(\mathbb{V} \mid \mathbb{U}, \mathbb{L})$ (the conditional table model) requires to generate each node in $\mathbb{V}$ based on the first table and all connections. As per node in $\mathbb{V}$ is connected to a subset of nodes in $\mathbb{U}$ via $\mathbb{L}$,

Presentation

Xu, Kai, et al. "Synthetic Data Generation of Many-to-Many Datasets via Random Graph Generation." *The Eleventh International Conference on Learning Representations*. 2022.



Published as a conference paper at ICLR 2023

SYNTHETIC DATA GENERATION OF MANY-TO-MANY DATASETS VIA RANDOM GRAPH GENERATION

Kai Xu* Hazy me@xuk.ai

Georgi Ganev† Hazy georgi@hazy.com

Emile Joubert Hazy emile@hazy.com

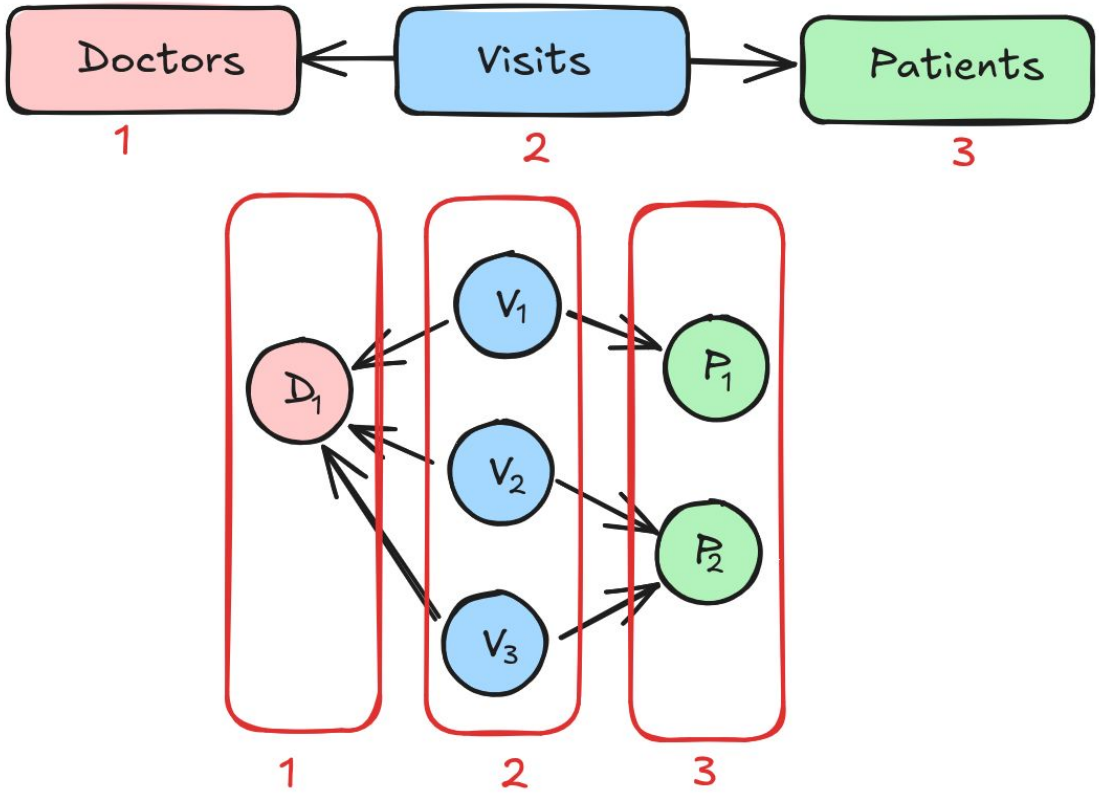
Rees Davison Hazy rees@hazy.com

Olivier Van Acker Hazy ovanac01@mail.bbk.ac.uk

Luke Robinson Hazy luke@hazy.com

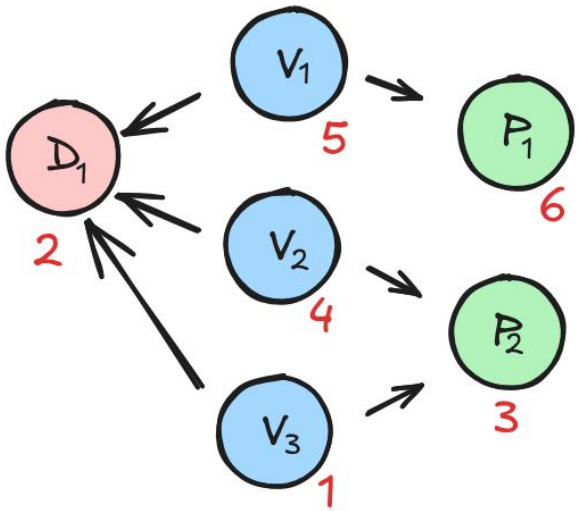
Hazy

1. Generate graph structure with a statistical method
2. Generate a table using an AR conditional model, where the conditioning are statistics about the topology (ex: the number of edges)
3. Generate a new table, conditioning on topology statistics and aggregating features from already “filled” rows from generated tables
4. repeat 3 until every table is generated



Pre-generated topology?

- Idea: first generate the empty graph (topology) with another method, then fill nodes with the content one-by-one.
  - Actually, in the Hazy paper they do this but they generate all the rows of a table at the same time, making it not fully autoregressive
- The experiment I tried with AR:
  - Start from original dataset topology (just keys) and then fill node-by-node conditioning on the rest of nodes using a GNN conditioning table-specific ARs.
  - Too slow, it requires to compute node embeddings n times, where n is the number of nodes in the dataset.
  - The reason is similar to why you cannot do AR with images, a single data point has too many components. In this case the single data point is the dataset.



Is diffusion good for tabular data?

- Unfortunately plain diffusion struggles with tabular data, from personal experience with single table datasets AR works better
- Diffusion is good for capturing correlations, but shows limits when data has multi modal marginals or complex shapes
- Also diffusion with discrete data is not trivial
- Some people used trees for diffusion on tabular data, but I need a GNN and at the moment I don't think there are tree based methods able to deal with graphs

Jolicoeur-Martineau, Alexia, Kilian Fatras, and Tal Kachman. "Generating and imputing tabular data via diffusion and flow-based gradient-boosted trees." *International Conference on Artificial Intelligence and Statistics*. PMLR, 2024.