Thèse n° 9880

Equivariant Neural Architectures for Representing and Generating Graphs

Présentée le 1er septembre 2023

Faculté des sciences et techniques de l'ingénieur
Laboratoire de traitement des signaux 4
Programme doctoral en génie électrique

pour l'obtention du grade de Docteur ès Sciences

par

Clément Arthur Yvon VIGNAC

Acceptée sur proposition du jury

Prof. D. N. A. Van De Ville, président du jury
Prof. P. Frossard, directeur de thèse
Prof. S. Günnemann, rapporteur
Prof. Y. Lipman, rapporteur
Prof. P. Vandergheynst, rapporteur

École
polytechnique
fédérale
de Lausanne

2023

Acknowledgements

I'd first like to thank my supervisor, Pascal Frossard. Pascal, your trust in me has been a defining feature of my PhD journey. You advised me with goodwill, letting me choose topics and collaborators, but still asking the right questions to help me choose research directions.
To my PhD committee - Stephan Günnemann, Yaron Lipman, Pierre Vandergheynst, and Dimitri Van de Ville - thank you for dedicating your time and efforts to review my work, and for encouraging me to pursue my research.
I'm deeply appreciative of Max Welling for his warm welcome into his lab and for fostering collaborations that became integral to my research. The time spent in your lab was pivotal and greatly enriched my PhD experience.
A huge thanks to Renata Khasanova, Bastien Pasdeloup, Michaël Defferrard, and Roberto Azevedo, who all mentored me at the beginning of my PhD. Your insights helped me understand my research area better and also introduced me to the unique facets of PhD life, for which I am deeply grateful.
To Andreas, Ilia, Nagham, and Laura, thank you for our collaborations. It was a pleasure to work with you, and I hope that we will have other opportunities to work together.
To all the members of LTS4, both past and present, thank you for sharing this journey with me. Despite the hurdles of Covid, we have maintained an uplifting spirit, engaged in great social activities, and fostered a positive spirit of collaboration that I hope will continue to thrive. A particular thanks to Guillermo, Tolis, Beril, Eda and Nikos, who did their PhD around the same time as me and with whom I spent so much time.
Special thanks to Anne, our lab secretary. Your dedication has made our lab feel like part of a top-tier university. Your work truly makes a difference.
To my friends in Lausanne: Alexandre, Violette, Matthieu, Halla, Vincent, Quentin, Anne-Claire, Sebastien, and the Frouzes team — I will miss you all so much, it was great to spend these years together. Thank you, Eloi, for our shared adventures in the mountains, which are memories that I will never forget.
Thank you to all my roommates, past and present. I was glad to have friends to talk to when going back home, and I have great memories with all of you.
And finally, to Virginie and my family, thank you for your enduring support and faith in me. Your backing has been a constant source of strength and motivation throughout this journey.

Lausanne, July 14th, 2023 C.V.

Abstract

Graph machine learning offers a powerful framework with natural applications in scientific fields such as chemistry, biology and material sciences. By representing data as a graph, we encode the prior knowledge that the data is composed of a set of entities, that the interactions between these entities are as informative as their individual properties, and that the task at hand does not depend on their ordering. While graph representations allow for effective learning, building graph neural networks that are both expressive and efficient poses important challenges.

The primary focus of this dissertation is to advance the state-of-the-art in building permutation equivariant neural networks capable of handling large-scale data. To achieve this, we will first develop our understanding of permutation equivariance and its practical implications, and then propose novel algorithms that exploit the strengths of graph neural networks while circumventing challenging problems such as graph matching.

In our first contribution, we propose the Structural Message-Passing (SMP) model. SMP introduces node identifiers in the form of a one-hot encoding, and processes them in a permutation equivariant way. The use of identifiers confers to the network a higher expressive power than standard Message-Passing Neural Networks, but retains its inductive bias towards learning local functions. Empirically, SMP demonstrates superior generalization capabilities compared to alternative expressive architectures that do not leverage the message-passing scheme.

While architectures for learning representations of unordered data are well studied, we show in the next contributions that architectures for graph generation feature on the contrary still many open questions. We begin by examining an important component of many set and graph generation architectures, namely operators that transform vectors to sets. As permutations transform sets but not vectors, vector-to-set layers constitute a challenge for standard equivariance theory, which cannot be used when a group acts trivially on a function input. To address this challenge, we analyze the equivariance requirements in this setting and propose a novel vector to set layer called the *Top-n creation* layer. This layer can be plugged into several architectures as a replacement to other vector-to-set functions. We show that it improves generation quality on several set and graph generation tasks.

We then show that performance can greatly be improved by building denoising diffusion models, i.e., architectures that iterative denoise a random set or graph. Specifically, we propose the *DiGress* model for graph generation and *MiDi* for jointly generating the 2D and 3D structure of molecules. These models exploit the discrete structure of graphs and are able to preserve their sparsity during diffusion. As a result, they can successfully model larger graphs than previous Gaussian diffusion models.

Overall, the presented research shows that representing and generating graph data requires careful consideration. Using architectures developed for other data modalities, such as images, can lead to serious theoretical problems that are as demanding as isomorphism testing. By identifying these limitations, we are able propose novel methods that are more effective, paving the way to novel applications of graph neural networks in various scientific fields.

Keywords: Permutation Equivariance, Graph Neural Networks, Message-Passing Neural Networks, Graph Generation, Denoising Diffusion Models.

Résumé

L'apprentissage automatique sur les graphes offre un cadre très général qui trouve des applications naturelles dans des domaines scientifiques tels que la chimie, la biologie et la science des matériaux. En représentant des données sous forme de graphes, nous codons la connaissance préalable que les données sont composées de plusieurs entités, que les interactions entre ces entités sont aussi importantes que leurs propriétés individuelles, et que la tâche à accomplir ne dépend pas de leur ordre. Si les représentations graphiques permettent un apprentissage efficace, la construction de réseaux de neurones pour les graphes qui sont à la fois expressifs et efficaces pose cependant d'importants défis.

L'objectif principal de cette thèse est de faire progresser l'état de l'art en matière de réseaux de neurones équivariants aux permutation capables de traiter des données à grande échelle. Pour y parvenir, nous développerons d'abord notre compréhension des implications pratiques de l'equivariance aux permutations, puis nous proposerons de nouveaux algorithmes qui exploitent les forces des réseaux de neurones sur les graphes, tout en contournant des problèmes difficiles tels que les tests d'isomorphisme de graphes.

Dans notre première contribution, nous proposons le modèle Structural Message-Passing (SMP). Le modèle SMP introduit des identifiants à chaque noeud sous la forme d'un codage unique, et les traite de manière équivariante aux permutations. L'utilisation d'identifiants confère au réseau un pouvoir expressif supérieur à celui des réseaux neuronaux à échange de messages classique, mais il conserve son biais inductif en faveur de l'apprentissage de descripteurs locaux des noeuds. Empiriquement, SMP démontre des capacités de généralisation supérieures à celles d'autres architectures expressives qui n'exploitent pas le schéma d'échange de messages.

Alors que les architectures pour l'apprentissage sur des données non ordonnées sont bien étudiées, nous montrons dans les contributions suivantes que les architectures pour générer de graphes soulèvent au contraire de nombreuses questions. Nous commençons par examiner un élément important de nombreuses architectures de génération d'ensembles et de graphes, à savoir les opérateurs qui transforment les vecteurs en ensembles. Comme les permutations transforment les ensembles mais pas les vecteurs, les couches "vecteur-vers-ensemble" constituent un défi pour la théorie standard de l'équivariance, qui ne peut pas être utilisée lorsqu'un groupe agit trivialement

sur l'entrée d'une fonction. Pour relever ce défi, nous analysons les exigences d'équivariance dans ce contexte et proposons une nouvelle couche "vecteur-vers-ensemble" appelée couche de *Création Top-n*. Cette couche peut être intégrée dans plusieurs architectures et d'autres fonctions "vecteur-vers-ensemble". Nous montrons qu'elle améliore la qualité de la génération dans plusieurs tâches de génération d'ensembles et de graphes.

Nous montrons ensuite que les performances peuvent être grandement améliorées en construisant des modèles de diffusion, c'est-à-dire des architectures qui débruitent de manière itérative un ensemble aléatoire ou un graphe. Plus précisément, nous proposons le modèle *DiGress* pour la génération de graphes et *MiDi* pour la génération conjointe de la structure 2D et 3D des molécules. Ces modèles exploitent la structure discrète des graphes et sont capables de préserver leur parcimonie pendant la diffusion. Par conséquent, ils peuvent modéliser avec succès des graphes plus grands que les modèles de diffusion gaussiens existants.

À travers les contributions présentées, ces recherches montrent que la représentation et la génération de graphes nécessitent une attention particulière. L'utilisation d'architectures développées pour d'autres modalités de données, comme les images, peut conduire à de sérieux problèmes théoriques tels que les tests d'isomorphisme de graphes. En identifiant ces limitations, nous sommes en mesure de proposer de nouvelles méthodes plus efficaces qui ouvrant la voie à de nouvelles applications des réseaux neuronaux pour les graphes graphes dans divers domaines scientifiques.

Mots clefs :
Equivariance aux permutations, réseaux de neurones sur les graphes, réseaux de neurones à échange de message, génération de graphes, modèles de diffusion.

Contents

List of Figures

List of Tables

Notations

Notation	Tensor size	Description
$G = (\mathbb{V}, E)$		A graph with n nodes and m edges
$N(v_i)$		Neighbors of node v_i
$\boldsymbol{A}$	$n \times n$	Adjacency matrix of G
$\boldsymbol{D}$	$n \times n$	Degree matrix: $\boldsymbol{D} = \mathrm{diag}((d_i)_{i=1}^n)$
$\boldsymbol{w}$	d_w	Vector aggregating graph-level features
$\boldsymbol{X}$	$n \times d_x$	Matrix aggregating node features
Y	$n \times n \times d_y$	Tensor aggregating edge features
$\boldsymbol{R}$	$n \times 3$	Matrix aggregating the 3D coordinates $[\boldsymbol{r}_i]_{i=1}^n$ of a point cloud
$\boldsymbol{I}$	$n \times n$	Identity matrix of size n
$\mathbf{1}_n$	n	Vector containing ones
$\mathbb{1}_i$	n	Indicator vector of element i
$\mathbb{G}$		A group
$\mathbb{S}_n$		Symmetric group with n elements
$SE(k)$		Special Euclidean group in dimension k
$E(k)$		Euclidean group in dimension k
$\{\boldsymbol{x}_i\}_{1 \leq i \leq n}$		A *multi*-set with n elements
σ		ReLU non-linearity: $\sigma(x) = (x, 0)_+$

Vectors are represented by bold characters ($\boldsymbol{w}$), matrices by capital bold characters ($\boldsymbol{A}$) and higher dimensional tensors by sans-serif bold letters (Y). Blackboard bolds are used for sets ($\mathbb{R}$).

1 Introduction

Graphs machine learning has emerged as a powerful and versatile tool in machine learning, gaining popularity across a wide range of applications. Originally employed for sensor and social network analysis, graph machine learning is now being used increasingly in scientific domains such as chemistry, computational biology, and material sciences. The advantage of representing data as a graph lies in the invariance that graphs naturally encode, particularly the assumption that the data ordering is irrelevant for the task at hand. This prior is beneficial for many real-world systems, including molecules, n-body systems, agents in a game, social networks, and objects in a scene. However, computers require graphs to be represented with ordered tensors such as adjacency matrices or adjacency lists. This presents a critical challenge. Permutation equivariance, namely, the need to process unordered objects using an ordered representation, lies at the core of graph machine learning and its associated difficulties.

To leverage the power of graph-based representations, specialized neural architectures are needed. Arguably, it is theoretically possible to use a multi-layer perceptron (MLP) to accomplish graph-based tasks, as it is a universal approximator of continuous functions on compact spaces (Cybenko, 1989). This approach is however limited in practice (Zaheer et al., 2017), as MLPs do not encode a lot of prior knowledge about the task at hand (Xu et al., 2020b). As continuity and smoothness do not constitute sufficient priors to overcome the curse of dimensionality (von Luxburg & Bousquet, 2004), we can expect the amount of data required to learn graphs with MLPs to be exponential in the graph size. This strong limitation calls for the development of graph neural networks that properly constrain the hypothesis class by leveraging the unordered nature of graphs.

Graph neural networks (GNNs) have emerged as a common framework for learning on graph-structured data. GNNs have historically been constructed based on the definition of a Fourier transform for graphs (Bruna et al., 2013). These spectral filters were then formulated as Laplacian polynomials (Defferrard et al., 2016; Khasanova & Frossard, 2017), which can be implemented using an iterative propagation scheme. This perspective led to the rediscovery of Message-Passing Neural Networks (MPNNs) (Gilmer et al., 2017), which had originally been proposed in the pioneering work of (Scarselli et al., 2008). MPNNs have a computational complexity which

is linear in the number of edges, and they compute local descriptors of the data similarly to convolutional kernels for images. Thanks to these beneficial properties, MPNNs have achieved impressive performance on various tasks, including semi-supervised classification in social networks (Kipf & Welling, 2016) and physical simulations (Garcia & Bruna, 2017). Theoretically, graph neural networks have a sample complexity that only grows quadratically in the graph size (Scarselli et al., 2018; Garg et al., 2020), making them more suitable than MLPs for learning on graphs.

Although graph neural networks have shown impressive performance on various tasks, their limitations have been identified, particularly on graphs with limited attributes (Cai & Wang, 2018). For example, they perform surprisingly poorly on tasks that require to learn structural properties of graphs such as their connectivity (Corso et al., 2020) or cycle counts (Vignac et al., 2020). These limitations are not only observed empirically, but are rooted in the analysis of the representation power of MPNNs. Specifically, Morris et al. (2019) and Xu et al. (2019) have shown that MPNNs can only learn a restricted class of functions on graphs by proving that they are not more powerful at graph isomorphism testing than the 1-Weisfeiler-Lehman test (1-WL) proposed in (Weisfeiler & Lehman, 1968). This means that, for any two graphs that the 1-WL test considers as isomorphic, MPNNs will output the same embedding. Among other consequences, this result implies that MPNNs have limited ability to count substructures in graphs (Chen et al., 2020). The study of the representation power of MPNNs has led to the development of more powerful architectures that typically consider triplets of nodes (Morris et al., 2019) or perform message-passing on larger structures than typical MPNNs (de Haan et al., 2020). These architectures gave rise to rich theoretical analysis (Morris et al., 2021), but they usually do not incorporate the important locality prior of MPNNs, which eventually restricts their performance.

The expressive power of graph neural networks is not their only limitation. It is also surprisingly challenging to build efficient graph coarsening functions (Mesquita et al., 2020) or rich global pooling layers (Corso et al., 2020). Furthermore, on several datasets with rich node attributes, MPNNs do not perform better than simple spectral filters (Wu et al., 2019). While some of these results come from the specific properties of the datasets used in standard benchmarks, other constitute fundamental limitations of MPNNs and permutation equivariant networks.

While methods and open problems in graph representation learning are now relatively well understood, graph generation is in contrast still at its infancy. Yet, it has important applications in fields such as drug discovery (Xia et al., 2019), computational biology (Yang et al., 2021a), catalyst design (dos Passos Gomes et al., 2021), code completion (Brockschmidt et al., 2019), and circuit design (Brophy & Voigt, 2014). Graph generation can be used in both distribution learning settings, where the goal is to generate graphs similar to those in the training set, and in goal-oriented settings where a specific property needs to be optimized under constraints. In both cases, correctly capturing the training set distribution is a crucial prerequisite.

Graph generation shares the challenges of graph representation learning, but also features its own

difficulties. To understand them, we can compare graph generative architectures to corresponding architectures for image generation. Neural networks for image generation typically feature a "U-net" structure with pooling layers in the encoder and up-sampling layers in the decoder. As neighboring image pixels tend to be very correlated, such layers are often very effective. The U-Net structure permits to define a latent space of reduced dimension where the data representation is compressed, which is an effective strategy for designing generative architectures. In contrast, graphs are unordered, and two neighboring entries of an adjacency matrix cannot be assumed to be correlated. Graph coarsening and graph up-sampling is therefore significantly more difficult for graphs than for images. As a result, graphs are typically hard to compress in a vector or in a smaller graph.

Theoretically, we can observe that, if we had access to a permutation invariant graph-to-vector model and a deterministic decoder that reconstructs the input graph from its latent vector, we would have access to a graph canonization algorithm. Graph canonization being at least as hard as graph isomorphism testing, it is not surprising that early graph generation architectures that used a low-dimensional latent space could only be successful at generating tiny graphs (Simonovsky & Komodakis, 2018; De Cao & Kipf, 2018).

In this thesis, we aim at developing our understanding of the practical consequences of permutation equivariance on machine learning tasks. By better understanding the benefits of permutation equivariant functions and the challenges associated to them, we are able to design architectures that leverage the strengths of graph neural networks and rely as little as possible on their weaknesses. These architectural choices can have a huge impact on performance, as exemplified in the comparison between the Set2GraphVAE and DiGress architectures proposed in this thesis. In Chapter 4, we propose a graph generation architecture that can generate 60% of realistic molecules after 6 hours of training on the QM9 dataset, which was close to state-of-the-art at the time of publication. In the model proposed in Chapter 5, we achieve 99% validity in one hour on the same dataset, with most of the improvement coming from another problem formulation that does not require compressing the data in a latent space.

In the rest of this introduction, we summarize the contributions of this thesis. In Chapter 2, we introduce background on permutation equivariant neural architectures, and recall known results about their representation power.

Chapter 3 then answers the following question: how can we design more expressive architectures that retain the inductive bias of MPNNs, and when are such architectures needed? We propose the Structural Message-Passing framework (SMP), an equivariant message-passing architecture that overcomes the limitations of MPNNs at the cost of manipulating larger order tensors. SMP is the first architecture based on the message-passing framework that achieves a higher representation power without introducing randomness in the network. In contrast to other powerful architectures, it retains the locality prior of MPNNs, which empirically results in better generalization properties.

We then turn our attention to graph generation architectures. In Chapter 4, we first study general

formulations of equivariance in generative settings. We then propose the *Top-n* creation module, which is a layer to transform vectors to set. We show on various architectures that performance can be improved by replacing other set creation methods by our module.

Finally, we propose in Chapter 5 two denoising diffusion models for graphs. With the first model, DiGress, we show that discrete diffusion is better suited to graph generation than Gaussian diffusion, as it respect the sparsity properties of graphs and their discrete nature. We then propose the MiDi model for generating simultaneously the graph structure and the 3D conformation of molecules. We demonstrate that when both graph and 3D information is simultaneously available, it is clearly beneficial to define one single model that generates jointly both data modalities.

Structural message-passing

Despite their limited expressivity, message-passing neural networks constitute the most used class of graph neural network. Through their iterative propagation scheme, they compute local information around each node, similarly to what convolutional networks do for images. While more powerful architectures such as (Chen et al., 2019) or (Maron et al., 2019a) have been proposed, these networks unfortunately lack the important locality prior of MPNNs.

To gain insight into the limitations of MPNNs, we adopt the perspective of distributed algorithms, where the class of functions that can be executed using iterative propagation schemes is well studied. Standard results in distributed algorithms suggest that any graph algorithm can be executed when nodes are uniquely identified, but the class of executable algorithms is very limited when they are not (Suomela, 2013). These results are applicable to machine learning settings as well. When node features can uniquely distinguish nodes, message-passing networks can be used to build universal approximators of functions on graphs (Loukas, 2020a). However, when dealing with unattributed graphs, the representation power of MPNNs is limited by the Weisfeiler-Lehman test, as observed in (Xu et al., 2019) and (Morris et al., 2019).

To address the limitations of message-passing neural networks on graphs with poor node features, it is therefore natural to introduce identifiers. Previous approaches have used random identifiers, but their performance on complex tasks has been disappointing due to the high level of noise introduced during network training (Murphy et al., 2019; Sato et al., 2020).

Instead, we propose propose the Structural Message-Passing (SMP), which utilizes a one-hot encoding of the nodes as identifiers. As these one-hot encodings depend on the ordering of the nodes, the message-passing layers need to be adapted in order to preserve permutation equivariance and to guarantee that the final output does not depend on this initial ordering. This process is illustrated in Figure 1.1. SMP can be described by the following steps:

- *Initialization* First, to increase the expressive power of the network, we initialize higher-order tensors known as *local contexts* instead of manipulating vectors as in standard message-passing networks. These local contexts are multisets denoted by $\boldsymbol{U} \in \mathbb{R}^{n \times d}$ and

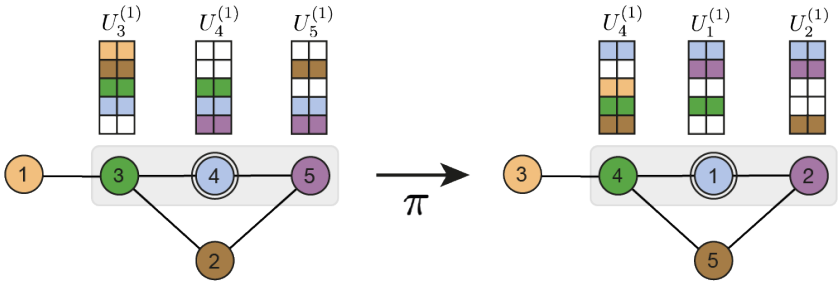


Figure 1.1 – In the structural message-passing model, each local context $\boldsymbol{U}_i^{(l)}$ is an $n \times d$ matrix, with each row storing the d-dimensional representation of a node (denoted by color). The figure shows the local context in the output of the first layer and blank rows correspond to nodes that have not been encountered yet. The network is parameterized in such a way that, upon node reordering, the lines of the local context are permuted but their content remains unchanged.

are initialized with a one-hot encoding of the nodes.

- *Propagation* Similarly to standard MPNNs, each message-passing layer of SMP contains a message function, an aggregation function and an update function. However, while the message and update functions of MPNNs can be chosen arbitrarily, SMP requires that these functions preserve equivariance to transformation of the local contexts. This implies that these functions should be *equivariant functions on sets*.
- *Pooling* After several message-passing layers have been applied, each node carries a local context with rich information. To perform tasks such as node classification, the contexts must be mapped to a vector using a *permutation-invariant* pooling function.

By following this framework, SMP models can learn from graphs with poor node features and achieve high performance on complex tasks. Experimental results show the benefits of both rich expressive power and message-passing inductive bias. We observe SMP can solve tasks that standard MPNNs cannot solve while generalizing better than powerful architectures that do not use the message-passing locality prior.

Equivariance in generative models

The background section and our first contribution show that MPNNs can be interpreted as local permutation equivariant functions, and that equivariant identifiers can be used to improve their representation power. However, as universal function approximation on graphs is equivalent to isomorphism testing (Chen et al., 2019), there is little hope to build architectures that do not have a restricted representation power in practice. Overall, we can therefore consider that the consequences of permutation equivariance on graph neural networks are relatively well understood, and that the main limitations that have been identified are fundamental. In this chapter, we shift our focus towards graph generation and explore permutation equivariance in generative settings.

We start by exploring architectures which feature a probabilistic decoder that maps a latent vector to a set or a graph, such as those proposed by Achlioptas et al. (2018), De Cao & Kipf (2018), Simonovsky & Komodakis (2018), and Kosiorek & Kim (2020). While standard equivariance theory requires the symmetry group to have a non trivial action on the input space, permutations left vectors invariant. A new definition of equivariance is therefore required in this setting.

To address the challenge of defining equivariance in generative settings, we propose a definition that applies to both discriminative and generative architectures. Specifically, we define a pair (architecture, loss function) as equivariant if the updates to the parameters during training on the loss function using gradient descent do not depend on the group elements used to represent the training data. We derive several consequences of this definition:

- For a discriminative architecture to be equivariant, both an equivariant neural network and an invariant loss function are required. An example of a non-invariant loss function is the l_1 loss, which is not suitable for a rotation-equivariant point cloud prediction problem in 3D.
- In generative tasks, equivariance imposes constraints on the loss function, but not always on the decoder. It may not matter what permutations of a generated graph are produced or whether all permutations are equally likely or not.
- Enforcing exchangeability, i.e., requiring that all permutations of a generated graph are equally likely, is mostly needed for likelihood computation in normalizing flows architectures.

These definitions do not really provide novel conditions, but capture standard practice with a single definition that captures both discriminative and generative settings.

Using these observations, and based on a review of layers that have previously been used to map a vector to a set, we propose the novel *Top-n creation* module. This layer, depicted in Fig. 1.2, uses a bank of reference points with trainable representations. Based on the value of the latent vector, n points are chosen in this set. Since this selection process is not differentiable, a modulation mechanism is used to obtain gradients and train the point selection. Top-n creation can be used in replacement for other set creation layers, or combined with a Set2Graph layer (Serviansky et al., 2020) in order to generate graphs. We demonstrate on multiple models that this replacement consistently improves generation quality.

Diffusion models for unordered data

Despite the consistent performance improvements obtained when replacing set creation layers by our Top-n creation module, we observe that generating sets and graphs with a vector-shaped latent space remains challenging. In particular, graph generation methods based on this framework are limited to very small graphs, and do not seem to scale well.

To address this limitation, we investigate generative architectures that take an entirely different

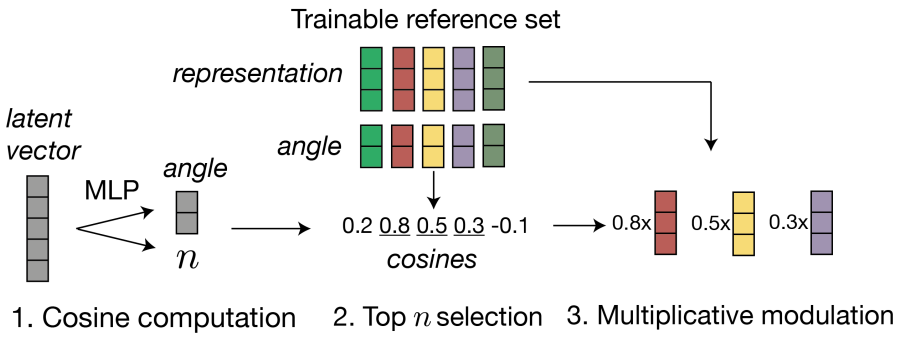


Figure 1.2 – The Top-n creation module learns to select the most relevant points in a trainable reference set based on the value of the latent vector. To obtain gradients and train the angles and the MLP despite the non-differentiable $\mathrm{argsort}$ operation, we modulate the selected representations with the values of the cosines.

approach, and propose to generate graphs with denoising diffusion models. Unlike other popular generative models such as VAEs, GANs, or normalizing flows, denoising diffusion models do not require the specification of a latent space. Instead, they view generation as a sequence of denoising tasks. In particular, an important advantage of this framework is that it does not require to compress data into a lower-dimensional representation. The denoising network is trained to perform node and edge-level predictions, which are tasks at which graph networks excel. Moreover, the iterative denoising scheme employed by diffusion models allows them to correct their own predictions, which results in the production of high-quality outputs.

We introduce two denoising diffusion models for graphs: DiGress generates graphs with categorical attributes and the nodes and edges, while MiDi extends the generation to graphs that also feature continuous 3D coordinates. Although they operate on different data modalities, these models share strong similarities.

DiGress and MiDi both use discrete diffusion, which means that corrupting the data or performing one denoising step amounts to sampling each node and edge type from a categorical distribution. We show that discrete diffusion significantly outperforms Gaussian based models. This is due to the fact that Gaussian-based diffusion models for graphs try to predict continuous values that do not exist in the data, and destroy the graph's sparsity. Instead, discrete diffusion can adapt the noise model to preserve the graph's sparsity during diffusion.

We then present the MiDi model which learns to generate molecular graphs jointly with their corresponding 3D structures. Unlike existing methods that rely on predefined rules to determine molecular bonds based on the 3D conformation, MiDi offers an end-to-end differentiable approach that streamlines the molecule generation process. Learning jointly to generate the 2D and 3D structure of a molecule allows to generate molecular bonds that are consistent with the 3D structure, while also alleviating the limited expressivity of MPNNs.

Both DiGress and MiDi provide significant performance gains over previous work on their

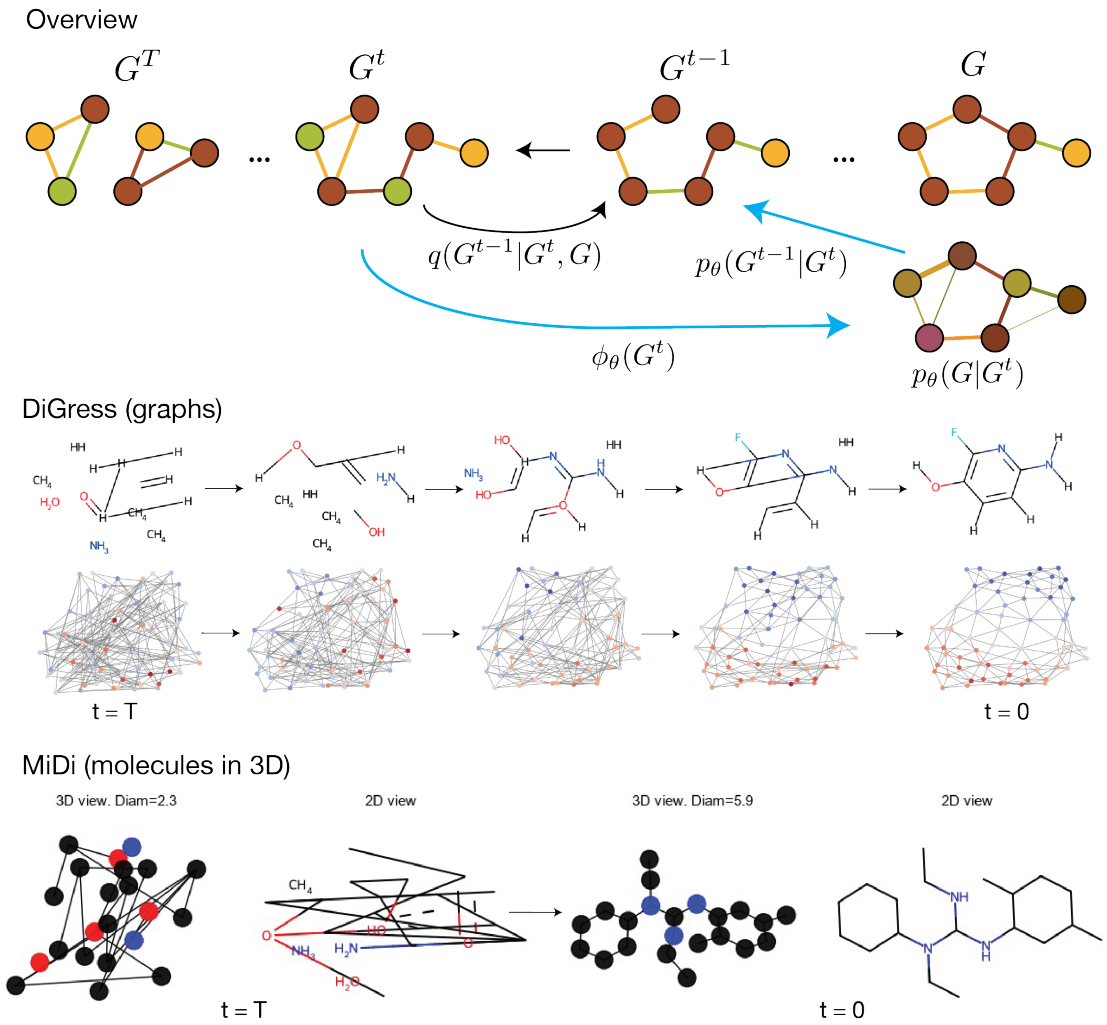


Figure 1.3 – Denoising diffusion models define a Markov noise model q progressively applied to data, and a denoising network ϕ_θ trained to predict clean data from a noisy input. They can be used to produce new samples by iteratively sampling $G^{t-1} \sim \int_G q(G^{t-1}|G^t, G) dp_\theta(G)$. DiGress defines a discrete diffusion process for graphs with categorical node and edge attributes. MiDi is a SE(3) equivariant model for molecule generation in 3D. While previous models would produce a 2D graph from a 3D conformer, MiDi learns to generate both modalities together, which results in more realistic compounds.

respective data modalities. These successes can be to a large part attributed to the structure of denoising diffusion models, that are particularly suited to permutation equivariant learning.

List of contributions

This thesis is based on the following papers:

- *Building powerful and equivariant graph neural networks with structural message-passing, Clément Vignac, Andreas Loukas, Pascal Frossard – NeurIPS 2020*
- *Top-n: Equivariant Set and Graph Generation without Exchangeability, Clément Vignac, Pascal Frossard – ICLR 2022*
- *Equivariant Diffusion for Molecule Generation in 3D, Emiel Hoogeboom*, Victor Garcia Satorras*, Clement Vignac*, Max Welling – ICML 2022*
- *DiGress: Discrete Denoising diffusion for graph generation, Clement Vignac*, Igor Krawczuk*, Antoine Siraudin, Bohan Wang, Volkan Cevher, Pascal Frossard – ICLR 2023*
- *MiDi: Mixed Graph and 3D Denoising Diffusion for Molecule Generation, Clement Vignac*, Nagham Osman*, Laura Toni, Pascal Frossard – under review*

2 Background

In this chapter, we introduce graph neural networks and discuss their expressive power. While graph neural networks were historically introduced as an extension of convolutional networks to graphs, we rather describe them as *local permutation equivariant networks*. This perspective uses group equivariance as a tool to design novel architectures that respect a problem symmetries (Bronstein et al., 2021). We will show that this perspective better reflects their connection with equivariant networks for sets such as Transformers (Vaswani et al., 2017), and permits to understand the specificity of graph data.

2.1 Equivariance

2.1.1 Group representations

Equivariant machine learning is a broad and rapidly evolving field that extends beyond graph neural networks (Kondor, 2008; Cohen et al., 2021). Its primary goal is to develop algorithms that can effectively model data with symmetry or invariance properties. While some symmetries, such as scale invariance (Worrall & Welling, 2019) or stability to diffeomorphisms (Bruna & Mallat, 2013), can be considered, most works in this field focus on symmetries that have the structure of a group $\mathbb{G}$. These symmetries have a profound impact on the structure and behavior of the data and are therefore crucial to consider in any analysis or generation task.

Groups

A group $\mathbb{G}$ is a set endowed with a composition operation denoted $\circ$ (usually omitted in practice) that satisfies 3 requirements:

- Associativity: $\forall g, g', g'' \in \mathbb{G},\ (g \circ g') \circ g'' = g \circ (g' \circ g'')$

- Existence of an identity element: $\exists e \in \mathbb{G}, \forall g \in \mathbb{G}, e \circ g = g \circ e = g$

- Existence of an inverse: $\forall g \in \mathbb{G}, \exists g^{-1} \in \mathbb{G}, g \circ g^{-1} = g^{-1} \circ g = e$, where e is the identity element.

In this work, the groups that we will consider are mostly the symmetric group $\mathbb{S}_n$ containing all permutations of n elements, and the special Euclidean group SE(3) generated by translations and rotations of the 3D space.

Group actions

In this thesis, we use groups to study symmetries, i.e., properties of a space that are preserved when a transformation is applied. These transformations are mathematically defined by group actions. Specifically, given a group $\mathbb{G}$ and a set $\mathbb{X}$, a group action of $\mathbb{G}$ on $\mathbb{X}$ is a function (denoted by a dot) from $\mathbb{G} \times \mathbb{X}$ to $\mathbb{X}$ such that the following conditions hold: (1) the identity element of $\mathbb{G}$ fixes every element of $\mathbb{X}$, (2) the action is associative, and (3) $\forall g, h \in \mathbb{G},\ \forall x \in \mathbb{X}$, we have $(gh).x = g.(h.x)$.

An example of group action is the action of the rotation group SO(2) on the plane: each rotation $r_\theta \in SO(2)$ defines a bijective transformation of the plane that maps $(x, y) \in \mathbb{R}^2$ to $(\cos\theta\ x + \sin\theta\ y, \sin\theta\ x - \cos\theta\ y) \in \mathbb{R}^2$. We can also consider actions of SO(2) on the 3D space: each action will be parameterized by an axis of rotation.

Group representations

In practice, group actions are not a sufficient notion for machine learning applications: usually, we do not consider the way a symmetry transforms a space, but the way it transform a *signal defined on that space* (Kondor, 2008). Among the possible ways to transform a signal, the class of transformations that can be described by matrix multiplications are prone to a rich mathematical analysis. Such transformations are called *linear group representations* (Serre, 1977).

A linear representation ρ of a group $\mathbb{G}$ on a vector space $\mathbb{X}$ of dimension d is a group homomorphism from $\mathbb{G}$ to the invertible $d \times d$ matrices $GL(d)$. This means that for any group elements $g_1, g_2 \in \mathbb{G}$, their associated matrices $\rho(g_1)$ and $\rho(g_2)$ satisfy the group multiplication law: $\rho(g_1 g_2) = \rho(g_1)\rho(g_2)$. In other words, the linear transformation induced by the group action on $\mathbb{X}$ is described by a matrix that respects the group structure. Certain representations are particularly natural: for example, when we refer to a rotation matrix in 2D, we do not directly refer to an element of SO(2), but rather to a representation of SO(2) on the plane. To simplify the notation, we will use the shorthand $g.x$ to denote the image of x under the action of $g \in \mathbb{G}$, rather than the more cumbersome notation $\rho(g)(x) \in \mathbb{X}$. This notation emphasizes the intuitive interpretation of linear group representations in most practical cases.

Linear group representations have found a wide range of applications in physics, and particularly in quantum mechanics, where symmetries play a fundamental role. By associating matrices with

group elements, linear representations permit to study the behavior of physical systems under the action of symmetries, and to construct invariant quantities that capture the essential properties of these systems. The key theorem of linear representation theory states that the decomposition of any group action into a direct sum of irreducible representations, or irreps, which are the most basic building blocks of representations that cannot be further decomposed. The manipulation of irreps has led to the development of many machine learning methods, particularly for the special Euclidean group SE(3) (Cohen et al., 2021; Geiger & Smidt, 2022).

In this thesis, we will only consider intuitive representations, and will not manipulate irreps theory. We describe below the representations that we will use for the permutation group and the special Euclidean group.

Representations of the permutation group

As this thesis considers unordered objects, we are primarily interested in the representations of the permutation group. Let $\mathbf{T}$ be a tensor of size $n^k \times d$. We refer to k as the *order* of the tensor: a vector is an order 0 tensor for the permutation group, a set as in order 1 tensor, and a graph is an order 2 tensor. Hyper-graphs can also be considered, but their cost make them impractical for many applications. In this thesis, the only representation of the permutation group that we will consider permutes the indices on the dimensions that index nodes. For any tensor $\mathbf{T} \in \mathbb{R}^{n^k \times d}$, it acts as:

$$\forall \pi \in \mathbb{S}_n,\ \pi.\mathbf{T}[i_1, ..., i_k, j] = \mathbf{T}[\pi^{-1}(i_1), ..., \pi^{-1}(i_k), j]$$

In particular, if $\boldsymbol{\pi} \in \mathbb{R}^{n \times n}$ is the permutation matrix corresponding to π, we get $\pi.\boldsymbol{z} = \boldsymbol{z} \in \mathbb{R}^d$ for vectors, $\pi.\boldsymbol{X} = \boldsymbol{\pi}^T \boldsymbol{X}$ for multisets, and $\pi.\boldsymbol{A} = \boldsymbol{\pi}^T \boldsymbol{A} \boldsymbol{\pi}$ for adjacency matrices.

Representations of SO(3)

In chapter 5, we will also consider 3D point clouds with a rotational symmetry, which requires to also study representations of the rotation group SO(3). We will only consider two simple representations. The first one is the trivial representation, which is used for rotation invariant quantities such as the atom type $\boldsymbol{a}$ of a molecule. In this case, for any rotation r, we simply have $r.\boldsymbol{a} = \boldsymbol{a}$. The second representation corresponds to quantities, such as a momentum vector $\boldsymbol{p}$, that rotate with the molecule. In this case, we have $r.\boldsymbol{p} = \boldsymbol{R}\boldsymbol{p}$, where $\boldsymbol{R}$ is the rotation matrix associated with r.

The possible representations of SO(3) are much more diverse than these two basic examples. The irreducible representations (irreps) of SO(3) correspond to the spherical harmonics, which have important applications in quantum physics. Several neural networks have been proposed that manipulate these spherical harmonics (Thomas et al., 2018; Liao et al., 2019; Brandstetter et al., 2021; Liao & Smidt, 2022).

2.1.2 Equivariance

Definition We can now move to the definition of equivariance. Consider a function $f : \mathbb{X} \to \mathbb{Y}$, and two representations $\rho(g)$ and $\rho'(g)$ of a group $\mathbb{G}$ on respectively $\mathbb{X}$ and $\mathbb{Y}$. We say that f is equivariant to the action of $\mathbb{G}$ if:

$$\forall g \in \mathbb{G}, \forall x \in \mathbb{X},\ f(\rho(g)x) = \rho'(g)f(x) \tag{2.1}$$

Here again, since we will in practice manipulate intuitive representations, we simplify the notations and write $f(g.x) = g.f(x)$. In this formula, the two representations, both denoted by "g.", might not be the same.

In the particular case where the group representation on $\mathbb{Y}$ is trivial, we say that f is invariant to the action of $\mathbb{G}$:

$$\forall g \in \mathbb{G}, \forall x \in \mathbb{X},\ f(g.x) = f(x) \tag{2.2}$$

Group averaging When the symmetry group $\mathbb{G}$ is finite and small enough, equivariance or invariance can be achieved by averaging any function f over $\mathbb{G}$. Group averaging is described by the two following functions:

$$f^{\text{inv}}(x) = \sum_{g \in \mathbb{G}} f(g.x) \qquad \text{and} \qquad f^{\text{eq}}(x) = \sum_{g \in \mathbb{G}} g^{-1}.f(g.x) \tag{2.3}$$

It is elementary to check that these functions are respectively invariant and equivariant. Unfortunately, the groups considered in this thesis are either infinite (e.g. SE(3)) or too large ($\mathbb{S}_n$) to compute an average over the group elements. Although networks that are approximately equivariant can be designed by summing over random permutations (Murphy et al., 2019), this strategy is not computationally efficient.

2.1.3 Equivariance through the choice of the data representation

While most of this thesis is devoted to the design of equivariant neural network, it needs to be noted that adapting the neural network parametrization is not the only way to respect the symmetry of a problem. When possible, it can be more efficient to directly represent the data in a coordinate system that is invariant to the symmetry. For example:

- When predicting the future state of a n-body system, it can be useful to model the data as a graph with edge features representing the distance between points. Modeling a physical system as a distance graph incorporates many symmetries such as the invariance to translations, rotations, reflections and permutations.
- Invariance to permutations in graphs can be obtained by learning only functions of the

graph eigenvalues. For equivariance, we can also consider the eigenvectors (u_i) associated to eigenvalues of multiplicity one, although the sign ambiguity needs to be resolved by considering both u_i and $-u_i$ (Lim et al., 2022).

- Since proteins are essentially chain graphs of alpha carbons (C_α), many symmetries of proteins can be leveraged by choosing the data representation appropriately. For example, the 3D arrangement of the proteins can modeled using the angles between successive carbons. This representation naturally encodes invariance to translations and rotations, but also reduces the number of parameters from $3n$ to n, if n is the number of C_α atoms in the chain. This representation plays a key role in the success of both AlphaFold (Jumper et al., 2021) and diffusion models for proteins, which can scale to much larger structures than arbitrary graphs or point clouds (Ingraham et al., 2022).

At first sight, it might not be clear how to derive general principles to guide the choice of the data representation. Frame averaging (Puny et al., 2021) constitutes a first step towards this direction. Given a point $x \in \mathbb{X}$, a frame corresponds to a subset $F(x) \in \mathbb{G}$ such that equivariance is achieved when averaging over $F(x)$. For a simple example of frame averaging, consider translation invariant functions of a point cloud $\boldsymbol{C} \in \mathbb{R}^{n \times d}$. Such functions can be obtained by defining $f^{\text{inv}}(\boldsymbol{X}) = f(\boldsymbol{X} - \mathbf{1}_n(\frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i)^T)$, which corresponds to an averaging over a frame $f(\boldsymbol{X}) = \{\frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i\}$ that contains a single element. Frame averaging can both be seen as an extension of group averaging, and as a change of representation of the data.

Despite these advances, it is sometimes often clear how to represent the data in a way that naturally incorporates all symmetries. In such cases, the neural networks need to be constrained in order to only represent the relevant class of functions. In the following section, we describe methods to construct neural networks that possess equivariance with respect to a particular group.

2.2 Equivariant neural networks for sets

In order to build a rich family on nonlinear and differentiable functions, a common approach is to combine linear equivariant functions with equivariant non-linearities. Other operators that preserve equivariance can also be considered. For permutation equivariance, these operators include multiplication, addition, composition, and tensor products. By constructing a computational tree using these building blocks, we can create rich classes of equivariant function. It is worth noting that the class of invariant linear functions is often much more restricted than the class of equivariant functions. Thus, when the task requires invariance to a group action (e.g., image classification, where the identity of an animal does not change upon rotation), a typical strategy is to employ equivariant layers and then to apply an invariant operator at the end of the network.

2.2.1 Equivariant linear layers between arbitrary dimensions

To construct equivariant neural networks, the computation of linear equivariant layers is a critical component. When the group $\mathbb{G}$ is compact, it can be demonstrated that the linear layers that are equivariant to the action of $\mathbb{G}$ are convolutions on the group $\mathbb{G}$. When considering the group of translations in $\mathbb{R}^d$, these convolutions correspond to the standard definition of convolutions on Euclidean spaces, such that $f(x)[u] = \int_{\mathbb{R}^d} x(u-\tau)k(\tau)d\tau$, where k is the convolutional kernel. Translations are however peculiar, as the symmetry group can be identified with the base space $\mathbb{R}^d$. For other groups, the integral is defined over $\mathbb{G}$ and not over $\mathbb{R}^d$ (Kondor & Trivedi, 2018).

Fortunately, when dealing with equivariance to the symmetric group $\mathbb{S} = \bigcup_{n\in\mathbb{N}^*} \mathbb{S}_n$, the computation of equivariant linear layers does not require integrating over the group. Since for each n, $\mathbb{S}_n$ is a finite group, the equation $\forall \pi \in \mathbb{S}_n, \pi.f(\mathbf{T}_1) = f(\pi.f(\mathbf{T}_2))$ forms a linear system with a finite number of equations. By solving these equations, Maron et al. (2018) showed that the space of functions from $\mathbb{R}^{n^k\times d}$ to $\mathbb{R}^{n^{k'}\times d'}$ is of dimension $dd'Bell(k+k')$, where $Bell(k+k')$ is the Bell number of order $k+k'$, i.e., the number of partitions of a set of $k+k'$ elements. Notably, this number is independent of the number of nodes n and only depends on the tensor order.

Different values of k, k' correspond to different types of functions: $k=1$ and $k'=0$ correspond to functions that map sets to vectors, $k=1$ and $k'=2$ corresponds to set to graph functions, etc. In general, we write $l_{k\to k'}$ the linear permutation functions from $\mathbb{R}^{n^k\times d}$ to $\mathbb{R}^{n^{k'}\times d'}$.

2.2.2 Neural networks for sets

Before analyzing graph neural networks, we first consider neural networks for sets, i.e., layers with $k=1$. We therefore consider a set $\mathbb{V}$ where each point v_i has an attribute vector $\boldsymbol{x}_i$. For small values of k', we obtain the following functions:

Global pooling of sets The space of linear set to vector functions is spawned by a single element:

$$l_{1\to 0}(\boldsymbol{X}) = \sum_{v_i\in\mathbb{V}} \boldsymbol{x}_i\boldsymbol{W} \tag{2.4}$$

In practice, a popular alternative to linear set to vector layers is the use of several pooling operators, as done in the PNA layer (Corso et al., 2020):

$$\mathrm{PNA}(\boldsymbol{X}) = \mathrm{cat}(\mathrm{mean}(\boldsymbol{X}), \max(\boldsymbol{X}), \min(\boldsymbol{X}), \mathrm{std}(\boldsymbol{X})) \tag{2.5}$$

Set to Set: Deep sets The space of linear functions mapping sets to sets is spawned by two elements:

$$l_{1\to 1}(\boldsymbol{X}) = \boldsymbol{I}\boldsymbol{X}\boldsymbol{W}_1 + \mathbf{1}_n\mathbf{1}_n^T\boldsymbol{X}\boldsymbol{W}_2 \tag{2.6}$$

This class of functions, called Deep Sets (Zaheer et al., 2017), can equivalently be written:

$$\forall v_i \in \mathbb{V},\ l_{1\to 1}(\boldsymbol{X})_i = \boldsymbol{W}_1^T \boldsymbol{x}_i + \sum_{v_j \in \mathbb{V}} \boldsymbol{W}_2^T \boldsymbol{x}_j \tag{2.7}$$

This equivalence may seem trivial, but it provides valuable insights into permutation-equivariant networks. Specifically, in a permutation equivariant network, each point can distinguish itself from the other nodes and learn a separate coefficient for its own representation, but it cannot distinguish between the other points. This insight can be used to guess the equivariant functions for other interaction orders. For example, in linear graph-to-vector functions, since each node v_{i_0} can distinguish the pair $i_0 - i_0$ from other pairs $i_0 - j$, we can predict a basis for linear equivariant graph-to-vector functions is spawned by the trace of $\boldsymbol{A}$ and the sum of non diagonal entries.

The approach outlined above can be extended to construct neural networks that operate on tensors of any order, including graph neural networks (Maron et al., 2019a). However, these networks lack a principled inductive bias that can leverage the locality and sparsity of adjacency matrices. As a result, these methods tend to be hard to train and do not generalize very well (Vignac et al., 2020).

Interaction networks for sets Another line of works proposed an alternative to the use of linear layers. Since the main assumption of graph neural networks is that the data is composed of interacting entities, these methods propose to explicitely model the interaction between components (Battaglia et al., 2016; Lee et al., 2019). That is, instead of Eq. (2.7), these networks compute

$$\forall v_i \in \mathbb{V},\ \mathrm{InteractionNet}(\boldsymbol{X})_i = h(\boldsymbol{x}_i, \sum_{j \neq i} g(\boldsymbol{x}_i, \boldsymbol{x}_j)), \tag{2.8}$$

where g and h are two arbitrary functions, such as multi-layer perceptrons (MLP). This function class can be slightly extended to:

$$\mathrm{InteractionNet}(\boldsymbol{X})_i = h(\boldsymbol{x}_i, \sum_{j \neq i} g(\boldsymbol{x}_i, \boldsymbol{x}_j, \{\boldsymbol{x}_k\}_{k \neq i,j}) \tag{2.9}$$

Here, the interaction function also depends on the other node, for example through a normalization factor. A special parametrization of this architecture is:

$$\forall v_i \in \mathbb{V},\ f(\boldsymbol{X})_i = \sum_{j \neq i} \mathrm{softmax}(\underbrace{\boldsymbol{W}_q^T \boldsymbol{x}_i}_{\text{query}} \underbrace{\boldsymbol{W}_k^T \boldsymbol{x}_j}_{key}, \{\boldsymbol{W}_k^T \boldsymbol{x}_k \boldsymbol{W}_q^T \boldsymbol{x}_i\}_{k \neq j}) \underbrace{\boldsymbol{W}_v^T \boldsymbol{x}_i}_{\text{value}}, \tag{2.10}$$

where $\boldsymbol{W}_q, \boldsymbol{W}_k, \boldsymbol{W}_v$ are three matrices of learnable parameters. In this formula, we recognize the the self-attention layer of the Transformer architecture (Vaswani et al., 2017) which has become widely popular in many areas of machine learning. Interestingly, Transformers can therefore be seen as equivariant neural network for sets (Joshi, 2020). Given the success of

Transformers across many data modalities, a natural question arises as to what makes this parametrization superior to other equivariant networks for sets such as the interaction networks $h(\boldsymbol{x}_i, \sum_{j \neq i} g(\boldsymbol{x}_i, \boldsymbol{x}_j))$ of Eq. (2.8). Although this question is not completely resolved yet, two key factors can be identified. The first one is the fact that in Transformer, the variance of the activations is controlled through the use of residual connections and normalization layers. The second one is the relative memory efficiency of the Transformer model. Despite a $O(n^2)$ memory cost due to the computation of interaction terms, the only quadratic tensor manipulated in the Transformer is of size $n \times n \times n_{\text{heads}}$, with a number of attention heads that is typically very small. This is in contrast to other interaction networks, in which each $g(\boldsymbol{x}_i, \boldsymbol{x}_j)$ can be a much larger vector.

Overall, the introduction of equivariant neural networks for sets permits to show that there is no fundamental distinction in the parametrization of set equivariant networks and graph neural networks. Even for sets, it is often relevant to introduce interaction terms in the learned function, and therefore to consider pairs of nodes. In graph neural networks however, the data contains edge features that need to additionally be taken into account.

2.3 Graph neural networks

2.3.1 Graph representations

We now consider the case of data represented as graphs. In this context, in addition to the node features, we have access to an adjacency matrix $\boldsymbol{A}$ and potentially edge features $\boldsymbol{y}_{ij} \in \mathbb{R}^{d_y}$ on each edge. These edge features are grouped in a tensor $\mathbf{Y} \in \mathbb{R}^{n \times n \times d_y}$.

Graphs can be represented using either adjacency matrices or adjacency lists. While these two representations are theoretically equivalent, the computational implications of each representation are distinct. Adjacency matrices are dense tensors that require to store n^2 entries. Since the size of the representation does not depend on the sparsity level, it is therefore often beneficial to consider fully connected graphs when working with adjacency matrices. In this case, $\boldsymbol{A}$ can be seen as an edge feature on a fully connected graph. Conversely, adjacency lists result in the manipulation of sparse tensors. The main drawback of sparse operations is that they are challenging to parallelize on GPUs due to their irregular memory access patterns. However, graph libraries such as PyTorch Geometric (Fey & Lenssen, 2019) and Deep Graph Library (Wang, 2019) are now able to implement message-passing operations on GPUs using only adjacency lists. These libraries permit to process graphs with up to $\sim 10^5$ nodes provided that the graphs are sparse enough, and even bigger graphs with the use of stochastic node samplers. The importance of these libraries in the rise of graph neural networks cannot be overstated.

However, it should be noted that graph libraries introduce an overhead compared to dense tensor operations. With the increasing availability of GPUs with large memory capacity, the trend has shifted back towards the use of dense representations for processing small graphs. Ultimately,

the choice of graph representation depends on the size of the graphs, their sparsity level, and the available memory for computations.

2.3.2 Parametrization

The interaction networks presented in Eq. (2.8)) can naturally be extended to incorporate edge features, as they compute a function for all pairs of points. Specifically, the following "Edge-Interaction network" generalizes the interaction network to take into account edge features:

$$\forall v_i \in \mathbb{V},\ \text{Edge-InteractionNet}(\boldsymbol{X}, \mathbf{Y})_i = h(\boldsymbol{x}_i, \sum_{j \neq i} g(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{y}_{ij})) \tag{2.11}$$

The main drawback of this parametrization is its $O(n^2)$ complexity, which makes it impractical for large graphs. Nevertheless, it offers a very efficient way to process small graphs, and has been used successfully on datasets of small molecules, with the edges encoding pairwise distances between atoms as well as bond types (Gilmer et al., 2017). When dealing with larger graphs, the quadratic complexity becomes a strong limitation, making it necessary to restrict the computation to nodes that are most likely to affect each other, i.e., neighboring nodes. This gives rise to Message-Passing Neural Networks (MPNNs) (Scarselli et al., 2008; Gilmer et al., 2017), which only model explicitly interactions between each node v_i and its neighbors $v_j \in N(v_i)$. By restricting interaction networks to the explicit modeling of interaction between neighbors only, we obtain a MPNN based on the sum aggregation:

$$\forall v_i \in \mathbb{V},\ \text{sum-MPNN}(\boldsymbol{X}, \mathbf{Y})_i = h(\boldsymbol{x}_i, \sum_{j \in N(i)} g(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{y}_{ij})) \tag{2.12}$$

The term message-passing arises from an interpretation of these algorithms as an iterative propagation scheme, similar to those used in distributed algorithms or belief propagation. The function g is viewed as a *message* passed on the edge $v_j \to v_i$, while the function h corresponds to the *update* rule of node v_i. Finally, the sum can be replaced by any permutation-invariant *aggregation* function $\square$. Message-passing neural networks are typically expressed as:

$$\forall v_i \in \mathbb{V}, \quad \text{MPNN}(\boldsymbol{X}, \mathbf{Y})_i = h(\boldsymbol{x}_i, \square_{j \in N(i)} g(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{y}_{ij})) \tag{2.13}$$

An important advantage of message-passing neural networks is their computational efficiency, with a complexity linear in the number of edges. This makes them well-suited for processing large graphs. Additionally, MPNNs are designed to compute and propagate local information, similar to how convolutional networks operate on images. As a result of these desirable properties, MPNNs serve as the most studied function class in graph machine learning.

Parametrization of the message and update functions The choice of parametrization is highly empirical, and architectures can vary significantly across applications. For graphs without

edge attributes, the message function can be parameterized as $g(\boldsymbol{x}_j)$, in which case only n unique messages need to be computed, or as $g(\boldsymbol{x}_i, \boldsymbol{x}_j)$, which requires computing m different messages. While the latter is more computationally intensive, it allows for the explicit modeling of interactions, which is beneficial in many settings. A example of a message function could for example be:

$$g(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{W}_1^T \operatorname{ReLU}(\boldsymbol{W}_2^T \operatorname{cat}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_i \odot \boldsymbol{x}_j, \boldsymbol{e}_{ij})), \tag{2.14}$$

where $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are learnable matrices. This function can be implemented more efficiently by leveraging the equivariance between addition and concatenation before a linear layer (Perez et al., 2018):

$$m(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{W}_1^T \operatorname{ReLU}(\boldsymbol{W}_3^T \boldsymbol{x}_i + \boldsymbol{W}_4^T \boldsymbol{x}_j + \boldsymbol{W}_5^T \boldsymbol{x}_i \odot \boldsymbol{x}_j + \boldsymbol{W}_6^T \boldsymbol{e}_{ij}), \tag{2.15}$$

where $\boldsymbol{W}_2$ is the horizontal concatenation of the matrices $\boldsymbol{W}_3, \ldots, \boldsymbol{W}_6$. The update function is often a simple residual connection $h(\boldsymbol{x}_i, \tilde{\boldsymbol{x}}_i) = \boldsymbol{x}_i + \tilde{\boldsymbol{x}}_i$, or a MLP applied to the concatenation of the input $\boldsymbol{x}_i$ and the aggregated messages $\tilde{\boldsymbol{x}}_i$.

Parametrization of the aggregation function The simplest aggregation function in MPNNs is the sum, but it has the drawback of being unstable when the number of nodes varies across graphs. On the contrary, mean aggregation stabilizes the activations but is unfortunately less expressive, as the neural network loses information about the number of nodes in the graph (Xu et al., 2019). To get the best of both worlds, it is possible to divide the sum by the average degree. The max aggregation function is mostly used for algorithmic reasoning tasks (Cappart et al., 2021). When it is not clear which aggregation function will work best, it is possible to let the network learn it by concatenating the output of several ones (Corso et al., 2020).

2.3.3 Standard architectures

MPNNs encompass a wide range of architectures. We describe some of the most popular ones below:

Graph Convolutional network (GCN)

The Graph Convolutional network (GCN) layer has been proposed in (Kipf & Welling, 2016) as a simplification of the ChebNet architecture for graph classification (Defferrard et al., 2016):

$$\operatorname{GCN}(\boldsymbol{X}, \boldsymbol{A})_i = \sigma(\frac{1}{d_i} \boldsymbol{W}^T \boldsymbol{x}_i + \sum_{v_j \in N(i)} \frac{1}{\sqrt{(d_i+1)(d_j+1)}} \boldsymbol{W}^T \boldsymbol{x}_j), \tag{2.16}$$

where σ is a ReLU non-linearity and d_i the degree of node v_i. While previous architectures were mostly designed as a equivalent of CNNs for graphs, with a succession of convolutional and pooling layers designed for graph classification tasks, an interesting contribution of (Kipf &

Welling, 2016) is the use of graph neural networks for node-level tasks such as semi-supervised node classification. Architectures for node classification are conceptually simpler than architectures for graph classification, as they do not require the use of pooling layers. Furthermore, node-classification architectures can be trained on one single graph, which makes it easier to build novel datasets for node-level tasks than for graph-level tasks. As a result, this very simple architecture has become the most popular one, despite several known limitations discussed later such as over-smoothing and a bias towards homophily.

Graph Isomorphism Network (GIN)

The Graph Isomorphism Network (GIN) proposed in (Xu et al., 2019) aims at being the simplest architecture that is maximally powerful in the class of MPNNs. GIN operates on graphs without edge attributes. It computes the following update:

$$\mathrm{GIN}(\boldsymbol{X}, \boldsymbol{A})_i = \mathrm{MLP}\left((1+\epsilon)\cdot \boldsymbol{x}_i + \sum_{j\in\mathcal{N}(i)} \boldsymbol{x}_j\right), \tag{2.17}$$

where ϵ is a small constant, and MLP a multi-layer perceptron applied in parallel on each node. GIN is a particularly cheap architecture, as it does not put any learnable parameters in the message function. However, it is usually not the best performing architecture (Dwivedi et al., 2023), which can be attributed to the fact that it does not compute explicitly any interaction term.

Graph Attention Network and Graph Transformers

Graph attention networks (GAT) (Veličković et al., 2017) aim at adapting the popular self-attention networks (Bahdanau et al., 2014) to graphs. They propose a model in which the node features are used to compute attention weights, i.e., to favor interactions between some nodes at the expense of others. The update rule of GAT can be written as:

$$\mathrm{GAT}(\boldsymbol{X}, \boldsymbol{A}, \mathbf{E})_i = \sum_{j\in v_i\cup N(i)} \alpha_{ij}\, \boldsymbol{W}_v \boldsymbol{x}_j, \tag{2.18}$$

where the attention coefficients $\alpha_{i,j}$ are computed using a softmax over a node and its neighbors:

$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{k\in v_i\cup N(i)} \exp(e_{i,k})} \tag{2.19}$$

and $e_{i,j}$ is the attention energy between nodes v_i and v_j. The energy function can be defined as a concatenation of the hidden states of nodes v_i and v_j, followed by a linear transformation:

$$e_{i,j} = \mathrm{LeakyReLU}\Big(\boldsymbol{a}^T[\boldsymbol{W}_q\boldsymbol{x}_i || \boldsymbol{W}_k\boldsymbol{x}_j]\Big) \tag{2.20}$$

where $\boldsymbol{a}$ is a learnable attention parameter vector, LeakyReLU is the leaky rectified linear unit activation function, $||$ denotes concatenation, $\boldsymbol{W}_q$ and $\boldsymbol{W}_k$ are respectively called the *key* and *query* matrices. (Brody et al., 2021) later found that the LeakyReLU should be placed before the learnable attention vector for greater expressivity. In this case, the attention energy writes:

$$e_{i,j} = \boldsymbol{a}^T \text{LeakyReLU}\Big([\boldsymbol{W}_q \boldsymbol{x}_i || \boldsymbol{W}_k \boldsymbol{x}_j]\Big) \tag{2.21}$$

On standard benchmarks, vanilla GATs tend to perform on par with simpler graph networks such as GCN (Shchur et al., 2018). However, the self-attention mechanism can be integrated into a graph Transformer architecture through the use of normalization layers and positional encoding. Graph Transformers have recently attracted a lot of attention and have shown impressive performance, outperforming vanilla GAT networks in many cases. For comprehensive surveys of recent architectures, we refer to (Chen et al., 2022), (Min et al., 2022), and (Müller et al., 2023).

2.4 Known Limitations of permutation equivariant networks

2.4.1 Expressive power

Despite their potential, graph neural networks have limitations in their expressive power, particularly in their ability to approximate all functions over graphs. In this section, we explore some known results about the expressive power of equivariant neural networks, beginning with those for sets. We will see that, while neural networks for sets and graph neural networks differ little in their parametrization, there are important differences in their theoretical properties.

Equivariant Networks for Sets

The design of universal approximators of equivariant functions on sets is straightforward. For example, Deep Set layers, which were introduced earlier in Eq. (2.7) and are parameterized by $f(\boldsymbol{X}) = \boldsymbol{X}\boldsymbol{W}_0 + \boldsymbol{1}\boldsymbol{1}^T\boldsymbol{X}\boldsymbol{W}_1$, can be used in conjunction with pointwise non-linearities to build a universal approximator of invariant and equivariant functions on sets (Zaheer et al., 2017; Segol & Lipman, 2019). An even simpler universal approximator of invariant functions on sets is provided by the PointNet architecture (Qi et al., 2017), which computes:

$$\text{PointNet}(\boldsymbol{X}) = \text{MLP}(\sum_{i=1}^{n} \text{MLP}(\boldsymbol{x}_i)) \tag{2.22}$$

Note however that these universal approximation results are limited to sets of a fixed maximal size, and the dimensionality of the feature maps considered needs to grow with n (Wagstaff et al., 2019).

Another way to obtain universal approximators of permutation equivariant functions for sets is

through canonization. Consider the following function:

$$f_{\text{canon}}(\boldsymbol{X}) = \text{unsort} \circ \text{group} \circ \text{MLP} \circ \text{flatten} \circ \text{sort}(\boldsymbol{X})$$

This function takes a set as input, sorts its rows by lexicographic order, flattens the resulting $n \times d$ matrix into a vector of size nd, learns an MLP on this vector, groups the output into an $n \times d$ matrix, and reverts the sort operation defined at the beginning. We can easily check that f_{canon} is permutation equivariant, and that it is a universal approximator of equivariant functions on sets. Note however that this function may not be practical as it learns an MLP over a very high-dimensional vector.

The expressive power of Message-Passing Neural Networks

We now turn our attention to the analysis of the expressive power of graph neural networks. First of all, one may wonder if the canonization strategy presented for sets can be used to build universal approximators of functions on graphs. Unfortunately, graph canonization is at least as hard as graph isomorphism testing, and no polynomial time algorithm is known for these tasks. This hints that universal approximators of functions on graphs will be difficult to obtain.

While a large literature has considered the expressive power of MPNNs, there is not one unique way to define expressive power. One interesting approach is for example to explore its ability to detect substructures, which has been explored in (Chen et al., 2020; Tahmasebi et al., 2020). Other approaches consider connections with distributed algorithms (Loukas, 2020b) and their algorithmic reasoning abilities (Xu et al., 2020a). Among the different perspectives on the expressivity of graph neural networks, the predominant viewpoint has however been to evaluate the network's capacity to differentiate between non-isomorphic graphs, which is the perspective that we adopt here. This perspective considers their relationship to the Weisfeiler-Lehman isomorphism test, that we know describe.

The Weisfeiler-Lehman test The Weisfeiler-Lehman (WL) algorithms (Weisfeiler, 2006; Cai et al., 1992) are a hierarchy of increasingly powerful algorithms for isomorphism testing. These algorithms propagate information on k-uplets of nodes, hash the embeddings obtained at every step, and compute the histograms of hashes obtained at each step. These histograms can then be compared in order to test graph isomorphism. The 1-WL test is the simplest instantiation of this algorithm, as it simply propagates information on the nodes. The algorithm iteratively computes

$$\boldsymbol{x}_i^{(k+1)} \leftarrow \text{hash}(\boldsymbol{x}_i^{(k)}, \sum_{j \in N(i)} \boldsymbol{x}_j^{(k)}), \tag{2.23}$$

and aggregates the histograms of the $\boldsymbol{x}_i^{(k)}$ at each step. The computations of the histograms can be seen as a global pooling operator that computes a permutation invariant descriptor from a multi-set of values. Two graphs are considered as isomorphic if the histograms of values are the

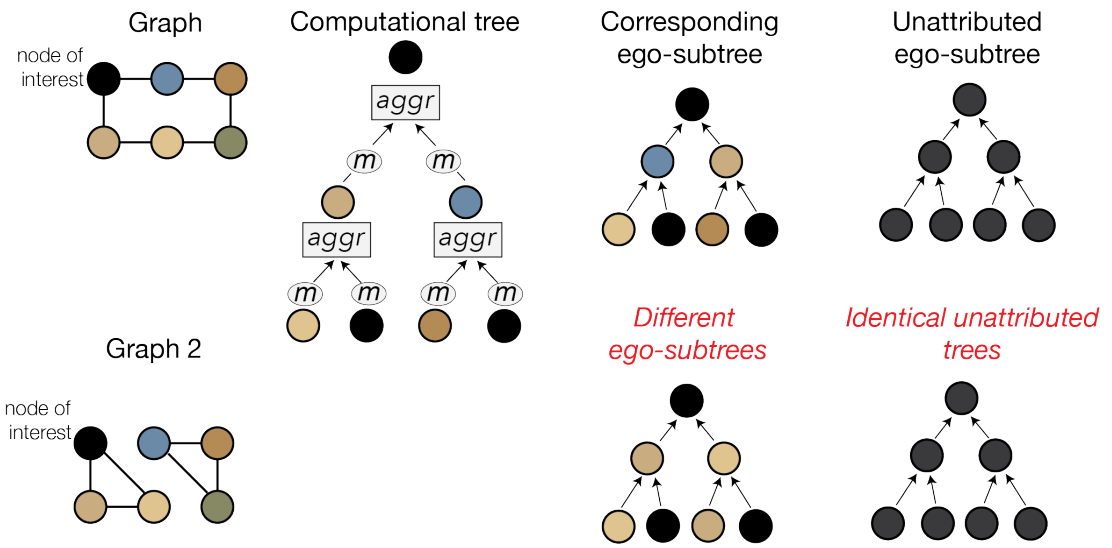


Figure 2.1 – Because of the iterative propagation scheme of message-passing neural networks, we can attach to each node an ego-subtree which corresponds to the computations made by MPNNs. These ego-subtrees informally represent the equivalence class of MPNNs: two nodes that have similar ego-subtrees will be mapped to the same value. When nodes do not have attributes or identifiers, nodes that do not have the same local topology can still have the same ego-subtree.

same for $k = 1$ to $k = K$, where K is a predefined number of iterations.

MPNNs are not more powerful than the 1-WL test The fundamental result, obtained simultaneously by Morris et al. (2019) and Xu et al. (2019), states that message-passing neural networks cannot surpass the Weisfeiler-Lehman test in terms at isomorphism testing. In the following, we provide more insight where this result comes from.

Consider the use of a MPNN of depth K. At each message-passing step, each node v_i updates its representation based on the representation of its neighbors $v_j \in N(v_i)$. The representations of each neighbor v_j depends, in turn, on the representation of the neighbors $v_k \in N(v_j)$ at previous step. Overall, the computations of the MPNN can be represented as a directed tree of depth K with root v_i, as shown in Figure 2.1. We refer to such a tree as the *ego-subtree* of depth K at node v_i.

Since 1-WL uses the same iterative propagation scheme as MPNNs, 1-WL also operates on rooted subtrees. Specifically, by comparing histograms, it compares the distribution of ego-subtrees of depth k at each step. It judges that two graphs are non isomorphic by identifying a depth k_0 at which the distribution of ego-subtrees differs between two graphs.

The results of (Xu et al., 2019) and (Morris et al., 2019) essentially shows that the rooted subtrees form the equivalence class of MPNNs. MPNNs cannot differentiate between nodes that have identical subtrees, but can potentially assign different values to all subtrees if they are

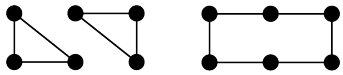


Figure 2.2 – These two simple graphs cannot be distinguished by the Weisfeiler-Lehman test. This reveals that message-passing neural networks are not able to detect very basic structural information about the graphs, such as connectivity or triangle counts.

parameterized in a maximally powerful way.

It is important to note that the ability to distinguish non-isomorphic graphs is highly dependent on the richness of the features. In graphs where each node can be uniquely identified by its features, the ego-subtrees become unique. This idea has motivated the introduction of node identifiers in message-passing networks (Loukas, 2020a), which we also use in Chapter 3. However, in unattributed graphs, MPNNs are limited in their ability to compute rich information on the nodes. At the first message passing step, MPNNs on an attributed graph can only learn a function of the node's degree. The second message-passing steps can only learn a function of distributions of the degrees, and so on. In regular graphs (where each node has the same degree) such as the one in Fig. 2.2, MPNNs are therefore not able to compute any useful information, as all nodes are assigned the same representation.

The limited expressive power of message-passing neural networks may appear to be a purely theoretical concern, but it has important practical implications on graph neural networks. The equivalence between the expressive power of MPNNs and the WL test highlights the inability of graph neural networks to learn fundamental graph structural information, such as the count of substructures (Chen et al., 2020; Tahmasebi et al., 2020). From Fig. 2.2 alone, the fact that MPNNs cannot distinguish between these two graphs shows that it is unable to detect if a graph is connected or if it possesses triangles, which is very basic information. As a result, graph neural networks tend in general to perform poorly on unattributed graphs (Cai & Wang, 2018).

The expressive power of higher-order graph neural networks

In order to overcome these strong limitations, many works have investigated more expressive classes of functions on graphs. Several of these architectures (Murphy et al., 2019; Sato et al., 2020) introduce stochasticity in the neural networks, which improves their theoretical power but makes them hard to train. Among the deterministic architectures, two main strategies have been proposed for building more expressive networks. The first approach involves defining linear layers between tensors of size $\mathbb{R}^{n^k \times d}$, which represent k-tuples of nodes. These linear layers can be concatenated and augmented with pointwise non-linearities (Maron et al., 2018). The second approach involves defining message-passing schemes between k-tuples of nodes, as done in (Morris et al., 2019; 2020).

Recent work by Keriven & Peyré (2019) showed that linear layers between high-order tensors are universal approximators of graph-to-set equivariant functions on graphs, which extends the

results of (Maron et al., 2019b) for invariant functions. However, the tensor order required for this approach may be prohibitively large, making it impractical.

In contrast, the expressive power of higher-order message-passing networks depends on the particular parameterization of the networks. Typically, networks that perform message-passing operations between k-tuples of nodes have an expressive power which is between the k-Weisfeiler-Lehman (k-WL) test (Morris et al., 2019) and the (k+1)-WL test (Maron et al., 2019a). However, Cai et al. (1992) constructed a graph that cannot be distinguished by the k-WL test, but that can be distinguished by the (k+1)-WL test, demonstrating that no k-WL powerful algorithm is a universal approximator of functions on graphs.

Overall, graph neural networks come with variable computational complexity and expressive power, but no deterministic architecture can approximate all functions on graphs. This should not surprise the reader: if we had access to a universal approximator of functions on graphs, it would be possible to use it in order to test the isomorphism of two graphs. Building universal approximators of functions on graphs is therefore at least as hard as isomorphism testing (Chen et al., 2019).

2.4.2 Other known limitations of graph neural networks

Isotropy. The equivalence between the expressive power of MPNNs and the Weisfeiler-Lehman test only holds when many MPNN layers are used. In practice, graph neural networks often use few layers, which results in even lower expressivity. Additionally, each graph network layer has undesirable local invariance properties due to the use of a symmetric (e.g., permutation invariant) aggregation operation (Kondor et al., 2018). Because of the use of symmetric aggregations, graph networks applied to grids compute filters with a spherical shape (shown in Figure 2.3), which is why they are commonly referred to as *isotropic* filters (Levie et al., 2018). This is in contrast to the kernels learned by CNNs for images, which are often directional (Krizhevsky et al., 2017). Isotropy is not necessarily a problem if it corresponds to a correct prior about the task. However, edge detectors cannot be obtained with spherical filters, and experiments have shown that graph networks applied to grids perform worse than CNNs with the same number of parameters (Vignac & Frossard, 2019). This suggests that isotropy could be a harmful inductive bias for other types of data as well.

The lack of an obvious notion of direction or orientation for arbitrary graphs makes it hard to design designing anisotropic or oriented filters. One approach to break the isotropy of message-passing layers involves using the first eigenvectors of the graph Laplacian to define general directions on the graph (Beaini et al., 2021). The use of the first eigenvectors can be theoretically justified when the graph is a cartesian product of small factors. In such cases, it is possible to retrieve filters that are as expressive as CNNs on regular data (Grassi et al., 2017; Vignac & Frossard, 2019). This approach is however limited when dealing with eigenspaces of dimension more than one, as the choice of eigenvectors is in this case not unique.

Figure 2.3 – When visualizing the filters learned by linear MPNNs on images, we observe an isotropy phenomenon: the value of the filters at a given pixel only depends to its distance from the kernel center. Figure from (Levie et al., 2018).

Oversmoothing (Cai & Wang, 2020). Early graph neural network architectures, such as the Graph Convolutional Network (GCN) of Kipf & Welling (2016), suffer from a phenomenon called oversmoothing (Cai & Wang, 2020). Oversmoothing designates the fact that, when many message-passing layers are used, some architectures tend to forget local information and learn similar embeddings for all nodes in a graph. Oversmoothing often leads to degraded performance, which explains why early architectures would typically only use 2 or 3 message-passing layers (Shchur et al., 2018). To mitigate this problem, residual connections can be included in the network, which allows for the effective use of graph networks with more layers. However, the design of very deep graph networks is still an active area of research (Liu et al., 2020; Rusch et al., 2023).

Homophily (Zhu et al., 2020). Homophily refers to the tendency of graph neural networks to learn functions that smooth the features over the graph, i.e., as low-pass filters (Gasteiger et al., 2018; Wu et al., 2019; Nt & Maehara, 2019). As such, it is closely related to oversmoothing. The inductive bias towards homophily is highly effective when the task involves making similar predictions for nearby nodes, such as predicting the scientific field of a paper given the citation network. However, homophily can be problematic if the task requires predicting opposite labels for neighboring nodes, such as in maximal independent set search. While standard architectures may suffer from this issue, carefully designed architectures can avoid it (Zhu et al., 2021; Yang et al., 2021b). Recent works, however, have questioned homophily, arguing that standard architectures can still work well on datasets that require heterophily, as long as the distribution of rooted subtrees differs across classes (Ma et al., 2021).

Oversquashing (Alon & Yahav, 2020). Oversquashing refers to the fact that the number of paths considered in the computational tree of MPNNs grows exponentially as the number of message-passing layers increases. This makes it difficult to find one particular path within a graph, as required by certain tasks, for example in algorithmic reasoning (Cappart et al., 2021). While oversquashing is inherent to message-passing architectures, the use of the $\max$ aggregation operator instead of sum or mean can mitigate this phenomenon.

Vulnerability to adversarial attacks Deep neural networks are known to be susceptible to small input changes known as adversarial attacks that result in a drastic change in the network predictions (Szegedy et al., 2013). In the context of graph neural networks (GNNs), adversarial attacks can be applied to manipulate the graph structure, node features, or both, with the goal of altering the GNN's output in a targeted way (Zügner et al., 2018; Sun et al., 2020a). An important feature of adversarial attacks is that they should be imperceptible. For the graph structure, it is clear that if an edge is removed that turns a connected graph into disconnected graph, this change can have large effects on a classifier. However, the corresponding perturbation would be very noticeable. To avoid this issue, some works analyze adversarial perturbations in the graph spectral domain, which allows to provide measures of perceptibility of a perturbation (Li et al., 2022; Lin et al., 2022). To alleviate the network sensitivity to adversarial attacks, methods such as defensive dropout (Wang et al., 2018) or adversarial training (Bojchevski & Günnemann, 2019) can be used. While we will not consider adversarial attacks in this thesis, their existence need to be kept in mind for sensitive applications.

Conclusion Overall, we can observe that the limitations of graph neural networks that matter in practice depend a lot on the dataset. In general, we can classify graph datasets into two regimes: datasets which feature graphs with many attributes, and datasets that feature graphs with few attributes. On graphs with many node attributes, the expressive power of graph neural networks is not really an issue, as most rooted subtrees are distinct. Usually, structural descriptors are not key, as the node features already provides rich information. In such settings, the main challenge is to understand whether the graph neural networks really learn complex functions, or if they only learn the equivalent of low-pass graph filters. It is often observed empirically that extremely simple graph architectures such as APPNP (Gasteiger et al., 2018) are close to state-of-the-art, which questions the need for developing more complex architectures for these datasets.

On the contrary, on graphs with relatively few node attributes such as molecules (where we often only have access to atom types and bond types), or on relational reasoning tasks, the limited expressive power of graph networks becomes an important issue. In these settings, we are looking for neural networks that are able to recognise patterns in the graphs, such as functional group in molecules. It is clear usually clear that tasks on such datasets cannot be solved with simple graph filters, and powerful neural networks have proved to be more effective than simple MPNNs.

2.5 Alternative perspectives on message-passing neural networks

While we introduced message-passing neural networks (MPNNs) as local permutation equivariant operators, it is worth noting that they can also be viewed from other perspectives: the spatial perspective generalizes convolutions to graphs through the definition of a weight-sharing mechanism, while the spectral perspectives defines convolution trough the spectral convolutional theorem. It is interesting to note that, while all these perspectives are very different, they eventually result in similar formulation of MPNNs. Here we briefly discuss the spatial and spectral frameworks.

2.5.1 MPNNs as a generalization of CNN to graphs (spatial perspective)

The spatial perspective on graph neural networks views them as a generalization of convolutional neural networks (CNNs) to graphs. This perspective builds on the remarkable success of CNNs on image processing tasks (Krizhevsky et al., 2017). There are many similarities between images and graphs, the first one being that the computation of local descriptors of the data is known to be important. Furthermore, the same pattern can appear at various locations of an image, similar to how the same subgraph can be present at different locations in a graph. Thus, weight sharing is essential in both cases. Additionally, both graphs and images can be analyzed at multiple scales. These similarities have led to the development of convolutional layers and pooling operators (Defferrard et al., 2016; Khasanova & Frossard, 2017; Ying et al., 2018; Ahmadi, 2020) for graphs.

However, images have a regular structure which is not the case for graphs. This creates creating specific challenges that need to be addressed. For example, consider a 2D convolutional kernel for images, where a different coefficient matrix is learned at each location. With a 3x3 kernel, the following equation describes the convolution operation around a central pixel (i, j):

$$\forall (i,j),\ \mathrm{CNN}(\boldsymbol{X})_{ij} = \sum_{k=-1}^{1} \sum_{l=-1}^{1} \boldsymbol{W}_{kl}\, \boldsymbol{x}_{i-k,j-l} \tag{2.24}$$

As images have a regular structure, shifting a convolutional kernel to another central pixel (i', j') does not pose significant problems (except for border effects) as there is a clear way to map the kernel weights to other pixels. Consider now the application of a similar kernel to a graph: a central node is chosen, and a coefficient is learned per neighbor. This is expressed in the following equation:

$$\forall v_i \in \mathbb{V},\ \mathrm{GraphCNN}(\boldsymbol{X})_i = \sum_{v_j \in N(v_i) \cup \{v_i\}} \boldsymbol{W}_j\, \boldsymbol{x}_j \tag{2.25}$$

Weight-sharing is a key component of convolutional networks, so we must define the same kernel centered around other nodes in the graph. This poses a problem illustrated in Fig. 2.4: while the center of the kernel is well-defined, it is unclear how to map the neighbors of one node to those of another. To address this challenge, two strategies have been proposed. The first method is to use heuristics that minimize a distortion criterion (Grelier et al., 2016) when translating the kernel over the graph. The second and more common approach is to assign the same coefficient to all the neighbors of a node, resulting in:

$$\forall v_i \in \mathbb{V},\ \mathrm{GraphCNN}(\boldsymbol{X})_i = \boldsymbol{W}_1 \boldsymbol{x}_i + \sum_{v_j \in N(v_i)} \boldsymbol{W}_2 \boldsymbol{x}_j \tag{2.26}$$

This equation represents the simplest form of MPNNs, and we recognise a formula that it close to the Deep Sets of Eq. 2.7. Non-linear versions of this function lead to the more general formulation

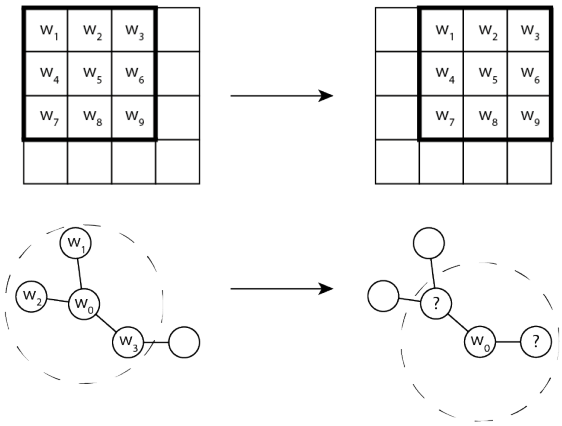


Figure 2.4 – In CNNs for Euclidean data, translating kernels does not pose particular problems (except for border effects). On graphs however, there is no obvious way to move a kernel centered at one node to another location. The solution to this problem is to attribute the same coefficient to all neighbors of the central node. The resulting convolution writes $f(\boldsymbol{X}, \boldsymbol{A})_i = \boldsymbol{W}_0^T \boldsymbol{x}_i + \sum_{v_j \in N(v_i)} \boldsymbol{W}_1^T \boldsymbol{x}_j$, which corresponds to a MPNN.

of MPNNs:

$$\forall v_i \in \mathbb{V}, f(\boldsymbol{X})_i = h(\boldsymbol{x}_i, \sum_{v_j \in N(v_i)} g(\boldsymbol{x}_j)) \tag{2.27}$$

This spatial formulation show that MPNNs constitute the most natural way to extend CNNs to graphs, and that the use of a permutation invariant aggregation function can be explained by the difficulty of defining a weight-sharing scheme on irregular structures. This is the approach that motivated the pioneering work of (Scarselli et al., 2008).

2.5.2 Graph convolutions in the Fourier domain (spectral perspective)

Another perspective introduces convolutions on graphs through the Graph Fourier Transform (GFT) and the convolution theorem. At the core of this idea is the observation that Laplacian matrices can be defined that converge to the standard Laplacian under certain hypotheses (Hein et al., 2005) as more and more points are sampled uniformly on a manifold.

Various definitions of the Laplacian can be used. The unnormalized Laplacian $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$ and the symmetric normalized Laplacian $\boldsymbol{L} = \boldsymbol{I}_n - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ are the most popular choices, as they lead to symmetric Laplacian matrices. However, the random walk Laplacian $\boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A}$ is also a natural quantity that arises when considering the probability of visiting nodes in a random walk.

Since the Fourier basis is the eigenbasis of the Laplacian operator on Euclidean domains, the GFT basis is similarly defined as the eigenbasis of the graph Laplacian. Specifically, if we denote the eigendecomposition of $\boldsymbol{L}$ by $\boldsymbol{L} = \boldsymbol{U}^T \boldsymbol{\Lambda} \boldsymbol{U}$, then $\boldsymbol{U}^T$ is interpreted as the change of basis

from the spatial domain to the spectral domain. The convolution theorem states that a convolution in the spatial domain corresponds to a pointwise multiplication in the spectral domain.

Spectral Graph convolutions (Henaff et al., 2015) use the spectral theorem as a definition. It defines the convolution of two signals $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^n$ on a graph as:

$$\boldsymbol{x} \star_G \boldsymbol{y} := \boldsymbol{U}^T(\boldsymbol{U}\boldsymbol{x} \odot \boldsymbol{U}\boldsymbol{y}) \tag{2.28}$$

Here, $\star_G$ designates the graph convolution operator, and $\odot$ is a point-wise multiplication. While convolution is mathematically defined between two signals, in signal processing and machine learning, one of the two signals constitute the kernel (or filter), while the other signal corresponds to the data $\boldsymbol{x}$. This can be expressed as:

$$f(\boldsymbol{X}, \boldsymbol{A}) = \boldsymbol{U}^T(\boldsymbol{w} \odot (\boldsymbol{U}\boldsymbol{x})) \tag{2.29}$$

where $\boldsymbol{w}$ is a vector of learnable parameters of length n. However, this formulation is not very convenient, as the learned filter on one graph of size n cannot be used for another graph of size $n' \neq n$. To address this limitation, spectral networks are typically constructed by defining a differentiable function $g_\theta : \mathbb{R} \to \mathbb{R}$ and computing:

$$f(\boldsymbol{X}, \boldsymbol{A}) = \boldsymbol{U}^T g_\theta(\Lambda)\boldsymbol{U}\boldsymbol{x} \tag{2.30}$$

Here, $g_\theta(\Lambda)$ is obtained by applying the element-wise function g_θ to the eigenvalues Λ_i of the graph Laplacian, vectorizing the result as $\mathrm{vec}((g_\theta(\Lambda_i))_{i \leq n}))$ and then mapping it to a diagonal matrix. This formulation of spectral convolution is transferable across graphs of different sizes and has some stability properties. Specifically, if a graph is perturbed by adding or removing edges, it is possible to bound the change in the response of the convolutional filter (Kenlay et al., 2021).

Despite its sound mathematical foundation, this definition of graph convolutions suffers from some practical limitations. Firstly, its naive implementation requires the computation of the eigendecomposition of the graph Laplacian, which has a complexity of $O(n^3)$. Moreover, although a smooth kernel can be defined in the spectral domain to achieve fast decay in the spatial domain, the resulting kernel is not strictly localized, which can hinder its applicability on large graphs.

To address some of the limitations of the original spectral convolution definition, Defferrard et al. (2016) proposed a new formulation built on the observation that Laplacian polynomials are a particular form of spectral filters. Specifically, Laplacian polynomials can be expressed as:

$$\sum_{k=1}^{K} \alpha_k \boldsymbol{L}^k = \boldsymbol{U}^T(\sum_{k=1}^{K} \alpha_k \Lambda^k)\boldsymbol{U}, \tag{2.31}$$

where $\boldsymbol{L}$ is the graph Laplacian, α_k are coefficients, and Λ is a diagonal matrix containing the eigenvalues of $\boldsymbol{L}$. Conversely, any spectral filter can be represented as a Laplacian polynomial with order up to n (Thanou et al., 2014; Sandryhaila & Moura, 2014).

The advantage of expressing spectral convolutions as Laplacian polynomials is that this formulation eliminates the need for computing the $O(n^3)$ eigendecomposition of the graph Laplacian. Furthermore, computing a polynomial filter at a node v_i (i.e., computing $(\sum_{k=1}^{K} \alpha_k \boldsymbol{L}^k \boldsymbol{X})_i$) only requires the K-hop neighborhood of v_i, making Laplacian polynomials strictly localized filters.

To enhance computational efficiency in graph convolutional networks, Chebyshev polynomials have been proposed as a class of orthogonal polynomials that can be efficiently computed (Shuman et al., 2011; Defferrard et al., 2016). The use of Chebyshev polynomials allows for a low-order approximation of the spectral filters in the Fourier domain, which can be computed efficiently using only the graph structure and a few parameters.

It is important to note that the distinction between spectral and spatial networks is not always clear-cut. In fact, any polynomial can be recursively formulated as $\sum_{k=0}^{K} \alpha_k x^k = (\beta_K x + \gamma_K 1)(\beta_{K-1} x + \gamma_{K-1} 1)...(\beta_1 x + \gamma_1)$ for a sequence $(\beta_k, \gamma_k) k \leq K$. For Laplacian polynomials, this means that any $\sum k = 1^K \alpha_k \boldsymbol{L}^k \boldsymbol{X}$ can be expressed as the recursive application of operators of the form $\beta \boldsymbol{L} + \gamma \boldsymbol{I}$ to a matrix $\tilde{\boldsymbol{X}}$.

In the case of the combinatorial Laplacian, we have:

$$[(\beta \boldsymbol{L} + \gamma \boldsymbol{I})\tilde{\boldsymbol{X}}]_i = [(\gamma \boldsymbol{I} - \beta \boldsymbol{D} + \beta \boldsymbol{A})\tilde{\boldsymbol{X}}]_i = (\gamma - \beta d_i)\boldsymbol{x}_i + \gamma \sum_{v_j \in N(v_i)} \beta \tilde{\boldsymbol{x}}_j$$

Here, we observe that applying a Laplacian polynomial of degree one to a signal can be interpreted and implemented as a message-passing step. Laplacian polynomials, which are defined spectrally, can therefore be implemented as MPNNs. This spatial implementation is commonly used in practice, as it allows for leveraging the potential sparsity of the adjacency matrix.

As a result, the distinction between spatial and spectral methods is not always clear, and spectral methods tend to be applied spatially in practice. Moreover, any linear message-passing neural network (MPNN) admits a spectral interpretation. This interpretation has been used to build efficient graph neural networks that can be trained without the use of graph libraries (Gasteiger et al., 2018; Wu et al., 2019) and to demonstrate that standard architectures tend to favor homophily (Nt & Maehara, 2019), as acted earlier.

Finally, it is important to mention the limitations of the Graph Fourier Transform. Although the eigenbasis of the Laplacian for Euclidean data is the Fourier Transform, the GFT was introduced as the eigenbasis of the graph Laplacian. However, the Laplacian operator is a second-order real-valued operator, which means that it does not depend on the phase of a complex signal. As a result, the graph Fourier Transform is real-valued, and not all properties of the Fourier transform apply to graphs. For instance, translation on a graph is not invertible. These issues arise because

the Fourier transform is fundamentally not defined as the eigenbasis of the Laplacian, but rather as the irreps corresponding to a symmetry group (Kondor, 2008). Unfortunately, it is not easy to extend this definition to graphs since there is no symmetry group that plays the exact same role as translations for Euclidean data. In summary, the GFT is not a perfect analog of the Fourier transform for graphs. It is still a very useful tool in many settings, but its limitations should be kept in mind.

Conclusion

In this chapter, we have seen that Message-Passing Neural Networks can be introduced in several ways. They can be seen as local permutation equivariant operators, as extensions of CNNs that implement a message-passing scheme on graphs, or as fast graph spectral filters. In all cases, a similar constraint arises, which is the fact that MPNNs need to feature a symmetric aggregation function. This function is required to achieve permutation equivariance, but it also has a strong impact on the expressive power of graph neural networks. First, kernels learned with a permutation invariant aggregation are isotropic, and they cannot learn to favor one direction over others. Second, when the nodes do not have unique identifiers, it is not possible for a node to know where incoming messages come from. This results in several local structures that are mapped to the same rooted subtree, which explains why MPNN are not more powerful than the Weisfeiler-Lehman test.

In the next chapter, we will therefore introduce identifiers to alleviate the expressivity limitations of MPNNs. We will show that it is possible to do so without losing permutation equivariance, at the cost of manipulating higher-order tensors.

3 Structural message-passing

3.1 Introduction

In previous chapter, we have seen that MPNNs are able to exploit the sparsity of graphs, have an inductive bias that is considered as well-suited to relational reasoning (Xu et al., 2020a), and satisfy permutation equivariance. As a result, they have become very popular on tasks such as such as tractable relational inference (Yoon et al., 2019; Satorras et al., 2019), problems in combinatorial optimization (Khalil et al., 2017; Li et al., 2018b; Joshi et al., 2019; Karalias & Loukas, 2020) or the simulation of physical interactions between objects (Battaglia et al., 2016; Sanchez-Gonzalez et al., 2018).

Despite their success, equivariant MPNNs possess limited expressive power (Xu et al., 2019; Morris et al., 2019), and cannot detect whether a graph is connected or if it contains cycles (Chen et al., 2020). For tasks where the graph structure is important, such as the prediction of chemical properties of molecules (Elton et al., 2019; Sun et al., 2019) and the solution to combinatorial optimization problems, more powerful graph neural networks are necessary.

Aiming to address these challenges, this work puts forth *structural message-passing* (SMP)—a new type of graph neural network that is strictly more powerful than MPNNs, while also sharing the attractive inductive bias of message-passing architectures. SMP inherits its power from its ability to manipulate node identifiers. However, in contrast to previous studies that relied on identifiers (Murphy et al., 2019; Loukas, 2020a), it does so *in a permutation equivariant way* without introducing new sources of randomness. As a result, SMP can be powerful without sacrificing its ability to generalize to unseen data. In particular, we show that if SMP is built out of powerful layers, the resulting model is computationally universal over the space of equivariant functions.

Concretely, SMP maintains at each node a matrix called "local context" (instead of a feature vector as in MPNNs) that is initialized with a one-hot encoding of the nodes and the node features. These local contexts are then propagated in such a way that a permutation of the nodes or a

change in the one-hot encoding will reorder the lines of each context without changing their content, which is key to efficient learning and good generalization.

We evaluate SMP on a diverse set of structural tasks that are known to be difficult for message-passing architectures, such as cycle detection, connectivity testing, diameter and shortest path distance computation. In all cases, our approach compares favorably to previous methods: for example, SMP solves cycle detection in all evaluated configurations, whereas other powerful networks struggle when the graphs become larger, and MPNNs do not manage to solve the task completely.

Finally, we evaluate our method on the ZINC chemistry dataset and achieve state-of-the-art performance among methods that do not use expert features. It shows that SMP is able to successfully learn both from the features and from topological information, which is essential in chemistry applications. Overall, our method is able to overcome a major limitation of MPNNs, while retaining their ability to process features with a bias towards locality.

3.2 Related work

3.2.1 Permutation equivariant graph neural networks

In order to build equivariant networks that are more expressive than the 1-WL test, (Morris et al., 2019) proposed to exploit the hierarchy of higher-dimensional k-WL tests. However, these higher-order networks are global, in the sense that they iteratively update the state of a k-tuple of nodes based on all other nodes (and not only neighbours), a procedure which is very costly both in time and memory. While a faster procedure was proposed in Morris et al. (2020) concurrently to our work, key differences with SMP remain: we propose to learn richer embeddings for each node instead of one embedding per k-tuple of nodes, and build our theoretical analysis on distributed algorithms rather than vertex refinement methods.

Recent studies have also characterized the expressive power of MPNNs from other perspectives, such as the ability to approximate continuous functions on graphs (Chen et al., 2019) and solutions to combinatorial problems (Sato et al., 2019), highlighting similar limitations of MPNNs — see also (Barceló et al., 2019; Geerts et al., 2020; Sato, 2020; Garg et al., 2020; Magner et al., 2020).

Beyond higher-order message-passing architectures, there have been efforts to construct more powerful equivariant networks. One way to do so is to incorporate hand-crafted topological features (such as the presence of cliques or cycles) Bouritsas et al. (2022), which requires expert knowledge on what features are relevant for a given task. A more task-agnostic alternative is to build networks by arranging together a set of simple permutation equivariant functions and operators. These building blocks are:

- Linear equivariant functions between tensors of arbitrary orders: a basis for these functions was computed by Maron et al. (2018), by solving the linear system imposed by

equivariance.
- Element-wise functions, applied independently to each feature of a tensor.
- Operators that preserve equivariance, such as $+$, $-$, tensor and elementwise products, composition and concatenation along the dimension of the channels.

Similarly to Morris et al. (2019), networks built this way obtain a better expressive power than MPNN by using higher-order tensors (Kondor et al., 2018; Maron et al., 2018). Since k-th order tensors can represent any k-tuple of nodes, architectures manipulating them can exploit more information to compute structural properties (and be as powerful as the k-WL test). Unfortunately, memory requirements are exponential in the tensor order, which makes these methods of little practical interest. More recently, Maron et al. (2019a) proposed provably powerful graph networks (PPGN) based on the observation that the use of matrix multiplication can make their model more expressive for the same tensor order. This principle was also used in the design of Ring-GNN (Chen et al., 2019), which has many similarities with PPGN. Key differences between such methods and ours are that (*i*) SMP can be parametrized to have a lower time complexity, due to the ability of message-passing to exploit the sparsity of adjacency matrices, (*ii*) SMP retains the message-passing inductive bias, which is different from PPGN and, as we will show empirically, makes it better suited to practical tasks such as the detection of substructures in a graph.

3.2.2 Non-equivariant graph neural networks

In order to better understand the limitations of current graph neural networks, analogies with graph theory and distributed systems have been exploited. In these fields, a large class of problems cannot be solved without using node identifiers (Alon et al., 1995; Suomela, 2013). The reasoning is that, in message-passing architectures, each node has access to a local view of the graph created by the reception of messages. Without identifiers, each node can count the number of incoming messages and process them, but cannot tell from how many unique nodes they come from. They are therefore unable to reconstruct the graph structure.

This observation has motivated researchers to provide nodes with randomly selected identifiers (Murphy et al., 2019; Loukas, 2020a; Dasoulas et al., 2019; Sato et al., 2020). Encouragingly, by showing the equivalence between message-passing and a model in distributed algorithms, Loukas (2020a) proved that graph neural networks with identifiers and sufficiently expressive message and update functions can be Turing universal, which was also confirmed on small instances of the graph isomorphism problem (Loukas, 2020b).

Nevertheless, the main issue with these approaches is sample efficiency. Identifiers introduce a dependency of the network on a random input and the loss of permutation equivariance, causing poor generalization. Although empirical evidence has been presented that the aforementioned dependency can be overcome with large amounts of training data or other augmentations (Murphy et al., 2019; Loukas, 2020b), overfitting and optimization issues can occur. In this work, we propose to overcome this problem by introducing a network which is both powerful and

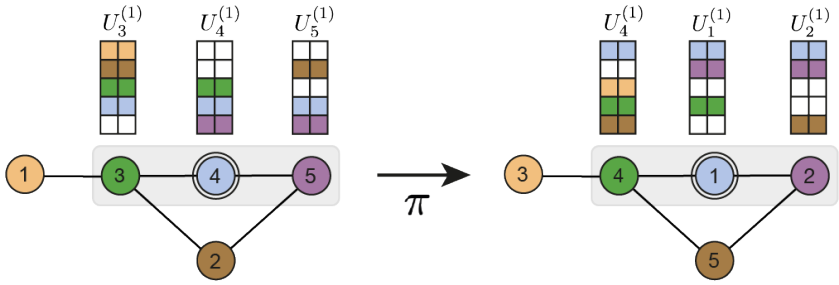


Figure 3.1 – In the SMP model, each local context $\boldsymbol{U}_i^{(l)}$ is an $n \times c_l$ matrix, with each row storing the c_l-dimensional representation of a node (denoted by color). The figure shows the local context in the output of the first layer and blank rows correspond to nodes that have not been encountered yet. Upon node reordering, the lines of the local context are permuted but their content remains unchanged.

permutation equivariant.

3.3 Structural message-passing

We present the structural message-passing neural networks (SMP), as generalization of MPNNs that follows a similar design principle. However, rather than processing vectors with permutation invariant operators, SMP propagates *matrices* and processes them in a permutation equivariant way. This subtle change greatly improves the network's ability of to learn information about the graph structure.

3.3.1 Method

In SMP, each node of a graph maintains a *local context* matrix $\boldsymbol{U}_i \in \mathbb{R}^{n \times c}$ rather than a feature vector $\boldsymbol{x}_i \in \mathbb{R}^{d_x}$ as in MPNN. The j-th row of $\boldsymbol{U}_i$ contains the c-dimensional representation that node v_i has of node v_j. Intuitively, equivariance means that the lines of the local context after each layer are simply permuted when the nodes are reordered, as shown in Fig. 3.1.

Initialization The local context is initialized as a one-hot encoding $\boldsymbol{U}_i^{(0)} = \mathbb{1}_i \in \mathbb{R}^{n \times 1}$ for every $v_i \in V$, which corresponds to having initially a unique identifier for each node. In addition, if there are features $\boldsymbol{x}_i$ associated with node v_i, they are appended to the same row of the local context as the identifiers: $\boldsymbol{U}_i^{(0)}[i,:] = [1, \boldsymbol{x}_i] \in \mathbb{R}^{1+d_x}$. Later, we will show that when SMP is parameterized in the proper way, the ordering induced by the one-hot encoding is actually irrelevant to the output.

Layers At layer $l+1$, the state of each node is updated as in standard MPNNs (Battaglia et al., 2018): messages are computed on each edge before being aggregated into a single matrix via a

symmetric function. The result can then be updated using the local context of previous layer at this node:

$$\boldsymbol{U}_i^{(l+1)} = u^{(l)}\left(\boldsymbol{U}_i^{(l)}, \tilde{\boldsymbol{U}}_i^{(l)}\right) \in \mathbb{R}^{n \times c_{l+1}} \quad \text{with} \quad \tilde{\boldsymbol{U}}_i^{(l)} = \phi\left(\left\{m^{(l)}(\boldsymbol{U}_i^{(l)}, \boldsymbol{U}_j^{(l)}, \boldsymbol{y}_{ij})\right\}_{v_j \in N(v_i)}\right)$$

Above, $u^{(l)}$, $m^{(l)}$, ϕ are the *update*, *message* and *aggregation* functions of the $(l+1)$-th layer, respectively, whereas c_{l+1} denotes the layer's width.

It might be interesting to observe that, starting from a one-hot encoding and using the update rule $\boldsymbol{U}_i^{(l+1)} = \sum_{v_j \in N(v_i)} \boldsymbol{U}_j^{(l)}$, SMP iteratively compute powers of $\boldsymbol{A}$. Since $\boldsymbol{A}^l[i,j]$ corresponds to the count of walks of length l between v_i and v_j, there is a natural connection between the propagation of identifiers and the detection of topological features: even with simple parametrizations, SMP can manipulate polynomials in the adjacency matrix and therefore learn spectral properties (Sandryhaila & Moura, 2014) that MPNNs cannot detect. In the following, it will be convenient to express each SMP layer $f^{(l)}$ in a tensor form:

$$\mathbf{U}^{(l+1)} = f^{(l)}(\mathbf{U}^{(l)}, \mathbf{Y}, \boldsymbol{A}) = [\boldsymbol{U}_1^{(l+1)}, \ldots, \boldsymbol{U}_n^{(l+1)}] \ \in \ \mathbb{R}^{n \times n \times c_{l+1}}$$

Pooling After all L message-passing layers have been applied, the aggregated contexts $\mathbf{U}^{(L)}$ can be pooled to a vector or to a matrix (e.g, for graph and node classification, respectively). To obtain an equivariant representation for node classification, we aggregate each $\boldsymbol{U}_i^{(L)} \in \mathbb{R}^{n \times c_L}$ into a vector using an invariant neural network for sets σ (Zaheer et al., 2017; Qi et al., 2017) applied simultaneously to each node v_i:

$$f_{eq}(\mathbf{U}^{(0)}, \mathbf{Y}, \boldsymbol{A}) = \sigma \circ f^{(L)} \circ \cdots \circ f^{(1)}(\mathbf{U}^{(0)}, \mathbf{Y}, \boldsymbol{A}) \ \in \ \mathbb{R}^{n \times c},$$

whereas a permutation invariant representation is obtained after the application of a pooling function *pool*. It may be a simple sum or average followed by a soft-max, or a more complex operator (Ying et al., 2018):

$$f_{inv}(\mathbf{U}^{(0)}, \mathbf{Y}, \boldsymbol{A}) = pool \circ f_{eq}(\mathbf{U}^{(0)}, \mathbf{Y}, \boldsymbol{A}) \ \in \ \mathbb{R}^{c}$$

3.3.2 Analysis

The following section characterizes the equivariance properties and representation power of SMP.

Equivariance Before providing sufficient conditions for permutation equivariance, we define it formally. A change in the ordering of n nodes can be described by a permutation π of the symmetric group $\mathbb{S}_n$. π acts on a tensor by permuting the axes indexing nodes (but not the other axes):

$$(\pi \,.\, \mathbf{U})[i,j,k] = \mathbf{U}[\pi^{-1}(i),\ \pi^{-1}(j), k], \quad \text{where} \quad \mathbf{U} \in \mathbb{R}^{n \times n \times c}$$

For vector and matrices, the action of a permutation is more easily described g the matrix $\boldsymbol{\pi}$ canonically associated to π: $\pi.z = z$ for $z \in \mathbb{R}^c$, $\pi.\boldsymbol{X} = \boldsymbol{\pi}^T \boldsymbol{X}$ for $\boldsymbol{X} \in \mathbb{R}^{n \times c}$, and $\pi.\boldsymbol{A} = \boldsymbol{\pi}^T \boldsymbol{A}\, \boldsymbol{\pi}$ for $\boldsymbol{A} \in \mathbb{R}^{n \times n}$. An SMP layer f is said to be permutation equivariant if permuting the inputs and applying f is equivalent to first applying f and then permuting the result:

$$\forall \pi \in \mathbb{S}_n, \quad \pi \,.\, f(\mathbf{U}, \mathbf{Y}, \boldsymbol{A}) = f(\pi.\mathbf{U}, \pi.\mathbf{Y}, \pi.\boldsymbol{A}))$$

We stress that an equivariant SMP network should yield the same output (up to a permutation) for every one-hot encoding used to construct the node identifiers. We can now state some sufficient conditions for equivariance:

Theorem 1 (Permutation equivariance)**.** *Let functions m, ϕ and u be permutation equivariant, that is, for every permutation $\pi \in \mathbb{S}_n$ we have $u(\pi.\boldsymbol{U}, \pi.\boldsymbol{U}') = \pi.u(\boldsymbol{U}, \boldsymbol{U}')$, $\phi(\{\pi.\boldsymbol{U}_j\}_{v_j \in N(v_i)}) = \pi.\phi(\{\boldsymbol{U}_j\}_{v_j \in N(v_i)})$, and $m(\pi.\boldsymbol{U}, \pi.\boldsymbol{U}', \boldsymbol{y}) = \pi.m(\boldsymbol{U}, \boldsymbol{U}', \boldsymbol{y})$.*
Then, SMP is permutation equivariant.

Proof. Let f be a layer of SMP:

$$f(\mathbf{U}, \mathbf{Y}, \boldsymbol{A})[i, :, :] = u(\boldsymbol{U}_i, \phi(\{m(\boldsymbol{U}_i, \boldsymbol{U}_j, \boldsymbol{y}_{ij})\}_{v_j \in N(v_i)})) = u(\boldsymbol{U}_i, \phi(\{m(\boldsymbol{U}_i, \boldsymbol{U}_j, \boldsymbol{y}_{ij})\}_{v_j : A[i,j] > 0}))$$

The action of a permutation π on the inputs is defined as $f(\pi.(\mathbf{U}, \mathbf{Y}, \boldsymbol{A})) = f(\pi.\mathbf{U}, \pi.\mathbf{Y}, \pi.\boldsymbol{A})$. In order to simplify notation, we will consider π^{-1} instead of π. We have for example $(\pi^{-1}.\, \boldsymbol{A})[i, j] = A[\pi_i, \pi_j]$ and $(\pi^{-1}.\mathbf{U})[i, j, k] = \mathbf{U}[\pi_i, \pi_j, k]$, which can be written as

$$(\pi^{-1}.\mathbf{U})[i, :, :] = \boldsymbol{\pi}\, \boldsymbol{U}_{\pi_i}.$$

As shown next, the theorem's conditions suffice to render SMP equivariant:

$$\begin{aligned}
f(\pi^{-1}.(\mathbf{U}, \mathbf{Y}, \boldsymbol{A}))_{i::} &= u(\boldsymbol{\pi}\, \boldsymbol{U}_{\pi_i},\ \phi(\{m(\boldsymbol{\pi}\, \boldsymbol{U}_{\pi_i},\ \boldsymbol{\pi}\, \boldsymbol{U}_{\pi_j},\ \boldsymbol{y}_{\pi_i \pi_j})\}_{v_j : A[\pi_i, \pi_j] > 0})) \\
&= u(\boldsymbol{\pi}\, \boldsymbol{U}_{\pi_i},\ \phi(\{m(\boldsymbol{\pi}\, \boldsymbol{U}_{\pi_i},\ \boldsymbol{\pi}\, \boldsymbol{U}_k,\ \boldsymbol{y}_{\pi_i k})\}_{v_k : A[\pi_i, k] > 0})) && (\pi \text{ bijective}) \\
&= u(\boldsymbol{\pi}\, \boldsymbol{U}_{\pi_i}, \boldsymbol{\pi}\, \phi(\{m(\boldsymbol{U}_{\pi_i}, \boldsymbol{U}_k, \boldsymbol{y}_{\pi_i k})\}_{v_k : A[\pi_i, k] > 0})) && (\phi, m \text{ equivariant}) \\
&= \boldsymbol{\pi}\, u(\boldsymbol{U}_{\pi_i}, \phi(\{m(\boldsymbol{U}_{\pi_i}, \boldsymbol{U}_k, \boldsymbol{y}_{\pi_i k})\}_{v_k : A[\pi_i, k] > 0})) && (u \text{ equivariant}) \\
&= \boldsymbol{\pi}\, f(\mathbf{U}, \mathbf{Y}, \boldsymbol{A})[\pi_i, :, :] \\
&= (\pi^{-1}.f(\mathbf{U}, \mathbf{Y}, \boldsymbol{A}))[i, :, :],
\end{aligned}$$

which matches the definition of equivariance. □

This theorem defines the class of functions that can be used in our model. For example, if the message and update functions are operators applied simultaneously to each row of the local context, the whole layer is guaranteed to be equivariant. However, more general functions can

be used: each $\boldsymbol{U}_i$ is a $n \times c$ matrix which can be viewed as the representation of a set of nodes. Hence, any equivariant neural network for sets can be used, which allows the network to have several desirable properties:

- *Inductivity*: as an equivariant neural network for sets can take sets of different size as input, SMP can be trained on graphs with various sizes as well. Furthermore, it can be used in inductive settings on graphs whose size has not been seen during training, which we will confirm experimentally.
- *Invariance to local isomorphisms*: SMP learns *structural embeddings*, in the sense that it yields the same result on isomorphic subgraphs. More precisely, if the subgraphs G_i^k and G_j^k induced by G on the k-hop neighborhoods of v_i and v_j are isomorphic, then on node classification, any k-layer SMP f will yield the same result for v_i and v_j. This is in contrast with several popular methods (Zhang & Chen, 2018; You et al., 2019) that learn *positional embeddings* which do not have this property.

Representation and expressive power The following theorem characterizes the representation power of SMP when parametrized with powerful layers. Simply put, Theorem 2 asserts that it is possible to parameterize an SMP network such that it maps non-isomorphic graphs to different representations:

Theorem 2 (Representation power – informal)**.** *Consider the class S of simple graphs with n nodes, diameter at most Δ and degree at most d_{max}. We assume that these graphs have respectively d_x and d_y attributes on the nodes and the edges. Then, there exists a SMP network f of depth at most $\Delta + 1$ and width at most $2d_{max} + d_x + n\, d_y$ such that the full structure of any graph in S (with the attributes) can be recovered from the output of f at any node.*

The formal statement and the proof are detailed in Appendix A.1. We first show the result for the simple case where f can pass messages of size $n \times n$, and then consider the case of $n \times 2d_{\text{max}}$ matrices using the following lemma:

Lemma 1 (Maehara & Rödl (1990))**.** *For any simple graph $G = (V, E)$ of n nodes and maximum degree d_{max}, there exists a unit-norm embedding of the nodes $\boldsymbol{X} \in \mathbb{R}^{n \times 2d_{max}}$ such that for every $v_i, v_j \in V$, $(v_i, v_j) \in E \iff \langle \boldsymbol{X}_i, \boldsymbol{X}_j \rangle = 0$.*

The universality of SMP is a direct corollary: since each node can have the ability to reconstruct the adjacency matrix from its local context, it can also employ a universal network for sets (Zaheer et al., 2017) to compute any equivariant function on the graph. Interestingly, this result shows that propagating matrices instead of vectors might be a way to solve the bottleneck problem (Alon & Yahav, 2020): while MPNNs need feature maps that exponentially grow with the graph size in order to recover the topology, SMPs can do it with $O(d_{\text{max}} n^2)$ memory.

Corollary 1 (Expressive power)**.** *Let G be a simple graph of diameter at most Δ and degree at most d_{max}. Consider an SMP $f = f^{(L)} \circ \cdots \circ f^{(1)}$ of depth $L = \Delta$ and width $2d_{max} + d_x + n\, d_y$*

satisfying the properties of Theorem 4. Then, any equivariant function can be computed as $f_{eq} = \sigma \circ f$*, where* σ *is a universal function of sets applied simultaneously to each node. Similarly, any permutation invariant function can be computed as* $f_{in} = \frac{1}{n}\sum_{v_i \in V} \sigma \circ f$*.*

Proof. Lemma 1 proves the existence of an injective mapping from adjacency matrices of simple graphs to features for a set of nodes. Therefore, any permutation equivariant function $h_{\text{eq}}(\boldsymbol{A})$ on adjacency matrices can be expressed by an equivariant function on sets

$$\forall v_i \in V,\ h_{\text{eq}}(\boldsymbol{A}) = h'_{\text{eq}}(\boldsymbol{U}) \quad \text{with} \quad \boldsymbol{U}[i,:] = \boldsymbol{u}_i \in \mathbb{R}^{2d_{\max}+d_x+n\ d_y}$$

as long as the node embeddings $\boldsymbol{u}_1, \dots, \boldsymbol{u}_n$ allow the reconstruction of $\boldsymbol{A}$, e.g., through orthogonality conditions. It was proven in Theorem 4 that, under the corollary's conditions, the local context $\boldsymbol{U}_i^{(L)}$ of any node v_i yields an appropriate matrix $\boldsymbol{U}$. In order to compute h_{eq}, each node can then rely on the universal σ to compute the invariant function:

$$h''_{\text{inv}}(\boldsymbol{U}, \mathbb{1}_i) = h'_{\text{eq}}(\boldsymbol{U})[i,:] = h_{\text{eq}}(\boldsymbol{A})[i,:] \in \mathbb{R}^c.$$

For invariant functions $h_{in}(\boldsymbol{A}) \in \mathbb{R}^c$, it suffices to build the equivariant function $h_{eq}(\boldsymbol{A}) = [h_{in}(\boldsymbol{A}), \dots, h_{in}(\boldsymbol{A})] \in \mathbb{R}^{n \times c}$. Then, if each node v_i computes $h_{\text{eq}}(\boldsymbol{A})[i,:] = h_{in}(\boldsymbol{A})$, averaging will yield $\frac{1}{n}\sum_{v_i \in V} h_{\text{eq}}(\boldsymbol{A})[i,:] = h_{in}(\boldsymbol{A})$, as required. □

These results show that two components are required to build a universal approximator of functions on graphs. First, one needs an algorithm that breaks symmetry during message passing, which SMP manages to do in an equivariant manner. Second, one needs powerful layers to parameterize the message, aggregation and update functions. Here, we note that the proofs of Theorem 2 and Corollary 1 are not constructive and that deriving practical parametrizations that are universal remains an open question (Keriven & Peyré, 2019). Nevertheless, we do constructively prove the following more straightforward claim using a simple parametrization:

Proposition 1. *SMP is strictly more powerful than MPNN: SMP can simulate any MPNN with the same number of layers, but MPNNs cannot simulate all SMPs.*

First part of the proof: SMPs are at least as powerful as MPNNs We will show by induction that any MPNN can be simulated by an SMP:

Lemma 2. *For any MPNN mapping initial node features* $(\boldsymbol{x}_i^{(0)})_{v_i \in V}$ *to* $(\boldsymbol{x}_i^{(L)})_{v_i \in V}$*, there is an SMP with the same number of layers such that*

$$\forall\, v_i \in V,\ \forall\, l \leq L,\ \mathbf{U}^{(l)}[i,i,:] = \boldsymbol{x}_i^{(l)} \quad \textit{and} \quad \forall j \neq i,\ \mathbf{U}^{(l)}[i,j,:] = 0.$$

Proof. Consider a graph with node features $(\boldsymbol{x}_i^{(0)})_{v_i \in V}$ and edge features $(\boldsymbol{y}_{ij})_{(v_i,v_j) \in E}$.

Initialization: The context tensor is initialized by mapping the node features on the diagonal of $\mathbf{U}$: $\mathbf{U}^{(0)}[i, i, :] = \boldsymbol{x}_i^{(0)}$. The desired property is then true by construction.

Inductive step: Denote by $(\boldsymbol{x}_i^{(l)})_{v_i \in V}$ the features obtained after l layers of the MPNN. Assume that there is a k-layer SMP such that the local context after l layers contains the same features in its diagonal elements: $\mathbf{U}^{(l)}[i, i, :] = \boldsymbol{x}_i^{(l)}$ and 0 in the other entries. Consider one additional layer of MPNN:

$$\boldsymbol{x}_i^{(l+1)} = u(\boldsymbol{x}_i^{(l)}, \phi(\{m(\boldsymbol{x}_i^{(l)}, \boldsymbol{x}_j^{(l)}, \boldsymbol{y}_{ij})\}_{j \in N(v_i)}))$$

and the following SMP layer:

$$\boldsymbol{U}_i^{(l+1)} = diag(\tilde{u}(\boldsymbol{U}_i^{(l)}, \tilde{\phi}(\{\tilde{m}(\mathbf{1}\mathbf{1}^T\, \boldsymbol{U}_i^{(l)}, \mathbf{1}\mathbf{1}^T\, \boldsymbol{U}_j^{(l)}, \boldsymbol{y}_{ij})\}_{j \in N(v_i)}))),$$

where $\tilde{m}$, $\tilde{\phi}$ and $\tilde{u}$ respectively apply the functions m, ϕ and u simultaneously on each line of the local context $\boldsymbol{U}_i$. As the only non-zero line of $\boldsymbol{U}_i$ is $\boldsymbol{U}_i[i, :]$, $\mathbf{1}\mathbf{1}^T\boldsymbol{U}_i^{(l)}$ replicates the i-th line of $\boldsymbol{U}_i^{(l)}$ on all the other lines, so that they all share the same content $\boldsymbol{x}_i^{(l)}$. After the application of the message passing functions $\tilde{m}, \tilde{\phi}$ and $\tilde{u}$, all the lines of $\boldsymbol{U}_i$ therefore contain $\boldsymbol{x}_i^{(l+1)}$.

Finally, the function *diag* extracts the main diagonal of the tensor $\mathbf{U}$ along the two first axes. Let $\delta_{i,j}$ be the function that is equal to 1 if $i = j$ and 0, otherwise. We have: $diag(\mathbf{U})[i, j, :] = \mathbf{U}[i, j, :]\ \delta_{i,j}$. Note that this function can equivalently be written as an update function applied separately to each node: $diag(\boldsymbol{U}_i)[j, :] = \boldsymbol{U}_i[j, :]\delta_{i,j}$. We now have $\mathbf{U}^{(l+1)}[i, i; :] = \boldsymbol{x}_i^{l+1}$ and $\mathbf{U}$ equal to 0 on all the other entries, so that the induction hypothesis is verified at layer $l + 1$. As any MPNN can be computed by an SMP, we conclude that SMPs are at least as powerful as MPNNs. □

Second part: SMP networks are strictly more powerful than MPNNs To prove that SMPs are strictly more powerful than MPNNs, we use a similar argument to Chen et al. (2019); Maron et al. (2019a):

Lemma 3. *There is an SMP network which yields different outputs for the two graphs of Fig. 3.2, while any MPNN will view these graphs are isomorphic.*

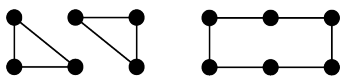


Figure 3.2 – While MPNNs cannot distinguish between two regular graphs such as these ones, SMPs can.

Proof. The two graphs of Fig. 3.2 are regular, which implies that they cannot be distinguished by the Weisfeiler-Lehman test or by MPNNs without special node features (Maron et al., 2019a). On the contrary, consider an SMP f made of three layers computing $\boldsymbol{U}_i^{(l+1)} = \sum_{v_i \in N(v_i)} \boldsymbol{U}_i^{(l)}$, followed by the trace of $\boldsymbol{U}^{(3)}$ as a a pooling function. As each layer can be written $\boldsymbol{U}^{(l+1)} =$

$\boldsymbol{A}\boldsymbol{U}^{(l)}$ and $\boldsymbol{U}^{(0)} = \boldsymbol{I}_n$, we have $f(\boldsymbol{A}) = tr(\boldsymbol{A}^3)$. In particular $f(\boldsymbol{A}) = 2$ for the graph on the left, while $f(\boldsymbol{A}) = 0$ on the right. □

3.4 Implementation

SMP offers a lot of flexibility in its implementation, as any equivariant function that combines the local context of two nodes and the edge features can be used. We propose two implementations that we found to work well, but our framework can also be implemented differently. In both cases, we split the computation of the messages in two steps. First, the local context of each node is updated using a neural network for sets. Then, a standard message passing network is applied separately on each row of the local contexts. For the first step, we use a subset of the linear equivariant functions computed by Maron et al. (2018):

$$\forall v_i \in V, \quad \hat{\boldsymbol{U}}_i^{(l)} = \boldsymbol{U}_i^{(l)}\, \boldsymbol{W}_1^{(l)} + \frac{1}{n}\mathbf{1}_n\, \mathbf{1}_n^T\, \boldsymbol{U}_i^{(l)}\, \boldsymbol{W}_2^{(l)} + \mathbf{1}_n(\boldsymbol{c}^{(l)})^\top + \frac{1}{n}\mathbb{1}_i \mathbf{1}^T \boldsymbol{U}_i^{(l)}\, \boldsymbol{W}_3^{(l)},$$

where $\mathbf{1}_n \in \mathbb{R}^{n\times 1}$ is a vector of ones, $\mathbb{1}_i \in \mathbb{R}^{n\times 1}$ the indicator of v_i, whereas $(\boldsymbol{W}_k^{(l)})_{1\leq k\leq 5}$ and $\boldsymbol{c}^{(l)}$ are learnable parameters. As for the message passing architecture, we propose two implementations with different computational complexities:

Default SMP This architecture corresponds to a standard MPNN, where the message and update functions are two-layer perceptrons. We use a sum aggregator normalized by the average degree d_{avg} over the graph: it retains useful properties of the sum (Xu et al., 2019), while also avoiding the exploding-norm problem (Velickovic et al., 2020). This network can be written:

$$m_{\text{def}}^{(l)}(\hat{\boldsymbol{U}}_i^{(l)}, \hat{\boldsymbol{U}}_j^{(l)}, \boldsymbol{y}_{ij}) = MLP(\hat{\boldsymbol{U}}_i^{(l)}, \hat{\boldsymbol{U}}_j^{(l)}, \boldsymbol{y}_{ij}) \tag{3.1}$$

$$\boldsymbol{U}_i^{(l+1)} = MLP(\hat{\boldsymbol{U}}_i^{(l)}, \textstyle\sum_{v_j\in N(v_i)} m_{\text{def}}^{(l)}(\boldsymbol{U}_i^{(l)}, \boldsymbol{U}_j^{(l)}, \boldsymbol{y}_{ij})/d_{\text{avg}}),$$

Fast SMP For graphs without edge features, we propose a second implementation with a message function that uses a point-wise multiplication $\odot$:

$$m_{\text{fast}}^{(l)}(\hat{\boldsymbol{U}}_i^{(l)}, \hat{\boldsymbol{U}}_j^{(l)}) = \hat{\boldsymbol{U}}_j^{(l)} + \left(\hat{\boldsymbol{U}}_i^{(l)}\boldsymbol{W}_4^{(l)}\right) \odot \left(\hat{\boldsymbol{U}}_j^{(l)}\boldsymbol{W}_5^{(l)}\right),$$

where $\boldsymbol{W}_4$ and $\boldsymbol{W}_5$ are learnable matrices. The aggregation is the same, and the update is simply a residual connection, so that the l-th SMP layer updates each node's local context as

$$\begin{aligned}\boldsymbol{U}_i^{(l+1)} &= \hat{\boldsymbol{U}}_i^{(l)} + \tfrac{1}{d_{\text{avg}}} \textstyle\sum_{v_j\in N(v_i)} m_{\text{fast}}^{(l)}(\boldsymbol{U}_i^{(l)}, \boldsymbol{U}_j^{(l)}) \\ &= \hat{\boldsymbol{U}}_i^{(l)} + \left(\textstyle\sum_{v_j\in N(v_i)} \hat{\boldsymbol{U}}_j^{(l)} \;+\; \hat{\boldsymbol{U}}_i^{(l)}\boldsymbol{W}_4^{(l)} \odot \textstyle\sum_{v_j\in N(v_i)} \hat{\boldsymbol{U}}_j^{(l)}\boldsymbol{W}_5^{(l)}\right)/d_{\text{avg}}\end{aligned}$$

In this last equation, the arguments of the two sums are only functions of the local context of

node v_j. This allows for a more efficient implementation, where one message is computed per node, instead of one per edge as in default SMP. One might notice that Fast SMP can be seen as a local version of PPGN:

Proposition 2. *A Fast SMP with k layers can be approximated by a $2k$-block PPGN.*

Proof. We will prove by induction that any Fast SMP layer can be approximated by two blocks of PPGN. It implies that the expressive power of Fast SMP is bounded by that of PPGN.

Recall that a block of PPGN is parameterized as:

$$\mathbf{T}^{(l+1)} = m_4(m_3(\mathbf{T}^{(l)}) \| m_1(\mathbf{T}^{(l)}) \mathbin{@} m_2(\mathbf{T}^{(l)})),$$

where m_k are MLPs acting over the third dimension of $\mathbf{T} \in \mathbb{R}^{n \times n \times c}$: $\forall (i,j),\ m_k(\mathbf{T})[i,j,:] = m_k(\mathbf{T}[i,j,:])$. Symbol $\|$ denotes concatenation along the third axis and $@$ matrix multiplication performed in parallel on each channel: $(\mathbf{T} \mathbin{@} \mathbf{T}')[:,:,c] = \mathbf{T}[:,:,c]\ \mathbf{T}'[:,:,c]$.

To simplify the presentation, we assume that:

- At each layer l, one of the channels of $\mathbf{T}^{(l)}$ corresponds to the adjacency matrix $\boldsymbol{A}$, another contains a matrix full of ones $\mathbf{1}_n \mathbf{1}_n^\top$ and a third the identity matrix $\boldsymbol{I}_n$, so that each PPGN layer has access at all times to these quantities. These matrices can be computed by the first PPGN layer and then kept throughout the computations using residual connections.
- The neural network can compute entry-wise multiplications $\odot$. This computation is not possible in the original model, but it can be approximated by a neural network.
- $\mathbf{U}$ and $\mathbf{T}$ have only one channel (so that we write them $\boldsymbol{U}$ and $\boldsymbol{T}$). This hypothesis is not necessary, but it will allow us to manipulate matrices instead of tensors.

Initialization Initially, we simply use the same input for PPGN as for SMP ($\boldsymbol{U}^{(0)} = \boldsymbol{T}^{(0)} = \boldsymbol{I}_n$).

Induction Assume that at layer l we have $\boldsymbol{U}^{(l)} = \boldsymbol{T}^{(l)}$. Consider a layer of Fast SMP:

$$\boldsymbol{U}_i^{(l+1)} = \frac{1}{d_{\text{avg}}} \left(\sum_{v_j \in N(v_i)} \hat{\boldsymbol{U}}_j^{(l)} \ + \ \hat{\boldsymbol{U}}_i^{(l)} \boldsymbol{W}_4^{(l)} \odot \sum_{v_j \in N(v_i)} \hat{\boldsymbol{U}}_j^{(l)} \boldsymbol{W}_5^{(l)} \right),$$

where

$$\hat{\boldsymbol{U}}_i^{(l)} = \boldsymbol{U}_i^{(l)}\ \boldsymbol{W}_1^{(l)} + \frac{1}{n} \mathbf{1}_n\ \mathbf{1}_n^T\ \boldsymbol{U}_i^{(l)}\ \boldsymbol{W}_2^{(l)} + \mathbf{1}_n (\boldsymbol{c}^{(l)})^\top + \frac{1}{n} \mathbb{1}_i \mathbf{1}^T \boldsymbol{U}_i^{(l)}\ \boldsymbol{W}_3^{(l)}.$$

A first PPGN block can be used to compute $\hat{\boldsymbol{U}}_i^{(l)}$ for each node. This block is parameterized by:

$$m_1(\boldsymbol{U}^{(l)}) = \frac{1}{n}\ \mathbf{1}_{\boldsymbol{n}}\ \mathbf{1}_{\boldsymbol{n}}^T, \qquad m_2(\boldsymbol{U}^{(l)}) = \boldsymbol{U}^{(l)},$$

$$m_3(\boldsymbol{U}^{(l)}) = \boldsymbol{U}^{(l)}\ \boldsymbol{W}_1 + 1\boldsymbol{c}^T + (\boldsymbol{I}_n\ \odot\ \boldsymbol{U}^{(l)})\ \boldsymbol{W}_3, \qquad m_4(\hat{\boldsymbol{U}},\ \tilde{\boldsymbol{U}}) = \hat{\boldsymbol{U}} + \tilde{\boldsymbol{U}}\ \boldsymbol{W}_2 + \boldsymbol{I}_n\ \odot\ (\tilde{\boldsymbol{U}}\ \boldsymbol{W}_3)$$

Table 3.1 – Time and space complexity of the forward pass expressed in terms of number of nodes n, number of edges m, number of node colors χ, and width c. For connected graphs, we trivially have $\chi \leq n \leq m + 1 \leq n^2$.

Method	Memory per layer	Time complexity per layer
GIN (Xu et al., 2019)	$\Theta(n\ c)$	$\Theta(m\ c + n\ c^2)$
MPNN (Gilmer et al., 2017)	$\Theta(n\ c)$	$\Theta(m\ c^2)$
Fast SMP (with coloring)	$\Theta(n\ \chi\ c)$	$\Theta(m\ \chi\ c + n\ \chi\ c^2)$
Fast SMP	$\Theta(n^2\ c)$	$\Theta(m\ n\ c + n^2\ c^2)$
SMP	$\Theta(n^2\ c)$	$\Theta(m\ n\ c^2)$
PPGN (Maron et al., 2019a)	$\Theta(n^2\ c)$	$\Theta(n^3\ c + n^2\ c^2)$
Local order-3 WL (Morris et al., 2019)	$\Theta(n^3\ c)$	$\Theta(n^4\ c + n^3\ c^2)$

The output of this block exactly corresponds to $\hat{\boldsymbol{U}}^{(l)}$. Then, a second PPGN block can be used to compute the rest of the Fast SMP layer. It should be parameterized as:

$$m_1([\hat{\boldsymbol{U}}^{(l)}]) = \boldsymbol{A} \,/\, \bar{d}, \qquad m_2([\hat{\boldsymbol{U}}^{(l)}]) = \hat{\boldsymbol{U}}^{(l)},$$
$$m_3([\hat{\boldsymbol{U}}^{(l)}]) = \hat{\boldsymbol{U}}^{(l)}, \qquad m_4(\hat{\boldsymbol{U}}^{(l)}, \tilde{\boldsymbol{U}}) = \tilde{\boldsymbol{U}} + (\hat{\boldsymbol{U}}^{(l)}\ \boldsymbol{W_4}) \odot (\tilde{\boldsymbol{U}}\ \boldsymbol{W_5})$$

By plugging these expressions into the definition of a PPGN block, we obtain that the output of this block corresponds to $\boldsymbol{U}^{(l+1)}$ as desired. □

Despite not being more powerful, Fast SMP has the advantage of being more efficient than PPGN, as it can exploit the sparsity of adjacency matrices. Furthermore, as we will see experimentally, our method manages to learn topological information much more easily than PPGN, a property that we attribute to the inductive bias carried by message-passing.

Complexity Table 3.1 compares the per-layer space and time complexity induced by the forward pass of SMP with that of other standard graph networks. Whereas local order-3 Weisfeiler-Lehman networks need to store all triplets of nodes, both PPGN and SMP only store information for pairs of nodes. However, message-passing architectures (such as SMP) can leverage the sparsity of the adjacency matrix and hence benefit from a more favorable time complexity than architectures which perform global updates (as PPGN).

An apparent drawback of SMP (shared by all equivariant powerful architectures we are aware of) is the need for more memory than MPNN. This difference is partially misleading since it is known that the width of any MPNN needs to grow at least linearly with n (for any constant depth) for it to be able to solve many graph-theoretic problems (Loukas, 2020a; Corso et al., 2020; Loukas, 2020b). However, for graphs with a large diameter, the memory requirements of SMP can be relaxed by using the following observation: *if each node is colored differently from all nodes in its $2k$-hop neighborhood, then no node will see the same color twice in its k-hop neighborhood*. It implies that nodes which are far apart can use the same identifier without conflict. We propose below a procedure (Fast SMP with coloring) based on greedy coloring which can replace the

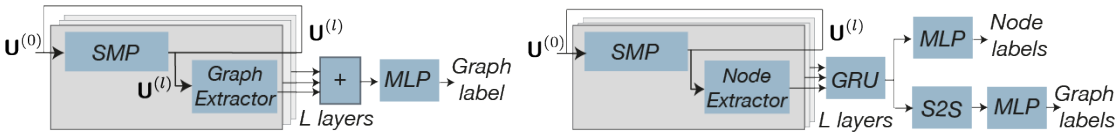


Figure 3.3 – (left) Architecture for cycle detection. The graph extractor computes the trace and the sum along the two first axes of $\mathbf{U}$, and passes the result into a two-layer MLP in order to produce a set of global features. (right) Architecture for multi-task learning: after each convolution, node features are extracted using a two-layer MLP followed by three pooling methods (mean, max, and the extraction of $\mathbf{U}[i, i, :]$ for each $v_i \in V$), and a final linear layer. The rest of the architecture is similar to Corso et al. (2020): it uses a Gated Recurrent Unit (GRU) and a Set-to-set network (S2S).

initial one-hot encoding, so that each node can manipulate smaller matrices $\boldsymbol{U}_i$. This method allows to theoretically improve both the time and space complexity of SMP, although the number of colors needed usually grows fast with the number of layers in the network.

Fast SMP with coloring In SMP, the initial local context is a one-hot encoding of each node: $\boldsymbol{U}_i^{(0)} = \boldsymbol{\delta}_i \in \mathbb{R}^n$. When the graph diameter Δ is large compared to the number of layers L, the memory requirements of this one-hot encoding can be reduced by attributing the same identifiers to nodes that are far away from each other. In particular, no node should see twice the same identifier in its L-hop neighborhood. To do so, we propose to build a graph G' where all $2L$-hop neighbors of G are connected, and to perform a greedy coloring of G' (Algorithm 1). Although the number of colors χ used by the greedy coloring might not be optimal, this procedure guarantees that identifiers do not conflict.

Algorithm 1: Node coloring

Input: A graph $G = (V, E)$ with n nodes, $L \in \mathbb{N}$ (number of layers.)
Output: A binary matrix $\boldsymbol{U}_i^0 \in \mathbb{R}^{n \times \chi}$, where χ is the number of colors.
Create the graph $G' = (V,\ \{(i, j), d(i, j\}) \leq 2L)$
$\boldsymbol{c} \in \mathbb{R}^n \leftarrow$ *greedy_coloring*(G')
return *one_hot_encoding*$(\boldsymbol{c})$

The one-hot encoding of the colors $\boldsymbol{U}_i^0 \in \mathbb{R}^\chi$ is then used to initialize the local context of v_i. The only change in the SMP network is that in order to update the representation that node i has of node j, we now update $\boldsymbol{U}_i[c_j, :]$ instead of $\boldsymbol{U}_i[j, :]$, where c_j is the color associated to node v_j. Note however that the coloring is only useful if the graph has a diameter $\Delta > 2L$. This is usually the case in geometric graphs such as meshes, but often not in scale-free networks.

Table 3.2 – Experiments on cycle detection, viewed as a graph classification problem.

(a) Test accuracy on the detection of cycles of various length with 10,000 training samples. (Best seen in color.) Only SMP solves the problem in all configurations.

Cycle length	4				6				8			
Graph size	12	20	28	36	20	31	42	56	28	50	66	72
MPNN	98.5	93.2	91.8	86.7	98.7	95.5	92.9	88.0	98.0	96.3	92.5	89.1
GIN	98.3	97.1	95.0	93.0	99.5	97.2	95.1	92.7	98.5	98.8	90.8	92.5
GIN + degree	99.3	98.2	97.3	96.7	99.2	97.1	97.1	94.5	99.3	98.7		95.4
GIN + rand id	99.0	96.2	94.9	88.3	99.0	97.8	95.1	96.1	98.6	98.0	97.2	95.3
RP Murphy et al. (2019)	100	99.9	99.7	97.7	99.0	97.4	92.1	84.1	99.2	97.1	92.8	80.6
PPGN	100	100	100	99.8	98.3	99.4	93.8	87.1	99.9	98.7	84.4	76.5
Ring-GNN	100	99.9	99.9	99.9	100	100	100	100	99.1	99.8	74.4	71.4
SMP	100	100	100	100	100	100	100	100	100	100	100	99.9

(b) Test accuracy (%) when evaluating the generalization ability of inductive networks. Each network is trained on one graph size ("In-distribution"), validated on a second size, then tested on a third ("Out-of-distribution"). SMP is the only powerful network evaluated that generalizes well. *OOM* = out of memory.

Setting	In-distribution			Out-of-distribution		
Cycle length	4	6	8	4	6	8
Graph size	20	31	50	36	56	72
GIN	93.9	99.7	98.8	81.1	85.8	**88.8**
PPGN	99.9	99.5	98.7	50.0	50.0	50.0
Ring-GNN	100	100	99.9	50.0	50.0	*OOM*
SMP	100	99.8	99.5	**99.8**	**87.8**	79.5

(c) Test accuracy (%) on the detection of 6 cycles for graphs with 56 nodes trained on less data. Thanks to its equivariance properties, SMP requires much less data for training.

Train samples	200	500	1000	5000
GIN + random identifiers	65.8	70.8	80.6	96.4
SMP	**87.7**	**97.4**	**97.6**	**99.5**

3.5 Experiments

Cycle detection

We first evaluate different architectures on the detection of cycles of length 4, 6 and 8 (for several graph sizes), implemented as a graph classification problem[1]. Models are retrained for each cycle length and graph size on 10k samples with balanced classes, and evaluated on $10,000$ samples as well. The same architecture (detailed in Figure 3.3) is used for all models, as we found it to perform better than the original implementation of each method: the methods under comparison thus only differ in the definition of the convolution, making comparison easy. We use the fast implementation of SMP, as we find its expressivity to be sufficient for this task.

[1]Our implementation with PyTorch Geometric (Fey & Lenssen, 2019) is available at `github.com/cvignac/SMP`.

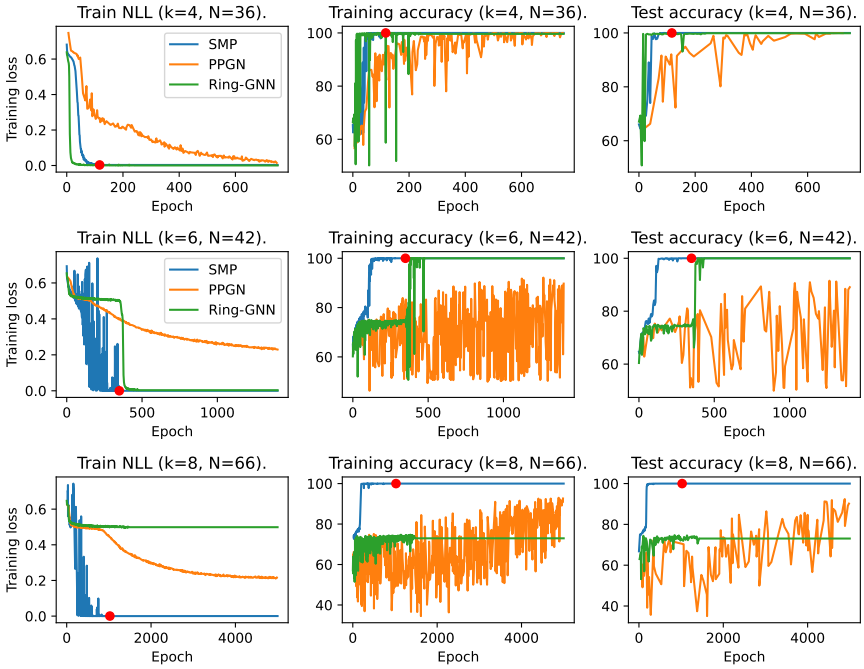


Figure 3.4 – Training curves of SMP, PPGN and Ring-GNN for different cycle lengths k. NLL stands for negative log-likelihood. Red dots indicate the epoch when SMP training was stopped. The training loss sometimes exhibits peaks of very high value which last one epoch – they were removed for readability. Provably powerful graph networks are much more difficult to train than SMP: their failure is not due to a poor generalization, but to the difficulty of optimizing them. Ring-GNN works well for small graphs, but we did not manage to train it with the largest graphs (66 or 72 nodes). We attribute this phenomenon to an inductive bias that is less suited to the task. PPGN and SMP training time per epoch are approximately the same, while RING-GNN is between two and three times slower.

Results are shown in Tab 3.2. For a given cycle length, the task becomes harder as the number of nodes in the graph grows: the bigger the graph, the more candidate paths that the network needs to verify as being cycles. SMP is able to solve the task almost perfectly for all graph and cycle sizes. For standard message-passing models, we observe a correlation between accuracy and the presence of identifiers: random identifiers and weak identifiers (a one-hot encoding of the degree) tend to perform better than the baseline GIN and MPNN. PPGN and RING-GNN solve the task well for small graphs, but fail when n grows. Perhaps due to a miss-aligned inductive bias, we encountered difficulties with training them, whereas message-passing architectures could be trained more easily. We provide a more detailed comparison between SMP, PPGN and Ring-GNN in Figure 3.4. We also compare the generalization ability of the different networks that can be used in inductive settings. GIN generalizes well, but SMP is the only one that achieves good performance among the powerful networks. This may be imputable to the inductive bias of message passing architectures, shared by GIN and SMP. Finally, we compare SMP and GIN with random identifiers in settings with less training data: SMP requires much fewer samples

Table 3.3 – Log MSE on the test set (lower is better). Baseline results are from Corso et al. (2020).

Model	*Average*	Dist.	Ecc.	Lap.	Conn.	Diam.	Rad.
GIN	-1.99	-2.00	-1.90	-1.60	-1.61	-2.17	-2.66
GAT	-2.26	-2.34	-2.09	-1.60	-2.44	-2.40	-2.70
MPNN (sum)	-2.53	-2.36	-2.16	-2.59	-2.54	-2.67	-2.87
PNA	-3.13	-2.89	-2.89	-3.77	-2.61	-3.04	-3.57
Fast MPNN (Ablation)	-2.37	-2.47	-1.99	-2.83	-1.61	-2.40	-2.93
MPNN (Ablation)	-2.77	-3.18	-2.05	-3.27	-2.24	-2.88	-2.97
Fast SMP	-3.53	-3.31	-3.36	$\mathbf{-4.30}$	-2.72	$\mathbf{-3.65}$	$\mathbf{-3.82}$
SMP	$\mathbf{-3.59}$	$\mathbf{-3.59}$	$\mathbf{-3.67}$	-4.27	$\mathbf{-2.97}$	-3.58	-3.46

to achieve good performance, which confirms that equivariance is important for good sample efficiency.

Multi-task detection of graph properties

We further benchmark SMP on the multi-task detection of graph properties proposed in Corso et al. (2020). The goal is to estimate three node-defined targets: *geodesic distance* from a given node (Dist.), *node eccentricity* (Ecc.), and computation of *Laplacian features* $\boldsymbol{Lx}$ given a vector $\boldsymbol{x}$ (Lap.), as well as three graph-defined targets: *connectivity* (Conn.), *graph diameter* (Diam.), and *spectral radius* (Rad.). The training set is composed of 5120 graphs with up to 24 nodes, while graphs in the test set have up to 19 nodes. Several MPNNs are evaluated as well as PNA (Corso et al., 2020), a message-passing model based on the combination of several aggregators. Importantly, random identifiers are used for all these models, so that all baseline methods are theoretically powerful (Loukas, 2020a), but not equivariant.

All models are benchmarked using the same architecture, apart from the fact that SMP manipulates local contexts. In order to pool these contexts into node features and use them as input to the Gated Recurrent Unit (Cho et al., 2014), we use an extractor described in Figure 3.3. As an ablation study, we also consider for each model a corresponding MPNN with the same architecture.

The results are summarized in Table 3.3. We find that both SMPs are able to exploit the local contexts, as they perform much better than the corresponding MPNN. SMP also outperforms other methods by a significant margin. Lastly, standard SMP tends to achieve better results than fast SMP on tasks that require graph traversals (shortest path computations, eccentricity, checking connectivity), which may be due to a better representation power.

Constrained solubility regression on ZINC

The ZINC database is a large scale dataset containing molecules with up to 37 atoms. The task is to predict the constrained solubility of each molecule, which can be seen as a graph

Table 3.4 – Mean absolute error (MAE) on ZINC, trained on a subset of 10k molecules.

Model	No edge features	With edge features
Gated-GCN Fey et al. (2020)	0.435	0.282
GIN Fey et al. (2020)	0.408	0.252
PNA Corso et al. (2020)	0.320	0.188
DGN Beaini et al. (2021)	**0.219**	0.168
MPNN-JT Fey et al. (2020)	–	0.151
MPNN (ablation)	0.272	0.189
SMP (Ours)	**0.219**	**0.138**

regression problem. We follow the setting of Dwivedi et al. (2023): we train SMP on the same subset of 10,000 molecules with a parameter budget of around 100k, reduce the learning rate when validation loss stagnates, and stop training when it reaches a predefined value. We use an expressive parametrization of SMP, with 12 layers and 2-layer MLPs both in the message and the update functions. In order to reduce the number of parameters, we share the same feature extractor after each layer (cf Fig. 3.3). Results are presented in Table 3.4. They show that in both cases (with or without edge features, which are a one-hot encoding of the bond type), SMP is able to achieve state of the art performance. Note however than even better results (0.108 MAE using a MPNN with edge features (Bouritsas et al., 2022)) can be achieved by augmenting the input with expert features. We did not use them in order to compare fairly with the baseline results.

3.6 Discussion

We now discuss open problems and advances in powerful graph neural networks that are subsequent to the publication of this chapter:

Expressivity A natural question is whether SMP, which is strictly more powerful than the 2-WL test, can be as expressive as the 3-WL test which considers triplets of nodes. To show this, one would like to establish the equivalence between SMP and the Folklore 2-WL test, which is as powerful as 3-WL but only manipulates tensors of size $n \times n \times d$ (and not triplets of nodes). However, proving or disproving this equivalence is not straightforward due to the differences in the definition of the neighborhood in the two tests. Specifically, in the Folklore 2-WL test, propagation happens simultaneously on the rows and the columns of the tensor $\mathbf{U} \in \mathbb{R}^{n \times n \times d}$ at each message-passing update, whereas in each SMP layer, message-passing propagation occurs only on the rows, while the columns are updated in the message function using a more affordable neural network for sets. It remains an open question whether this strategy is sufficient to achieve the expressive power of the Folklore 2-WL test.

Empirically, we also observed that SMP cannot distinguish between certain graphs that are indistinguishable by the 3-WL test (Bouritsas et al., 2022). This is unsurprising given its parametrization, as no message-passing architecture that manipulates $n \times n \times d$ embeddings is known to be superior to the Folklore 2-WL test.

Expressive graph neural networks Since the publication of our work, many other powerful graph neural networks have been proposed. Various strategies have been adopted, including the use of message-passing on larger substructures than one-hop neighborhood (de Haan et al., 2020), and neural networks that operate on bag-of-subgraphs (Bevilacqua et al., 2021; Frasca et al., 2022). However, these methods seem to suffer from similar trade-offs: networks that achieve high expressive power tend to be computationally more costly or introduce randomness, making them harder to train.

Currently, powerful neural networks find their main applications in the prediction of quantum properties of molecules. Expressive architectures are particularly suited for these tasks that feature very large datasets and where high accuracy is required. However, atomic coordinates are often available in molecular datasets, and the architectures used in practice leverage not only graph structures but also (potentially high-order) representations of the SE(3) group (Joshi et al., 2023).

For purely graph-based tasks, positional encoding (Beaini et al., 2021; Dwivedi et al., 2021) has become a popular alternative to the use of provably powerful networks. These methods typically cannot prove equivalence to higher-order WL tests, but they allow for improved expressive power with a moderate overhead.

Random graph analysis An interesting application of SMP is the analysis of graph networks in the limit of infinitely many nodes (Keriven et al., 2020; 2021). In the case of MPNNs, over-smoothing occurs when the graph and the number of layers grow, and the limit representation is not particularly interesting. In contrast, the one-hot encoding of the nodes in SMP enables the embeddings to scale with the graph size, which leads to more insightful analysis. Keriven et al. (2021) has in particular shown that the SMP model is universal on some random graph models such as the stochastic block model.

3.7 Conclusion

In this chapter, we introduced structural message-passing (SMP), a new architecture that is both powerful and permutation equivariant, solving a major weakness of previous message-passing networks. Empirically, SMP significantly outperforms previous models in learning graph structural properties, but retains the inductive bias of MPNNs and their good ability to process node features. We believe that our work paves the way to graph neural networks that efficiently manipulate both node and topological features, with potential applications to chemistry, computational biology and neural algorithmic reasoning.

4 Set and graph generation from a latent vector

4.1 Introduction

In the preceding chapters, we have established that message-passing neural networks can be viewed as local permutation equivariant networks. Through their iterative propagation scheme, these networks have a prior towards locality, which is effective on many tasks. However, the limited expressive power of MPNNs remains a key limitation. We have demonstrated that this constraint is not inherent to the message-passing scheme itself, but instead arises due to the difficulty of learning rich permutation equivariant functions of graphs. To address this issue, we have proposed a more powerful architecture that retains the inductive bias of message-passing, and that outperforms other high-capacity architectures. Overall, we can therefore consider that the implications of permutation equivariance on graph representation learning are relatively well understood. While it is not clear how to build equivariant and powerful graph neural networks that maintain the linear computational complexity of MPNNs, we know that universal function approximation on graphs is as hard as isomorphism testing, and there is little hope to obtain universal models without additional assumptions on the data.

We now extend our investigation to the domain of set and graph generation, exploring the implications of permutation equivariance in this context. On generative tasks, very different architectures can be used. In particular, one can consider using a latent space that contains vectors, that contains sets, or that contains latent graphs. In this chapter, we consider probabilistic decoders that map a low-dimensional vector prior to graphs. This architecture is probably the most natural, as it factors out permutation equivariance by representing each set or graph as one single vector.

Set and graph generation are very similar problems, and one can navigate between set and graph representations using Set2Graph functions (Serviansky et al., 2020) or graph neural networks. We therefore generally treat them together in this paper, but only use the terminology of sets to avoid overly abstract notations. The distinctive properties of graph generation (e.g., graph matching problems) are discussed when needed.

There are two main classes of probabilistic decoders for sets: recursive and one-shot. Recursive generators are conceptually simpler, since they add points one by one (Liu et al., 2018; Liao et al., 2019; Sun et al., 2020b; Nash et al., 2020). They are however slow and introduce an order in the points that does not exist in the data. In this work, we instead focus on one-shot generation, which allows for more principled designs. By definition, one-shot models feature a layer that maps a vector to an initial set. We group all layers until this one into a *creation module*, and the subsequent layers into an *update module*. The design of the update is well understood: as it maps a set to another set, any permutation equivariant network suits this task. In contrast, the creation step poses two major challenges, which are i) the need to generate sets of various sizes, and ii) the generation of different points from a single prior vector, which cannot be done with a permutation equivariant function.

In more details, set creation is typically performed by first sampling points independently from a normal distribution, and then appending the latent vector to each point. This design allows the generation of any number of points and decouples the set cardinality from the number of trainable parameters. Furthermore, it generates exchangeable distributions (all permutations of a set are equally likely), a property which is commonly held as the equivalent of equivariance for generative models. However, it was empirically observed that VAEs based on independent sampling are hard to train, which results in limited performance (Krawczuk et al., 2021).

We first propose a theoretical argument to this empirical observation, by showing that VAEs that use independent sampling only optimize a proxy to the evidence lower bound (ELBO). As the standard definition of equivariance cannot be used in generative settings, we then propose a generalization of this notion called (F, l)-equivariance: informally, an architecture is (F, l)-equivariant if the parameter updates do not depend on the group elements used to represent training data. We derive sufficient conditions for equivariance in this setting. They reveal that (F, l)-equivariance explains the loss functions commonly used both in generative and discriminative tasks, and suggest that exchangeability may not be useful in GANs and VAEs.

Based on these results, we finally propose a non-exchangeable set creation method called *Top-n creation*. Our method relies on a trainable reference set where each point i has a *representation* $\boldsymbol{r}_i \in \mathbb{R}^c$ and an *angle* $\boldsymbol{\phi}_i \in \mathbb{R}^a$. To generate a set with n elements, we select the n points whose angles have the largest cosine with the latent vector. In order to make this process differentiable, we build upon the Top-K pooling mechanism initially proposed for graph coarsening (Gao & Ji, 2019). Top-n can eventually be integrated in any VAE or GAN to form a complete generative model for sets or graphs. Our method is easier to train than stochastic generators, and has better generalization performance than other existing methods.

We benchmark Top-n on both set and graph generation tasks: it is able to reconstruct the data more accurately on a set version of MNIST, generalize better on the CLEVR object detection dataset, fit more closely the true distribution on a dataset of synthetic molecule-like structures in 3D, and generate realistic molecular graphs on QM9.

4.2 The one-shot set generation problem

We consider the problem of learning a probabilistic decoder f that maps latent vectors $\boldsymbol{z} \in \mathbb{R}^l$ to multi-sets[1] $\mathcal{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ that contain a varying number n of points $\boldsymbol{x}_i$ in $\mathbb{R}^d$. Given sample sets from an unknown distribution $\mathcal{D}$, f should be such that, if $\boldsymbol{z}$ is drawn from a prior distribution $p_Z(\boldsymbol{z})$, then the push-forward measure $f_\#(p_Z)$ (i.e., the law of $f(\boldsymbol{z})$) is close to $\mathcal{D}$. In practice, representing sets is not convenient on standard hardware, and sets are internally represented by matrices $\boldsymbol{X} \in \mathbb{X} = \bigcup_{n \in \mathbb{N}} \mathbb{R}^{n \times d}$ where each row represents a point $\boldsymbol{x}_i \in \mathbb{R}^d$. Algorithms that return a set implicitly assume the use of a function mat-to-set that maps $\boldsymbol{X}$ to the corresponding set $\mathcal{X}$.

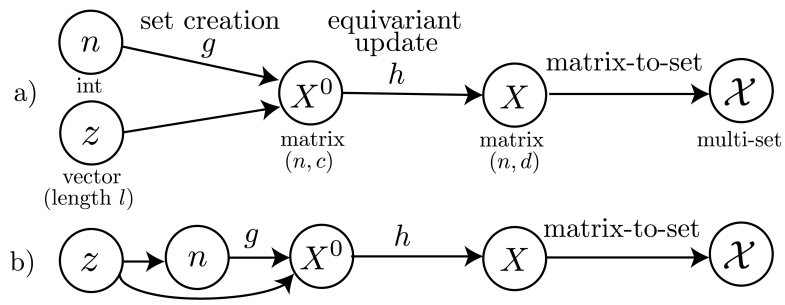


Figure 4.1 – The graphical models for set generation. The number of points can either be sampled from the dataset distribution (a), or learned from the latent vector (b). While any equivariant function can be used for the update h, the set creation g concentrates the challenges of set generation. For graph generation, edge weights are generated in addition to the node features matrices $\boldsymbol{X}^0$ and $\boldsymbol{X}$.

Existing architectures for one-shot generation use one of the graphical models described in Figure 4.1. First, a number of points for the set has to be sampled. Most works assume that the set cardinalities are known during training. At generation time, they sample n from the distribution of set cardinalities in the training data. This method assumes that the latent vector $\boldsymbol{z}$ is independent of the number of points n, so that the generative mechanism writes $p(\mathcal{X}|n, \boldsymbol{z})\ p(n)\ p(\boldsymbol{z})$.

Kosiorek & Kim (2020) instead propose to learn the value of n from the latent vector using a MLP. This layer is trained via an auxiliary loss, but the predicted value is used only at generation time (the ground truth cardinality is used during training). The generative model is in this case $p(\mathcal{X}|\boldsymbol{z}, n)\ p(n|\boldsymbol{z})\ p(\boldsymbol{z})$, i.e., $\boldsymbol{z}$ and n are not independent anymore.

Once n is sampled, one-shot generation models can formally be decomposed into several components: a first function g (that we call *creation function*) maps the latent vector to an initial set $\boldsymbol{X}^0 \in \mathbb{R}^{n \times c}$. This function is usually simple, and is therefore not able to model complex dependencies within each set. For this reason, $\boldsymbol{X}^0$ may then be refined by a second function h that we call *update*, so that the whole model can be written $f = \text{mat-to-set} \circ h \circ g$. We now review the parametrizations that have been proposed for the creation and update modules.

[1]For the sake of simplicity, we will refer to *sets* instead of *multi-sets* in the rest of the chapter.

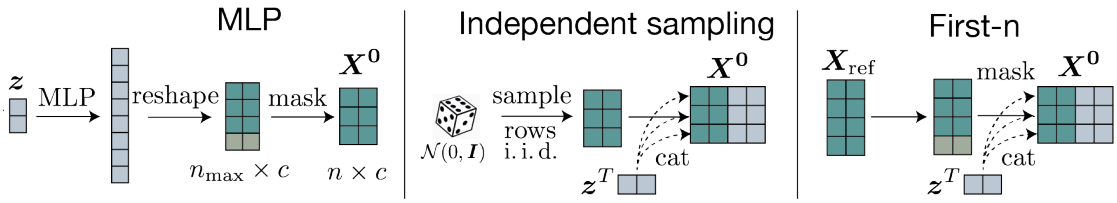


Figure 4.2 – Existing creation methods for mapping a latent vector $\boldsymbol{z}$ to a set of points $\boldsymbol{X}^0$. First-n creation empirically gives the best performance. It learns a reference set represented by a matrix $\boldsymbol{X}_{\text{ref}}$, and concatenates the latent vector to each point of this set.

4.2.1 Methods for set creation

MLP based

Many existing methods (Achlioptas et al. (2018); Zhang et al. (2020; 2021b) for sets, Guarino et al. (2017); De Cao & Kipf (2018); Simonovsky & Komodakis (2018) for graphs) learn a MLP from $\mathbb{R}^l$ to $\mathbb{R}^{n_{\max} c}$, where $n_{\max}$ is the largest set size in the training data. The output vector is then reshaped as a $n_{\max} \times c$ matrix, and masked to keep only the first n rows (Figure 4.2). MLPs ignore the symmetries of the problem and are only trained to generate up to $n_{\max}$ points, with no ability to extrapolate to larger sets. Despite these limitations, they often perform on par with more complex methods for small graphs (Madhawa et al., 2019; Mitton et al., 2021). We explain in Section 4.3 this surprising phenomenon.

Independent sampling

Along with MLP based generation, the most popular method for set and graph creation is to draw n points i.i.d. from a low dimensional normal distribution, and to concatenate the latent vector to each sample (Köhler et al., 2020; Yang et al., 2019b; Kosiorek & Kim, 2020; Stelzner et al., 2020; Satorras et al., 2021b; Zhang et al., 2021a; Liu et al., 2021). The main advantage of independent sampling is that it does not constrain the number of points that can be generated. Furthermore, it is exchangeable, i.e., all permutations of the rows of $\boldsymbol{X}^0$ are equally likely. This property is widely considered as the equivalent of equivariance for generative models (Yang et al., 2019a; Biloš & Günnemann, 2021; Kim et al., 2021; Köhler et al., 2020; Li et al., 2020).

However, it was empirically observed that VAEs built with a i.i.d. creation mechanism fail to fit the training data correctly (Krawczuk et al., 2021), which reflects in the poor quality of the sampled sets. To understand why, consider a VAE made of an encoder $q_\phi(\boldsymbol{z}|\boldsymbol{X})$ and a decoder $f_\theta(\boldsymbol{X}|\boldsymbol{z})$ parametrized by ϕ and θ. Variational autoencoders maximize the evidence lower bound (ELBO) $\mathcal{L}$, which is a proxy for the data likelihood under the model:

$$\mathcal{L}(\boldsymbol{X}) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{X})}[\log p_\theta(\boldsymbol{X}, \boldsymbol{z}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{X})] \leq p_\theta(\boldsymbol{X}) \tag{4.1}$$

Probabilistic decoders $f_\theta(\boldsymbol{z})$ based on independent sampling are stochastic, so that $\log p_\theta(\boldsymbol{X}, \boldsymbol{z})$ cannot be computed in close form. By conditioning on the initial set $\boldsymbol{X}^0$ and using Jensen's

inequality, we have:

$$\log p_\theta(\boldsymbol{X}, \boldsymbol{z}) = \log \mathbb{E}_{\boldsymbol{X}^0 \sim p(\boldsymbol{X}^0)}\, p_\theta(\boldsymbol{X}, \boldsymbol{z}|\boldsymbol{X}^0) \geq \mathbb{E}_{\boldsymbol{X}^0 \sim p(\boldsymbol{X}^0)} \log p_\theta(\boldsymbol{X}, \boldsymbol{z}|\boldsymbol{X}^0) \tag{4.2}$$

which gives in expectation

$$\mathcal{L}(\boldsymbol{X}) \geq \mathbb{E}_{\boldsymbol{X}^0, \boldsymbol{z}} \left[\log p_\theta(\boldsymbol{X}, \boldsymbol{z}|\boldsymbol{X}^0) - \log q_\phi(\boldsymbol{z}|\boldsymbol{X})\right] := \mathcal{L}'(\boldsymbol{X}). \tag{4.3}$$

Methods based on independent sampling use a Monte-Carlo estimate for $\nabla_{\theta,\phi}\mathcal{L}'(\boldsymbol{X})$ that leverages the reparametrization trick. They therefore only optimize a lower bound of the ELBO, which could explain why they are difficult to train.

First-n

Instead of sampling points, Zhang et al. (2019) and Krawczuk et al. (2021) propose to always start from the same learnable set $\boldsymbol{X}_{\text{ref}} \in \mathbb{R}^{n_{\max} \times c}$, and mask this matrix to keep only the first n rows: we therefore call this method First-n creation. Similarly to the independent sampling method, the latent vector is then concatenated to each point of this set.

Empirically, First-n converges much faster than sampling-based methods (Krawczuk et al., 2021), but the network is only trained to generate up to $n_{\max}$ points and has no ability to extrapolate to larger sets. Furthermore, selecting the first n rows of the reference set $\boldsymbol{X}_{\text{ref}}$ introduces a bias because the first rows are selected more often than the last ones.

Creation methods for graph generation

For graph generation, edge weights (or edge features) also need to be learned. To the best of our knowledge, First-n creation has not been used for this purpose yet, and the weights are generated either by a MLP or by sampling normal entries i.i.d. An alternative is to first generate a set, and then use a Set2Graph update function in order to learn the graph adjacency matrix as in (Bresson & Laurent, 2019; Krawczuk et al., 2021).

4.2.2 Methods for set update

Since all creation methods except MLPs can only generate very simple sets and adjacency matrices, additional layers are usually used to refine these objects – we gather these layers into the *update* module. Because these layers map a set or a graph to another set/graph, the update module falls into the standard framework of permutation equivariant representation learning and all existing equivariant layers can be used: Deep sets (Zaheer et al., 2017), self-attention (Vaswani et al., 2017; Lee et al., 2019), Set2Graph (Serviansky et al., 2020), graph neural networks (Battaglia et al., 2018) or higher-order neural networks (Morris et al., 2019). Recently, Transformer layers have constituted the most popular method for set and graph update (Bresson & Laurent, 2019; Kosiorek & Kim, 2020; Stelzner et al., 2020; Krawczuk et al., 2021) – we also use such layers in our experiments.

4.3 A permutation equivariance view on set generation

Whereas exchangeability is usually considered as a key feature of independent sampling, we have seen that this method empirically does not outperform other strategies. To understand why, we need to study equivariance in generative models and propose a relevant definition in this setting.

As set and graphs are unordered, the symmetric group $\mathbb{S} = \bigcup_{n \in \mathbb{N}^*} \mathbb{S}_n$ containing all permutations is a symmetry of these tasks. A permutation $\pi \in \mathbb{S}_n$ acts on a $n \times n$ matrix $\boldsymbol{A}$ by permuting its rows and columns (which we write $\pi.\boldsymbol{A} = \pi\ \boldsymbol{A}\ \pi^T$), on a $n \times c$ matrix by permuting its rows ($\pi.\boldsymbol{X} = \pi\boldsymbol{X}$), and leaves a vector $\boldsymbol{z} \in \mathbb{R}^h$ unchanged ($\pi.\boldsymbol{z} = \boldsymbol{z}$). This symmetry constitutes a useless factor of variation in the data that should be factored out in the latent space (i.e., $\pi.\boldsymbol{z} = \boldsymbol{z}$).

In discriminative models, symmetries are accounted for when a neural network f is equivariant to the action of a group, which writes $\pi.f(\boldsymbol{X}) = f(\pi.\boldsymbol{X})$ (Kondor, 2008). When the input of f is a vector, imposing $\pi.f(\boldsymbol{z}) = f(\pi.\boldsymbol{z}) = f(\boldsymbol{z})$ however only allows for solutions where all rows are equal, which is too restrictive. To solve this issue, we propose a definition called (F, l)-equivariance which generalizes the common one, but provides more relaxed conditions in generative settings.

Our proposition is based on the assumption that the main role of equivariance is to make data augmentation useless. In discriminative settings, this is normally done by combining an equivariant model with an invariant loss function. For example, the l_2 loss is commonly used to learn the future state of a n-body system, but not the l_1 loss, as it is not rotation invariant. Formally, if $F_\Theta = \{f_\theta : \mathbb{X} \to \mathbb{Y};\ \theta \in \Theta\}$ is an hypothesis class of $\mathbb{G}$-equivariant functions from $\mathbb{X}$ to $\mathbb{Y}$ (for example a neural architecture parameterized by θ), then the loss functions l should satisfy

$$\forall f \in F,\ \forall g \in \mathbb{G},\ \forall (\boldsymbol{X}, \boldsymbol{Y}) \in \mathbb{X} \times \mathbb{Y}, \qquad l(g.f(\boldsymbol{X}), g.\boldsymbol{Y}) = l(f(\boldsymbol{X}), \boldsymbol{Y}) \tag{4.4}$$

Furthermore, we observe that when l satisfies Eq. 4.4, the gradients with respect to the parameters satisfy $\nabla_\theta\, l(f(g.\boldsymbol{X}), g.\boldsymbol{Y}) = \nabla_\theta\, l(f(\boldsymbol{X}), \boldsymbol{Y})$, i.e., each parameter update is independent of the group elements that are used to represent $\boldsymbol{X}$ and $\boldsymbol{Y}$. It follows that the training dynamics as a whole become independent of the group elements used to represent the data. We propose to use this property to define equivariance:

Definition 1 ((F, l)-equivariance)**.** Consider an hypothesis class $F_\Theta \subset \mathbb{Y}^{\mathbb{X}}$, a group $\mathbb{G}$ that acts on $\mathbb{X}$ and $\mathbb{Y}$ and a loss function l defined on $\mathbb{Y}$. We say that the pair (F_Θ, l) is equivariant to the action of $\mathbb{G}$ if the dynamics of $\theta \in \Theta$ trained with gradient descent on l do not depend on the group elements that are used to represent the training data.

By construction, using an equivariant architecture and an invariant loss is sufficient for (F, l)-equivariance in discriminative settings. For standard generative architectures for sets and graphs, we derive the following sufficient conditions:

Lemma 4 (Sufficient conditions for (F, l)-equivariance)**.** *1. GANs: if F is a GAN architecture with a permutation invariant discriminator, and l the standard GAN loss, then (F, l) is permutation equivariant. No constraint is imposed on the generator.*

2. VAEs: if F is an encoder-decoder architecture with a permutation invariant encoder, and the reconstruction loss l satisfies $\forall \pi \in \mathbb{S}, l(\pi.\boldsymbol{X}, \hat{\boldsymbol{X}}) = l(\boldsymbol{X}, \hat{\boldsymbol{X}})$, then (F, l) is permutation equivariant. No constraint is imposed on the decoder function.

3. Normalizing flows: if F is an architecture such that the set creation yields an exchangeable distribution, the update is permutation equivariant and invertible, and p_θ denotes the model likelihood, then $(F, -\log p_\theta)$ is permutation equivariant (proved in Köhler et al. (2020)).

Proof. **Generative adversarial networks** Given a set generator f, a discriminator function d, and $\boldsymbol{X}_1, ..., \boldsymbol{X}_m$ a training dataset, the standard loss function for GANs is formulated as

$$l(f, d, \boldsymbol{X}_1, ..., \boldsymbol{X}_m) = \frac{1}{m} \sum_{i=1}^{m} \log(d(\boldsymbol{X}_i))] + \mathbb{E}_{\mathbb{Z}}[\log(1 - d(f(\boldsymbol{z}))).$$

In order to obtain $l(f, d, \boldsymbol{X}_1, ..., \boldsymbol{X}_m) = l(f, d, \pi_1.\boldsymbol{X}_1, ..., \pi_m.\boldsymbol{X}_m)$ for every choice of π_i, it is therefore sufficient to choose a permutation invariant discriminator.

Auto-encoder based models We assume that the autoencoder is made of a permutation invariant encoder *enc* and an arbitrary decoder f. For any set size n, set $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and permutation $\pi \in \mathbb{S}_n$ we have

$$\begin{aligned} l(\pi.\boldsymbol{X}, \hat{\boldsymbol{X}}) = l(\boldsymbol{X}, \hat{\boldsymbol{X}}) &\implies l(\pi.\boldsymbol{X}, f(enc(\boldsymbol{X})) = l(\boldsymbol{X}, f(enc(\boldsymbol{X})) \\ &\implies l(\pi.\boldsymbol{X}, f(enc(\pi.\boldsymbol{X})) = l(\boldsymbol{X}, f(enc(\boldsymbol{X})) \quad (enc \text{ is invariant}) \\ &\implies \nabla_\theta\, l(\pi.\boldsymbol{X}, f(enc(\pi.\boldsymbol{X}))) = \nabla_\theta\, l(\boldsymbol{X}, f(enc(\boldsymbol{X}))) \end{aligned}$$

□

As desired, (F, l)-equivariance does not impose $\pi.f(\boldsymbol{z}) = f(\pi.\boldsymbol{z})$ in generative architectures, but still makes data augmentation unnecessary. Furthermore, the constraints of Lemma 4 are satisfied by most existing architectures, including early ones (Simonovsky & Komodakis, 2018; De Cao & Kipf, 2018; Köhler et al., 2020). In particular, the constraint on the loss for VAEs is satisfied by the two loss functions commonly used for sets, namely Chamfer loss and the Wasserstein-2 distance. Our definition, introduced by observing common practice in discriminative settings, is therefore able to explain common practice for generative tasks as well. Chamfer and Wasserstein-2 d are defined as:

$$d_{\text{Cham}} = \sum_{1 \le i \le n} \min_{j \le n'} ||\boldsymbol{x}_i - \boldsymbol{x}'_j||_2^2 + \sum_{1 \le j \le n'} \min_{i \le n} ||\boldsymbol{x}_i - \boldsymbol{x}'_j||_2^2$$

$$d_{\mathcal{W}_2} = \inf_{u \in \{\Gamma(\boldsymbol{X}, \boldsymbol{X}')\}} \sum_{\substack{1 \le i \le n \\ 1 \le j \le n'}} u(\boldsymbol{x}_i, \boldsymbol{x}'_j)\, ||\boldsymbol{x}_i - \boldsymbol{x}'_j||_2^2$$

where Γ is the set of couplings (i.e., bistochastic matrices) between $\boldsymbol{X}$ and $\boldsymbol{X}'$. Both loss functions solve a matching problem over the space of permutations, which makes them invariant to permutations of one argument, as required by Lemma 4. Chamfer's loss runs in quadratic time, while the standard implementation of Wasserstein distance runs in $O(n^3)$ if both sets have the same size. However, efficient algorithms exist for approximating the Wasserstein distance, and the computations can be parallelized on GPUs (Cuturi, 2013; Feydy et al., 2019). Note however that the equation $l(\pi.\boldsymbol{X}, \hat{\boldsymbol{X}}) = l(\boldsymbol{X}, \hat{\boldsymbol{X}})$ is may be difficult to satisfy in other settings: matching graphs up to permutations, or sets up to the $SE(3)$ group, leads to difficult problems for which no polynomial time algorithm is known (Mémoli, 2007). In these settings, the design of VAE is harder and other architectures may be better suited.

We finally observe that exchangeability does not appear in the sufficient conditions for GANs and VAEs. To understand why, recall that in GANs and VAEs a $\mathrm{mat\text{-}to\text{-}set}$ function is implicitly applied to the model output: a method that generates matrices that are always permuted in the same way is therefore equivalent to one that generates exchangeable matrices. In other words, the fact that the output of the model is not a matrix but a set is an assumption, not something that needs to be proved[2]. This observation explains why independent sampling creation does not outperform non-exchangeable set creation methods such as MLPs and First-n. In the following section, we therefore design a new creation mechanism without worrying about the model exchangeability.

4.4 The Top-n creation mechanism

We have seen in Section 4.2 that existing set creations methods suffer from important limitations: independent sampling makes it hard to train the model, while MLPs and First-n use a fixed mask to select the correct number of points and cannot extrapolate to larger sets. In order to solve these limitations, we propose a new method called Top-n creation, which is summarized in Figure 4.3.

Similarly to First-n, Top-n also uses a reference set, but in Top-n this set can have an arbitrary size n_0. Each point in this set is a pair $(\boldsymbol{\phi}, \boldsymbol{r})$: the *angle* $\boldsymbol{\phi} \in \mathbb{R}^a$ is used to decide when to select the point, and $\boldsymbol{r} \in \mathbb{R}^c$ contains the *representation* of the point. Given a latent vector $\boldsymbol{z} \in \mathbb{R}^{l \times 1}$, a reference set made of angles $\boldsymbol{\Phi} \in \mathbb{R}^{n_0 \times a}$ and representations $\boldsymbol{R} \in \mathbb{R}^{n_0 \times c}$, as well as learnable

[2]On the contrary, normalizing flows cannot use the non-invertible $\mathrm{mat\text{-}to\text{-}set}$ function, and typically compute probability distributions on the space of matrices rather than multisets. It is therefore natural that exchangeability appears for normalizing flows and not for the other architectures.

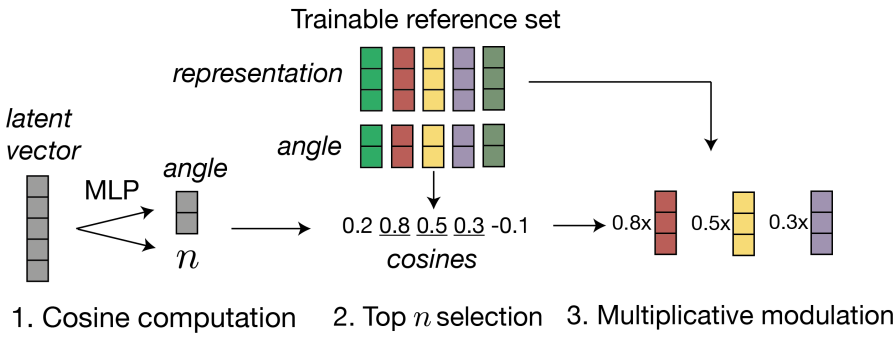


Figure 4.3 – Top-n creation learns to select the most relevant points in a trainable reference set based on the value of the latent vector. To obtain gradients and train the angles and the MLP despite the non-differentiable argsort operation, we modulate the selected representations with the values of the cosines – in practice, we use a FiLM layer (Perez et al., 2018) rather than multiplication.

matrices $\boldsymbol{W}_1$ to $\boldsymbol{W}_4$ (respectively of sizes $1 \times c,\ 1 \times c,\ l \times c,\ l \times c$), Top-n creation computes:

$$\boldsymbol{a} = \mathrm{MLP}_1(\boldsymbol{z}) \qquad \in \mathbb{R}^a \tag{4.5}$$

$$\boldsymbol{c} = \boldsymbol{\Phi}\ \boldsymbol{a}\ /\ \mathrm{vec}((||\boldsymbol{\phi}_i||_2)_{1 \leq i \leq n_0}) \qquad \in \mathbb{R}^{n_0} \tag{4.6}$$

$$\boldsymbol{s} = \mathrm{argsort}_{\downarrow}(\boldsymbol{c})[:n] \qquad \in \mathbb{N}^n \tag{4.7}$$

$$\tilde{\boldsymbol{c}} = \mathrm{softmax}(\boldsymbol{c}[\boldsymbol{s}]) \qquad \in \mathbb{R}^{n \times 1} \tag{4.8}$$

$$\boldsymbol{X}^0 = \boldsymbol{R}[\boldsymbol{s}] \odot \tilde{\boldsymbol{c}}\ \boldsymbol{W}_1 + \tilde{\boldsymbol{c}}\ \boldsymbol{W}_2 \qquad \in \mathbb{R}^{n \times c} \tag{4.9}$$

$$\boldsymbol{X}^0 = \boldsymbol{X}^0 \odot \mathbf{1}_n\ \boldsymbol{z}^T\ \boldsymbol{W}_3 + \mathbf{1}_n\ \boldsymbol{z}^T\ \boldsymbol{W}_4 \qquad \in \mathbb{R}^{n \times c} \tag{4.10}$$

The crux of the Top-n creation module is to select the points that will be used to generate a set based on the value of the latent vector (Eq. 5, 6 ,7). Unfortunately, the gradient of the argsort operation is 0 almost everywhere ($\partial \boldsymbol{R}[\boldsymbol{s}]/\partial \boldsymbol{\Phi} = 0$), and a mechanism has to be used in order to train the angles $\boldsymbol{\Phi}$ and the MLP of Eq. (5). In Top-n, we modulate the representation of the selected points $\boldsymbol{R}[\boldsymbol{s}]$ with the cosines $\boldsymbol{c}$ (Eq. 8). This operation provides a path in the computational graph that does not go through the argsort, so that the gradients of $\boldsymbol{\Phi}$ and $\boldsymbol{a}$ are not always 0. For example:

$$\frac{d\boldsymbol{X}^0}{d\boldsymbol{\Phi}} = \frac{\partial \boldsymbol{X}^0}{\partial \boldsymbol{R}[\boldsymbol{s}]} \frac{d\boldsymbol{R}[\boldsymbol{s}]}{d\boldsymbol{\Phi}} + \frac{\partial \boldsymbol{X}^0}{\partial \tilde{\boldsymbol{c}}} \frac{d\tilde{\boldsymbol{c}}}{d\boldsymbol{\Phi}} = \frac{\partial \boldsymbol{X}^0}{\partial \tilde{\boldsymbol{c}}} \frac{d\tilde{\boldsymbol{c}}}{d\boldsymbol{\Phi}},$$

Equations (5) to (9) build upon the Top-K pooling mechanism used by Gao & Ji (2019) for graph coarsening, but differ in several aspects. First, Gao & Ji (2019) compute cosines between the angle $\boldsymbol{a}$ and the representations $\boldsymbol{R}$. This method tends to select points that are similar. On the contrary, we parameterize the angles and the representations independently, so that two points can have similar angles (in which case they will usually be selected together) but very diverse representations at the same time. Second, Gao & Ji (2019) uses a multiplicative modulation

($\boldsymbol{X}^0 = \boldsymbol{X}_{\text{ref}}[\boldsymbol{s}] \odot \tilde{\boldsymbol{c}}$) in Eq. (9) while we use a more expressive FiLM layer (Perez et al., 2018) that combines both additive and multiplicative modulation.

Finally, while previous works usually append the latent vector to each point, we exploit the equivalence between summation and concatenation when a linear layer is applied, which writes

$$\text{cat}(\boldsymbol{X}^0,\ \mathbf{1}_n \boldsymbol{z}^T)\ \boldsymbol{W} = \boldsymbol{X}^0 \boldsymbol{W}_1 + \mathbf{1}_n(\boldsymbol{z}^T \boldsymbol{W}_2), \tag{4.11}$$

for $\boldsymbol{W} = \text{cat}(\boldsymbol{W}_1, \boldsymbol{W}_2)$. Contrary to the concatenation (left-hand side), the sum (right-hand side) does not compute $\boldsymbol{z}^T \boldsymbol{W}_2$ several times, which reduces the complexity of this layer from $O(n(c+l)c)$ to $O(nc^2 + cl + nl)$. Again, we combine the sum and multiplicative modulation in a FiLM layer to build a more expressive model in Eq. (10).

Our algorithm retains the advantages of First-n creation, but replaces the arbitrary selection of the first n points by a mechanism that learns to select the most relevant points for each set. Since it also decouples the number of points in the reference set from the number of points in the training examples, the size of the reference set becomes a hyper-parameter of the model. Empirically, we observe a tradeoff: when using more points in the reference set, each point is updated less often which makes training slower; however, the model tends to avoid overfitting and generalize better. Top-n creation can be used in any GAN or VAE architecture as a replacement for other set creation methods. It is however not suited to normalizing flows, because it is based on a hard selection process which is not invertible (the value of the reference points that are not selected is not used).

Since First-n and Top-n use a fixed set of reference points, one may wonder if they restrict the model expressivity. We however show that it is not the case: used with a two-layer MLP, the First-n and Top-n modules are universal approximators over sets.

Proposition 3 (Expressivity)**.** *For any set size n, maximal norm M and precision parameter ϵ, there is a 2-layer row-wise MLP f and reference points $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ such that, for any set $\{\boldsymbol{y}_1, ..., \boldsymbol{y}_n\}$ of points in $\mathbb{R}^d$ with $\forall i,\ ||\boldsymbol{y}_i|| \leq M$ there is a latent vector $\boldsymbol{z}$ of size $nd \times 1$ that satisfies:*

$$||f(\text{cat}(\boldsymbol{X}, \mathbf{1}_n \boldsymbol{z}^T)) - \boldsymbol{Y}||_{\mathcal{W}_2} \leq \epsilon \tag{4.12}$$

where $\mathcal{W}_2$ denotes the Wasserstein-2 distance.

Proof. We first give the proof for First-n creation:

First-n Given a set $\boldsymbol{Y} = \{\boldsymbol{y}_1, ..., \boldsymbol{y}_n\}$ of points in $\mathbb{R}^d$, we propose to sort the points $\boldsymbol{y}_i$ by alphanumeric sort (sort the values using the first feature, then sort points that have the same first features along the second feature, etc.). We denote the resulting matrix by $\boldsymbol{Y}'$. We choose as a latent vector $\boldsymbol{z} = \textit{flatten}(\boldsymbol{Y}') \in \mathbb{R}^{nd}$. It is the vector that contains the representation of $\boldsymbol{y}'_1$ in the first d features, $\boldsymbol{y}'_2$ is the next d features... By construction, $\boldsymbol{z}$ is a permutation invariant representation of the set $\boldsymbol{Y}$ (this reflects the fact that there are canonical ordering for sets, which

is not the case for general graphs).

We choose as a reference set the canonical basis in $\mathbb{R}^n$ ($\boldsymbol{R} = \boldsymbol{I}_n$). After the latent vector is appended to the representation of each point, we have $\boldsymbol{r}_i = (\boldsymbol{e}_i, \boldsymbol{z})$. We are now looking for a function f that allows to approximate the set $\boldsymbol{Y}$, i.e., that satisfies $\forall i \leq n, f(\boldsymbol{e}_i, \boldsymbol{z}) = \boldsymbol{z}[i\ d : (i+1)d]$. We can choose $f(\boldsymbol{e}_i, \boldsymbol{z}) = \boldsymbol{e}_i^T \boldsymbol{W}_1 \boldsymbol{W}_2 \boldsymbol{z}$, where

$$\boldsymbol{W}_1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & & 1 & 1 & 1 \end{bmatrix} \in \mathbb{R}^{n \times nd} \quad \text{and} \quad \boldsymbol{W}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ & \dots & \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{nd \times d}$$

.

This function is continuous, and its output is $\boldsymbol{Y}' = \textit{reshape}_{n \times d}(\boldsymbol{z})$. $\boldsymbol{Y}'$ is equal to $\boldsymbol{Y}$ up to a permutation of the rows, so that $||\boldsymbol{Y} - \boldsymbol{Y}'||_{\mathcal{W}_2} = 0$. If the entries of $\boldsymbol{z}$ are bounded, we can use standard approximation results for continuous functions over a compact space (Cybenko, 1989) to conclude that it be uniformly approximated by a 2-layer MLP.

Top-n Consider a Top-n network with n reference points such that:

- The angles of the reference points are 2d vectors such that $\phi_i = (\cos(\frac{i}{n}\frac{\pi}{4}), \sin(\frac{i}{n}\frac{\pi}{4}))$.
- The representations are $r_i = e_i / \cos(\frac{i}{n_0}\frac{\pi}{4})$, where (e_i) is the canonical basis in $\mathbb{R}^n$.
- The MLP of equation 1 (that predicts an angle from the latent vector) always outputs $(1, 0)$.

Then this Top-n creation module is equivalent to the First-n module build previously: for any set, it selects the first n points and returns $\boldsymbol{X}^0[i] = \boldsymbol{e}_i$. The same MLP that is built for First-n can therefore be used for this network. □

4.5 Experiments

We compare Top-n to other set and graph creation methods on several tasks: autoencoding a set version of MNIST, detecting objects on CLEVR, generating realistic 3D structures on a synthetic molecule-like dataset, and generating varied valid molecules on the QM9 chemical dataset[3]. All training curves are available in Appendix B.2.

[3]Source code is available at `github.com/cvignac/Top-N`

Table 4.1 – Mean Chamfer loss and 95% confidence interval over 6 runs. Methods in italic are those used in the original papers for TSPN (Kosiorek & Kim, 2020) and DSPN (Zhang et al., 2019) Result differ from the original papers due to a difference in the loss computation (cf. Appendix B.1).

Method	Set creation	Chamfer (e-5)	Method	Set creation	Chamfer (e-5)
TSPN	*i.i.d. sampling*	16.42 ± 0.53	DSPN	i.i.d. sampling	28.56 ± 1.23
	First-n	$\mathbf{15.45} \pm 1.41$		*First-n*	26.61 ± 0.54
	Top-n	$\mathbf{14.98} \pm 0.59$		Top-n	$\mathbf{22.59} \pm 1.71$

Set MNIST We first perform experiments on the SetMNIST benchmark, introduced in Zhang et al. (2019). The task consists in autoencoding point clouds that are built by thresholding the pixel values in MNIST images, adding noise on the locations and normalizing the coordinates. Our goal is to show that Top-n can favorably replace other set creation methods without having to tune the rest of the architecture extensively. For this purpose, we use existing implementations of DSPN (Zhang et al., 2019) TSPN (Kosiorek & Kim, 2020)[4], which are respectively a sort of diffusion model and a transformer-based autoencoder. Experiment details can be found in Appendix B.1.

The results for both methods are very similar (Figure 4.1): the model based on independent sampling has poor performance and needs more epochs to be trained than First-n and Top-n. Top-n performs consistently better for both DSPN and TSPN, which shows that it is able to select the most relevant reference points for each set.

Object detection on CLEVR We further benchmark Top-n on object detection with the CLEVR dataset, made of 70k training images and 15k validation images representing simple objects. Again, we use the implementation of DSPN and the setting proposed in (Zhang et al., 2019). Two tasks are evaluated. In the first one, the goal is to predict bounding boxes in each image. In the second one, the full scene should be predicted (with the shape, color, size and material of the objects). Images are encoded using a pretrained ResNet34 architecture – the resulting vector is used as input to the set generation model. The model is trained on 10 DSPN iterations and evaluated on 30, which is the setting that gave the best results in (Zhang et al., 2019).

Results are presented in Tables 4.2 and 4.3. For bounding box prediction (which is a simpler task), independent sampling and Top-n outperform MLP and First-n creation. As the metrics are computed for test data (which is not the case in SetMNIST), these results suggest that MLP and First-n creation may overfit the training images. On the full scene prediction, Top-n outperforms all other methods.

[4] github.com/LukeBolly/tf-tspn (reimplementation by someone else) and github.com/Cyanogenoid/dspn

Table 4.2 – Bounding box prediction on CLEVR. The metric is the average precision on the test set for different intersection-over-union thresholds, computed over 6 runs (higher is better).

Model	Generator	AP_{50}	AP_{60}	AP_{70}	AP_{80}	AP_{90}
DSPN	MLP	93.7±1.8	82.8±3.2	59.6±4.8	26.2±4.5	1.8±0.8
	i.i.d. sampling	**97.3**±2.0	**93.2**±3.7	**80.6**±5.4	**51.8**±5.5	**11.6**±2.3
	First-n	88.2±5.1	77.1±7.3	57.3±8.2	29.0±6.1	4.0±1.3
	Top-n	**97.3**±1.3	**93.0**±2.8	**80.8**±5.0	**53.0**±7.0	**12.5**±3.9

Table 4.3 – Full scene prediction on CLEVR. The metric is the average precision on the test set, computed over 6 runs (higher is better). While MLP and i.i.d. sampling have better training metrics, Top-n generalizes much better to new images.

Model	Generator	AP_{10}	AP_{20}	AP_{50}	AP_{100}	AP_{inf}
DSPN	MLP	2.7±1.4	17.9±8.6	42.1±16.8	54.5±19.4	71.2±3.0
	i.i.d. sampling	2.6±1.3	26.0±9.1	60.5±11.1	76.6±5.2	80.4±4.3
	First-n	0.7±0.4	11.7±4.3	50.3±9.1	81.2±5.3	84.8±5.0
	Top-n	**8.3**±1.9	**48.2**±6.4	**86.4**±3.8	**93.0**±2.6	**94.1**±2.3

Table 4.4 – Mean and 95% confidence interval over 5 runs on synthetic molecule-like data in 3d.

	Train	Test	Generation			Extrapolation		
	$\mathcal{W}_2$ distance	$\mathcal{W}_2$ distance	Valency loss	Incorrect valency	Diversity score	Valency loss	Incorrect valency	Diversity score
MLP	**0.47**±.07	**0.42**±.03	0.73±.06	22.4±2.2	4.6±.1	1.06±.14	26.6±1.9	4.3±.2
i.i.d.	1.20±.01	0.81±.12	1.40±.07	36.8±20.9	5.0±.3	**0.24**±.06	**7.8**±0.5	**4.9**±.2
First-n	**0.43**±.08	0.50±.07	0.84±.06	24.6±2.1	5.1±.3	0.81±.19	22.5±3.0	**4.6**±.3
Top-n	0.58±.05	**0.44**±.03	**0.37**±.12	**13.9**±3.7	4.8±.2	0.80±.10	17.0±1.8	4.5±.2

Synthetic dataset As previous datasets do not measure generation quality, we further benchmark the different set generators on a synthetic dataset for which the quality of the generated sets can be assessed. This dataset is a simplified model of molecules in 3D, and retains some of its characteristics: i) atoms are never too close to each other ii) there is a bond between two atoms if and only if they are closer than a given distance (that depends on the atom types) iii) if formal charges are forbidden, each atom has a predefined valency. The generation procedure is described in Appendix B.1.

The goal is to reconstruct the atom positions and generate new realistic sets. During training, we measure the Wasserstein ($\mathcal{W}_2$) distance between the input and reconstructed sets. At generation time, we compute the $\mathcal{W}_2$ distance between the distribution of valencies in the dataset and in the generated set, the proportion of generated atoms with valency 0 or more than 4, as well as a diversity score to ensure that a method does not always generate the same set. We also measure the same metrics in an extrapolation setting where sets have on average 10 more points.

We train a VAE with different creation methods on this dataset. Details about the model and the loss function can be found in Appendix B.1, as well as an ablation study on the number of

Table 4.5 – Molecular graph generation on QM9. Baseline results are from the original authors. Our architecture provides an effective approach to one-shot molecule generation. Apart from independent sampling creation, the different set creation methods seem to be equivalent in this setting.

Method	Generator	Valid (%)	Unique and valid
Graph VAE (Simonovsky & Komodakis, 2018)	MLP	55.7	42.3
Graph VAE + RL (Kwon et al., 2019)	MLP	94.5	32.4
MolGAN (De Cao & Kipf, 2018)	MLP	98.0	2.3
GTVAE (Mitton et al., 2021)	MLP	74.6	16.8
Set2GraphVAE (ours)	MLP	60.5± 2.2	**55.4**±2.3
	i.i.d. sampling	34.9±15.2	29.9±10.0
	First-n	59.9± 2.7	**56.2**±2.7
	Top-n	59.9± 1.4	**56.2**±1.1

points in the reference set. The results are shown in Table 4.4. We observe that the independent sampling generator generalizes well but reconstructs the training sets very poorly, which reflects the fact that the model is hard to train due to the stochastic i.i.d. generation. They tend to generate points that are too far apart, an issue which disappears when generating more points. On the contrary, MLP and First-n overfit the training data at the expense of generation quality. Finally, Top-n is able to generate points that have the right valency and generates the best new samples.

Finally, we evaluate Top-n on a graph generation task. We train a graph VAE (detailed in Appendix B.1) on QM9 molecules and check its ability to generate a wide range of valid molecules. As a generation metric, we simply report the validity (i.e., the proportion of molecules that satisfy a set of basic chemical rules) and uniqueness (the proportion non-isomorphic graphs among valid ones) of the generated molecules. We do not report novelty, as QM9 is an enumeration of all possible molecules up to 9 heavy atoms that satisfy a predefined set of constraints (Ruddigkeit et al., 2012; Ramakrishnan et al., 2014): in this setting, generating novel molecules is therefore not an indicator of good performance, but rather a sign that the distribution of the training data has not been properly captured. Note that most recently proposed methods proposed on this task use a non-learned validity correction code to generate almost 100% of valid molecules, which obfuscates the real performance of the learned model. We therefore only compare to works that do not correct validity. We also note that recursive methods such as Li et al. (2018a) can often generate higher rates of valid molecules because they can easily check at each step that the added edge will not break valency constraints. They have therefore an unfair advantage over one-shot models which do not incorporate these checks.

While MolGAN and GraphVAE+RL both suffer from mode collapse, our method is able to generate a higher rate of valid and unique molecules. We observe that the independent sampling method is not able to obtain a good train loss (Appendix B.2), which reflects in the poor generation performance as well. The three other set creation methods seem to perform similarly. Our interpretation is that almost all molecules in QM9 have the same size (9 heavy atoms, since hydrogens are not represented), which makes it less important to properly handle varying graph

sizes as in Top-n.

4.6 Discussion

While the Top-n creation layer is able to favorably replace other creation layers in diverse architectures, some limitations can be identified:

The Modulation Mechanism of Top-n Creation. While several methods had been proposed for learning sets and graphs from vectors, the set creation mechanisms had not been studied before. An important contribution of this work is to highlight the crucial role of this layer in the neural network and emphasize the need for careful selection. With respect to First-n creation, the proposed Top-n creation module introduces a differentiable mechanism for selecting the most relevant points inside a reference set. However, the modulation mechanism used to obtain gradients for backpropagation could probably be improved. In Top-n creation, the representation of each point is modulated by the cosine with the latent vector. As a result, the magnitude of the selected points depend on the value of the cosines, which allows to train the angle associated to each point in the reference set. However, the magnitude of the selected points constitutes limited information, there might be better mechanisms that allow to train the point selection process more effectively.

Curse of dimensionality. We note that most of our experiments feature only small sets, which is in line with existing literature (Meldgaard et al., 2021; Satorras et al., 2021b; Mitton et al., 2021). This raises the important question of whether set and graph generation methods can overcome the curse of dimensionality. For MLP-based generators, the answer is likely no, as they learn vectors of size $n_{\text{max}}d$, and standard universal approximation results suggest a sample complexity that is exponential in n_{max}. Whether other set generators can overcome this issue remains an open question that has implications for the scalability of one-shot models for larger sets and graphs.

Vector-shaped latent spaces. Another reason for the difficulty to to generate large graphs might be related to the use of vector-shaped latent spaces. Consider, for example, an autoencoder for graph generation that uses a latent space of size $\mathbb{R}^l$. If there was a way to perfectly encode a graph to a vector using a permutation invariant function, and then to decode this vector back to a graph, then this function would perform graph canonization. As canonization is at least as hard as isomorphism testing, it is not surprising that such a function is hard to build in practice. In the next section, we will define models that bypass this limitation by defining a generative mechanism that does not use a latent space.

4.7 Conclusion

Overall, we strengthened in this chapter the theoretical foundations of one-shot set and graph generation. We showed that, contrary to common belief, exchangeability is not a required property

in GANs and VAE. We then proposed Top-n, a non-exchangeable layer for one-shot set and graph creation which is able to select the most relevant points for each set. Our method can be incorporated in any GAN or VAE architecture and replace favorably other set creation methods. In next chapter, we will study a very different class of generative models, that do not require compressing the data to a lower dimensional space. We will show that viewing graph generation as the denoising of a random graph allows to significantly boost performance.

5 Diffusion models for unordered data

5.1 Introduction

In previous chapter, we studied generative settings in which the latent space has a vector structure. We proposed the Top-n creation mechanism, which was able to improve the performance of several architectures. Yet, the experiments mostly featured very small sets and graphs. With larger data, we observed that it was very difficult to design autoencoders that are able to reconstruct the data, and the scaling abilities seemed very limited. This is unsurprising given that a deterministic encoder-decoder architecture for graphs with a permutation invariant encoder would be able to solve graph canonization.

In this chapter, we consider denoising diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020), a class of architectures that operates very differently. Denoising diffusion models are trained to denoise corrupted data points, and generate new data by iteratively denoising pure noise. Diffusion models have been used successfully in a variety of settings, outperforming all other methods on image and video (Dhariwal & Nichol, 2021; Ho et al., 2022). These successes raise hope for building powerful models for point clouds and graphs as well.

Furthermore, diffusion models seem particularly suited to permutation equivariant learning: they do not define a latent space, and do not require compressing the data into a low-dimensional object. Consequently, these models avoid graph coarsening, graph pooling, and graph decoding from a vector. Instead, they simply use node and edge-level prediction, which is something that graph networks typically excel at. We will see that this results in drastic improvements in generation quality, which paves the way to many interesting applications in drug discovery, computational biology and material science.

In this chapter, we present the DiGress model for graph generation, and its follow-up MiDi for graphs whose nodes also have 3D coordinates. For the sake of brevity, we will not present the EDM model[1], which defines a diffusion model for attributed point cloud generation. EDM has

[1]Equivariant Diffusion for Molecule Generation in 3D, ICML 2022 – Hoogeboom*, Satorras*, Vignac* & Welling

been used to build the continuous baseline of DiGress, and its limitations serve as a motivation for the MiDi model.

Our first model, DiGress, is designed to generate graphs with categorical node and edge attributes. DiGress leverages discrete diffusion, which was previously applied successfully to text, images, audio, and attributed point clouds (Austin et al., 2021; Yang et al., 2022; Luo et al., 2022a). In discrete diffusion, the noise model is a Markov process consisting of successive graphs edits (edge addition or deletion, node or edge category edit) that can occur independently on each node or edge.

We propose several algorithmic enhancements to DiGress, including utilizing a noise model that preserves the marginal distribution of node and edge types during diffusion, introducing a novel guidance procedure for conditioning graph generation on graph-level properties, and augmenting the input of our denoising network with auxiliary structural and spectral features. These features, derived from the noisy graph, aid in overcoming the limited representation power of graph neural networks (Xu et al., 2019). Their use is made possible by the discrete nature of our noise model, which, in contrast to Gaussian-based models, preserves sparsity in the noisy graphs. These improvements enhance the performance of DiGress on a wide range of graph generation tasks.

We then propose the Mixed Graph+3D Denoising Diffusion (MiDi) model to simultaneously generate a molecular graph and its corresponding 3D coordinates. When both data modalities are simultaneously available, they can be considered together to better capture the chemical space. The molecular graph (or 2D structure) determines the existence and type of the chemical bonds, and allows the identification of functional groups in a compound. This provides information about its chemical properties and enables to predict synthetic pathways. On the other hand, the 3D conformation of a compound plays a key role in its interaction with other molecules, and governs in particular its biological activity and binding affinity to proteins.

Previous EDM model, and other models that built upon it, were trained to generate conformers only, thus ignoring bond information. They relied on a subsequent step to predicts the 2D structure using either interatomic distances (Satorras et al., 2021a; Hoogeboom et al., 2022) or chemical software such as OpenBabel (Gebauer et al., 2019). As a result, these models are not end-to-end differentiable, which hampers their ability to be fully optimized for various downstream tasks. This severely limits the potential of 3D molecule generators, particularly for complex tasks like pocket-conditioned generation.

In constrast, MiDi is trained to denoise both the graph and 3D coordinates in tandem, it is able to produce stable molecular graphs that are consistent with the generated conformers. It uses Gaussian noise for the 3D coordinates, while the other components use discrete diffusion. To further enhance the quality of the generated samples, we introduce a noise schedule whose parameters are adjusted to each component. Specifically, we add noise to the atom types and formal charges at a faster rate than to the coordinates and bond types. This encourages the

denoising network to first focus on generating a realistic 3D conformation and corresponding bond types, before refining the atom types and formal charges.

The second contribution of MiDi considers the denoising network: the Transformer architecture we propose incorporates a novel *rEGNN* layer, which improves upon the popular EGNN layers Satorras et al. (2021b) by leveraging features that are not translation-invariant. We show that, due to the use of Gaussian noise in the zero center-of-mass subspace of the molecules, the resulting model is nevertheless equivariant to translations and rotations, which is crucial for achieving high performance.

5.2 Related Work

5.2.1 Graph generation

Our work, DiGress, is the first discrete diffusion model for graphs. Concurrently, Haefeli et al. (2022) designed a model limited to unattributed graphs, and similarly observed that discrete diffusion is beneficial for graph generation. Previous diffusion models for graphs were based on Gaussian noise: Niu et al. (2020) generated adjacency matrices by thresholding a continuous value to indicate edges, and Jo et al. (2022) extended this model to handle node and edge attributes, and (Luo et al., 2022b) applied Gaussian noise to the graph eigenvalues. Gaussian noise can also easily be considered in the time continuous limit, as done in (Huang et al., 2022a).

In addition to diffusion models, many other classes of architectures have been proposed for graph generation. They can be divided in autoregressive methods and one-shot methods. Autoregressive methods (Liu et al., 2018; You et al., 2018a; Liao et al., 2019) add one node at a time, therefore artificially creating a node ordering that does not exist in the data. These methods typically check for validity at each sampling step, so that they tend to score very high on validity metrics. Furthermore, they can quite naturally incorporate domain knowledge through the use of reinforcement learning (Mercado et al., 2021).

On the contrary one-shot approaches generate the graphs at once, resulting in much faster sampling. Examples of these architectures are VAEs (Simonovsky & Komodakis, 2018), GANs (De Cao & Kipf, 2018; Krawczuk et al., 2021; Kosiorek & Kim, 2020) and normalizing flows. We refer the reader to (Zhu et al., 2022) for a survey of these methods. While many of these models operate on continuous tensor, (Madhawa et al., 2019; Lippe & Gavves, 2021; Luo et al., 2021c) are examples of discrete models using categorical normalizing flows.

In contrast to the proposed method, which operates at the node level, fragment-based methods (Hajduk & Greer, 2007; Jin et al., 2020; Maziarz et al., 2022) learn to combine chemically-relevant substructures from a fixed or learned dictionary (Wang et al., 2022). These methods are often very effective, as they only have to combine a limited number of fragments to build a molecule. They could constitute an interesting extension to the models proposed in this chapter.

5.2.2 Molecule generation in 3D

Some recent methods directly generate molecules in 3D without learning the connectivity structure: (Gebauer et al., 2019; Luo & Ji, 2021; Luo et al., 2021a; Gebauer et al., 2021) define an order-dependent autoregressive distribution from which atoms are iteratively sampled. (Ragoza et al., 2020) maps atoms to a fixed grid and trains a VAE using 3D convolutions. E-NF (Satorras et al., 2021a) defines an equivariant normalizing flow that integrates a differential equation. Instead, the proposed methods learn to denoise a diffusion process, which scales better during training. Our model EDM (Hoogeboom et al., 2022) was later extended by limiting the message-passing computations to neighboring nodes (Huang et al., 2022b) and using a more expressive denoising network (Morehead & Cheng, 2023).

All these diffusion models can be conditioned on molecule-level properties using guidance mechanisms (Bao et al., 2022) or on another point cloud. Conditioning on a second point cloud has been employed to generate molecules that bind to a specific protein (Corso et al., 2022; Schneuing et al., 2022) and to generate linkers between molecular fragments (Igashov et al., 2022). The main drawback of these models is that they do not learn the connectivity structure of the molecule. It needs to be obtained in a second stage using interatomic distances (Satorras et al., 2021a; Hoogeboom et al., 2022) or specialized software such as Open Babel (O'Boyle et al., 2011). This results in limited performance for complex molecules, but also prevents end-to-end differentiability for downstream applications.

Conformer generation A related branch of literature is concerned by solely predicting coordinates from molecular graphs, referred to as the conformation. Examples of such methods utilize conditional VAEs (Simm & Hernández-Lobato, 2019), Wasserstein GANs (Hoffmann & Noé, 2019), and normalizing flows (Noé et al., 2019), with adaptions for Euclidean symmetries in (Köhler et al., 2020; Xu et al., 2021; Simm et al., 2021; Ganea et al., 2021; Guan et al., 2022) resulting in performance improvements. In recent works (Shi et al., 2021; Luo et al., 2021b; Xu et al., 2022) it was shown that score-based and diffusion models are effective at coordinate prediction, especially when the underlying neural network respects the symmetries of the data. In particular, as conformer generation models assume that the graph is known, they are able to exploit symmetries of the molecule such as rotatable bonds (Corso et al., 2022). This is unfortunately more difficult for unconditional generation tasks, which do not have access to an input graph structure.

Protein generation While existing diffusion models for molecules operate on molecules of moderate size (up to 180 atoms), recent diffusion models for proteins have managed to scale to much larger structures (Trippe et al., 2022; Watson et al., 2022; Ingraham et al., 2022; Wu et al., 2022; Shi et al., 2022). These methods leverage the chain structure of proteins, which implies that the adjacency matrix does not need to be predicted. Furthermore, instead of predicting 3D coordinates for each atom, they only predict the angles between successive C_α carbons,

which significantly reduces the degrees of freedom and encodes roto-translation invariance in the representation. Those improvements are unfortunately specific to chain graphs, and cannot be used for arbitrary molecules.

5.3 Background on diffusion models

Diffusion models consist of two essential elements: a noise model and a denoising neural network. The noise model q takes as input a data point x and generates a trajectory of increasingly corrupted data points $(z_1, ..., z_t)$. The corruption process is chosen to be Markovian, i.e.,

$$q(z_1, \ldots, z_T|x) = q(z_1|x) \prod_{t=2}^{T} q(z_t|z_{t-1}).$$

The denoising network ϕ_θ takes noisy data z_t as input, and learns to invert the diffusion trajectories. A key property of diffusion models is that they do not directly try to predict z_{t-1}, which constitutes a noisy targets that depends on the sampled diffusion trajectory. Instead, modern diffusion models (Song & Ermon, 2019; Ho et al., 2020) predict the clean input x from z_t, or equivalently, the noise added to it. The diffusion sequences are then inverted by marginalizing over the network predictions $p_\theta(x|z_t)$:

$$p_\theta(z_{t-1}|z_t) = \int_x p_\theta(z_{t-1} \mid x, z_t)\, dp_\theta(x|z_t) \tag{5.1}$$

Although Eq. 5.1 leads to more efficient training, it requires the use of a noise model that satisfies several properties:

1. The distribution $q(z^t|x)$ should have a closed-form formula, to allow for parallel training on different time steps.
2. The posterior $p_\theta(z^{t-1}|z^t) = \int q(z^{t-1}|z^t, x) dp_\theta(x)$ should have a closed-form expression, so that x can be used as the target of the neural network.
3. The limit distribution $q_\infty = \lim_{T\to\infty} q(z^T|x)$ should not depend on x, so that we can use it as a prior distribution for inference.

Two main frameworks have been proposed under which these properties are satisfied: Gaussian noise, which is suitable for continuous data, and discrete state-space diffusion for categorical data. Table 5.1 summarizes the main properties of the two related noise models.

Table 5.1 – Gaussian and categorical distributions enable the efficient computation of the key quantities involved in training diffusion models and sampling from them. Formulas for all parameters can be found in Section 5.3.1.

Noise model	Gaussian diffusion	Discrete diffusion
$q(\boldsymbol{z}_t \vert \boldsymbol{z}_{t-1})$	$\mathcal{N}(\alpha_t \boldsymbol{z}_{t-1},\ \sigma_t^2 \boldsymbol{I})$	$\boldsymbol{z}_{t-1}\, \boldsymbol{Q}_t$
$q(\boldsymbol{z}_t \vert \boldsymbol{x})$	$\mathcal{N}(\bar{\alpha}_t \boldsymbol{x},\ \bar{\sigma}_t^2 \boldsymbol{I})$	$\boldsymbol{x}\, \bar{\boldsymbol{Q}}_t$
$\int_x p_\theta(\boldsymbol{z}_{t-1} \vert \boldsymbol{x}, \boldsymbol{z}_t) dp_\theta(\boldsymbol{x} \vert \boldsymbol{z}_t)$	$\mathcal{N}(\mu_t \hat{\boldsymbol{x}} + \nu_t \boldsymbol{z}_t, \tilde{\sigma}_t^2 \boldsymbol{I})$	$\propto \sum_x p_\theta(x)(\boldsymbol{z}_t \boldsymbol{Q}_t' \odot \boldsymbol{x} \bar{\boldsymbol{Q}}_{t-1})$

5.3.1 Gaussian Diffusion

Given a vector-shaped data point $\boldsymbol{x}$, a Gaussian diffusion process that adds noise to $\boldsymbol{z}_t$ for $t = 0, \ldots, T$ is defined by the multivariate normal distribution:

$$q(\boldsymbol{z}_t \vert \boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}_t \vert \alpha_t \boldsymbol{x}_t, \sigma_t^2 \mathbf{I}), \tag{5.2}$$

where $\alpha_t \in \mathbb{R}^+$ controls how much signal is retained and $\sigma_t \in \mathbb{R}^+$ controls how much noise is added. In general, α_t is modelled by a function that smoothly transitions from $\alpha_0 \approx 1$ towards $\alpha_T \approx 0$. A special case of noising process is the variance preserving process (Sohl-Dickstein et al., 2015; Ho et al., 2020) for which $\alpha_t = \sqrt{1 - \sigma_t^2}$. Following Kingma et al. (2021), we define the signal to noise ratio $\mathrm{SNR}(t) = \alpha_t^2 / \sigma_t^2$, which simplifies notations.

This diffusion process is Markov and can be equivalently written with transition distributions as:

$$q(\boldsymbol{z}_t \vert \boldsymbol{z}_s) = \mathcal{N}(\boldsymbol{z}_t \vert \alpha_{t\vert s} \boldsymbol{z}_s,\ \sigma_{t\vert s}^2 \mathbf{I}), \tag{5.3}$$

for any $t > s$ with $\alpha_{t\vert s} = \alpha_t / \alpha_s$ and $\sigma_{t\vert s}^2 = \sigma_t^2 - \alpha_{t\vert s}^2 \sigma_s^2$. The entire noising process is then written as:

$$q(\boldsymbol{z}_0, \boldsymbol{z}_1, \ldots, \boldsymbol{z}_T \vert \boldsymbol{x}) = q(\boldsymbol{z}_0 \vert \boldsymbol{x}) \prod_{t=1}^{T} q(\boldsymbol{z}_t \vert \boldsymbol{z}_{t-1}). \tag{5.4}$$

The posterior of the transitions conditioned on $\boldsymbol{x}$ gives the inverse of the noising process, the *true denoising process*. It is also normal and given by:

$$q(\boldsymbol{z}_s \vert \boldsymbol{x}, \boldsymbol{z}_t) = \mathcal{N}(\boldsymbol{z}_s \vert \boldsymbol{\mu}_{t \to s}(\boldsymbol{x}, \boldsymbol{z}_t), \sigma_{t \to s}^2 \mathbf{I}), \tag{5.5}$$

where the definitions for $\boldsymbol{\mu}_{t \to s}(\boldsymbol{x}, \boldsymbol{z}_t)$ and $\sigma_{t \to s}$ can be analytically obtained as

$$\boldsymbol{\mu}_{t \to s}(\boldsymbol{x}, \boldsymbol{z}_t) = \frac{\alpha_{t\vert s} \sigma_s^2}{\sigma_t^2} \boldsymbol{z}_t + \frac{\alpha_s \sigma_{t\vert s}^2}{\sigma_t^2} \boldsymbol{x} \quad \text{and} \quad \sigma_{t \to s} = \frac{\sigma_{t\vert s} \sigma_s}{\sigma_t}.$$

The Generative Denoising Process In contrast to other generative models, in diffusion models, the generative process is defined with respect to the *true denoising process*. The variable $\boldsymbol{x}$, which is unknown to the generative process, is replaced by an approximation $\hat{\boldsymbol{x}} = \phi(\boldsymbol{z}_t, t)$ given by a neural network ϕ. Then the generative transition distribution $p(\boldsymbol{z}_s \vert \boldsymbol{z}_t)$ is chosen to be

$q(\boldsymbol{z}_s|\hat{\boldsymbol{x}}(\boldsymbol{z}_t, t), \boldsymbol{z}_t)$. Similarly to Eq. C.3, it can be expressed using the approximation $\hat{\boldsymbol{x}}$ as:

$$p(\boldsymbol{z}_s|\boldsymbol{z}_t) = \mathcal{N}(\boldsymbol{z}_s|\boldsymbol{\mu}_{t\to s}(\hat{\boldsymbol{x}}, \boldsymbol{z}_t), \sigma^2_{t\to s}\mathbf{I}). \tag{5.6}$$

With the choice $s = t - 1$, a variational lower bound on the log-likelihood of $\boldsymbol{x}$ given the generative model is given by:

$$\log p(x) \geq \mathcal{L}_0 + \mathcal{L}_{\text{base}} + \sum\nolimits_{t=1}^{T} \mathcal{L}_t, \tag{5.7}$$

where $\mathcal{L}_0 = \log p(\boldsymbol{x}|\boldsymbol{z}_0)$ models the likelihood of the data given $\boldsymbol{z}_0$, $\mathcal{L}_{\text{base}} = -\text{KL}(q(\boldsymbol{z}_T|\boldsymbol{x})|p(\boldsymbol{z}_T))$ models the distance between a standard normal distribution and the final latent variable $q(\boldsymbol{z}_T|\boldsymbol{x})$, and

$$\mathcal{L}_t = -\text{KL}(q(\boldsymbol{z}_s|\boldsymbol{x}, \boldsymbol{z}_t)|p(\boldsymbol{z}_s|\boldsymbol{z}_t)) \quad \text{for } t = 1, \ldots, T.$$

While in this formulation the neural network directly predicts $\hat{\boldsymbol{x}}$, Ho et al. (2020) found that optimization is easier when predicting the Gaussian noise instead. Intuitively, the network is trying to predict which part of the observation $\boldsymbol{z}_t$ is noise originating from the diffusion process, and which part corresponds to the underlying data point $\boldsymbol{x}$. Specifically, if $\boldsymbol{z}_t = \alpha_t\boldsymbol{x} + \sigma_t\boldsymbol{\epsilon}$, then the neural network ϕ outputs $\hat{\boldsymbol{\epsilon}} = \phi(\boldsymbol{z}_t, t)$, so that:

$$\hat{\boldsymbol{x}} = (1/\alpha_t)\, \boldsymbol{z}_t - (\sigma_t/\alpha_t)\, \hat{\boldsymbol{\epsilon}} \tag{5.8}$$

As shown in (Kingma et al., 2021), with this parametrization $\mathcal{L}_t$ simplifies to:

$$\mathcal{L}_t = \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\frac{1}{2}(1 - \text{SNR}(t-1)/\text{SNR}(t))||\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}||^2\right] \tag{5.9}$$

In practice the term $\mathcal{L}_{\text{base}}$ is close to zero when the noising schedule is defined in such a way that $\alpha_T \approx 0$.

Furthermore, if $\alpha_0 \approx 1$ and $\boldsymbol{x}$ is discrete, then $\mathcal{L}_0$ is close to zero as well.

5.3.2 Discrete diffusion

Recent works have considered the discrete diffusion problem for text, image and audio data (Hoogeboom et al., 2021; Johnson et al., 2021; Yang et al., 2022). We follow here the setting proposed by Austin et al. (2021). It considers a data point x that belongs to one of d classes and $\boldsymbol{x} \in \mathbb{R}^d$ its one-hot encoding. The noise is now represented by transition matrices $(\boldsymbol{Q}^1, ..., \boldsymbol{Q}^T)$ such that $[\boldsymbol{Q}^t]_{ij}$ represents the probability of jumping from state i to state j: $q(z^t|z^{t-1}) = \boldsymbol{z}^{t-1}\boldsymbol{Q}^t$.

As the process is Markovian, the transition matrix from $\boldsymbol{x}$ to $\boldsymbol{z}^t$ reads $\bar{\boldsymbol{Q}}^t = \boldsymbol{Q}^1\boldsymbol{Q}^2...\boldsymbol{Q}^t$. As long as $\bar{\boldsymbol{Q}}^t$ is precomputed or has a closed-form expression, the noisy states $\boldsymbol{z}^t$ can be built from $\boldsymbol{x}$ using $q(z^t|x) = \boldsymbol{x}\bar{\boldsymbol{Q}}^t$ without having to apply noise recursively (Property 1). The posterior

distribution $q(z_{t-1}|z_t, x)$ can also be computed in closed-form using Bayes rule (Property 2):

$$q(z^{t-1}|z^t, x) \propto \boldsymbol{z}^t \, (\boldsymbol{Q}^t)' \odot \boldsymbol{x} \, \bar{\boldsymbol{Q}}^{t-1} \tag{5.10}$$

where $\odot$ denotes a pointwise product and $\boldsymbol{Q}'$ is the transpose of $\boldsymbol{Q}$.

Proof. By Bayes rule, we have:

$$q(z^{t-1}|z^t, x) \propto q(z^t|z^{t-1}, x) \, q(z^{t-1}|x)$$

Since the noise is Markovian, $q(z^t|z^{t-1}, x) = q(z^t|z^{t-1})$. A second application of Bayes rule gives $q(z^t|z^{t-1}) \propto q(z^{t-1}|z^t)q(z^t)$.

By writing the definition of $\boldsymbol{Q}^t$, we then observe that $q(z^{t-1}|z^t) = \boldsymbol{z}^t \, (\boldsymbol{Q}^t)'$. We also have $q(z^{t-1}|x) = \boldsymbol{x} \, \bar{\boldsymbol{Q}}^{t-1}$ by definition.

Finally, we observe that $q(z^t)$ does not depend on z^{t-1}. It can therefore be seen as a part of the normalization constant. By combining the terms, we have $q(z^{t-1}|z^t, x) \propto \boldsymbol{z}^t \, (\boldsymbol{Q}^t)' \odot \boldsymbol{x} \, \bar{\boldsymbol{Q}}^{t-1}$ as desired. □

Finally, the limit distribution of the noise model depends on the transition model. The simplest and most common one is a uniform transition (Hoogeboom et al., 2021; Austin et al., 2021; Yang et al., 2022) parametrized by $\boldsymbol{Q}^t = \alpha^t \boldsymbol{I} + (1 - \alpha^t)\mathbf{1}_d \mathbf{1}_d'/d$ with α^t transitioning from 1 to 0. When $\lim_{t\to\infty} \alpha^t = 0$, $q(z^t|x)$ converges to a uniform distribution independently of x (Property 3).

The above framework satisfies all three properties in a setting that is inherently discrete. However, while it has been applied successfully to several data modalities, graphs have unique challenges that need to be considered: they have varying sizes, permutation equivariance properties, and to this date no known tractable universal approximator. The models we propose in the next sections aim to address these specific challenges.

5.3.3 SE(3)-Equivariance with Diffusion Models

Molecules are dynamic entities that can undergo translation and rotation, and the arrangement of their atoms does not have a predetermined order. To effectively model molecules using generative models and avoid augmenting the data with random transformations, it is essential to ensure that the models are equivariant to these inherent symmetries.

To study equivariance with diffusion models, we first need to define invariant and equivariant distributions. In our setting, a conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$ is equivariant to the action of a

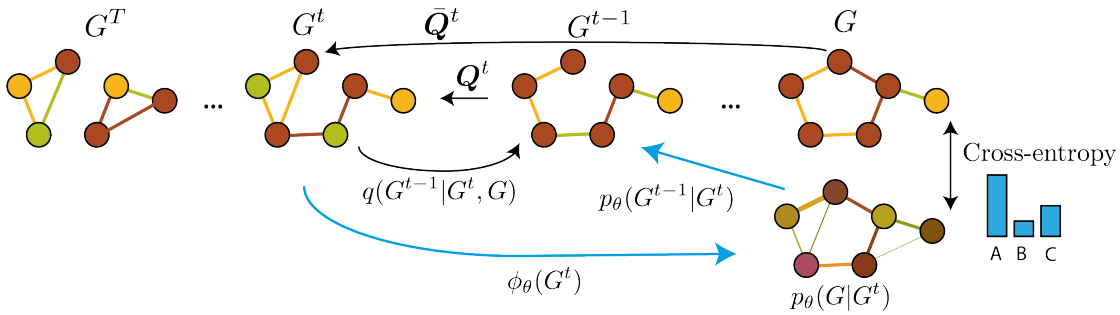


Figure 5.1 – Overview of DiGress. The noise model is defined by Markov transition matrices $\boldsymbol{Q}^t$ whose cumulative product is $\bar{\boldsymbol{Q}}^t$. The denoising network ϕ_θ learns to predict the clean graph from G^t. During inference, the predicted distribution is combined with $q(G^{t-1}|G, G^t)$ in order to compute $p_\theta(G^{t-1}|G^t)$ and sample a discrete G^{t-1} from this product of categorical distributions.

group $\mathbb{G}$ when

$$\forall g \in \mathbb{G},\ p(g.\boldsymbol{y}|g.\boldsymbol{x}) = p(\boldsymbol{y}|\boldsymbol{x}) \tag{5.11}$$

A distribution is invariant to the action of $\mathbb{G}$ if

$$\forall g \in \mathbb{G},\ p(g.\boldsymbol{y}) = p(\boldsymbol{y}) \tag{5.12}$$

In diffusion models, equivariance to a transformation group G can be achieved through several conditions. First, the noise model must be equivariant to the action of G: $\forall g \in G, q(g.z_t|g.x) = q(z_t|x)$. Second, the prior distribution q_∞ used at inference should be invariant to the group action, i.e., $q_\infty(g.z_T) = q_\infty(z_T)$, and this noise should be processed by an equivariant neural network in order to ensure that $p_\theta(g.z_{t-1}|g.z_t) = p_\theta(z_{t-1}|z_t)$. Finally, the network should be trained with a loss function that satisfies $l(p_\theta(g.x|g.z_t), g.x) = l(p_\theta(x|z_t), x)$. Together, these requirements create an architecture that is agnostic to the group elements used to represent the training data (Köhler et al., 2020; Xu et al., 2022).

5.4 DiGress: Discrete Denoising diffusion for graph generation

In this section, we present the Discrete Graph Denoising Diffusion model (DiGress) for graph generation. Our model handles graphs with categorical node and edge attributes, represented by the spaces $\mathbb{X}$ and $\mathbb{Y}$, respectively, with cardinalities d_x and d_y. We use x_i to denote the attribute of node i and $\boldsymbol{x}_i \in \mathbb{R}^a$ to denote its one-hot encoding. These encodings are organised in a matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d_x}$ where n is the number of nodes. Similarly, a tensor $\mathbf{Y} \in \mathbb{R}^{n \times n \times d_y}$ groups the one-hot encoding $\boldsymbol{y}_{ij}$ of each edge, treating the absence of edge as a particular edge type. We use $\boldsymbol{A}'$ to denote the matrix transpose of $\boldsymbol{A}$, while $\boldsymbol{A}^T$ is the value of $\boldsymbol{A}$ at time T.

5.4.1 Diffusion process and inverse denoising iterations

Similarly to diffusion models for images, which apply noise independently on each pixel, we diffuse separately on each node and edge feature. As a result, the state-space that we consider is not that of graphs (which would be too large to build a transition matrix), but only the space of node types $\mathbb{X}$ and edge types $\mathbb{Y}$. For any node (resp. edge), the transition probabilities are defined by the matrices $[\boldsymbol{Q}_X^t]_{ij} = q(x^t = j | x^{t-1} = i)$ and $[\boldsymbol{Q}_Y^t]_{ij} = q(y^t = j | y^{t-1} = i)$. Adding noise to form $G^t = (\boldsymbol{X}^t, \mathbf{Y}^t)$ simply means sampling each node and edge type from a categorical distribution defined by:

$$q(G^t | G^{t-1}) = (\boldsymbol{X}^{t-1}\boldsymbol{Q}_X^t, \mathbf{Y}^{t-1}\boldsymbol{Q}_Y^t) \quad \text{and} \quad q(G^t | G) = (\boldsymbol{X}\bar{\boldsymbol{Q}}_X^t, \mathbf{Y}\bar{\boldsymbol{Q}}_Y^t) \tag{5.13}$$

for $\bar{\boldsymbol{Q}}_X^t = \boldsymbol{Q}_X^1 ... \boldsymbol{Q}_X^t$ and $\bar{\boldsymbol{Q}}_Y^t = \boldsymbol{Q}_Y^1 ... \boldsymbol{Q}_Y^t$. When considering undirected graphs, we apply noise only to the upper-triangular part of $\mathbf{Y}$ and then symmetrize the matrix.

The second component of the DiGress model is the denoising neural network ϕ_θ parametrized by θ. It takes a noisy graph $G^t = (\boldsymbol{X}^t, \mathbf{Y}^t)$ as input and aims to predict the clean graph G, as illustrated in Figure 5.1. To train ϕ_θ, we optimize the cross-entropy loss l between the predicted probabilities $\hat{p}^G = (\hat{p}^X, \hat{p}^Y)$ for each node and edge and the true graph G:

$$l(\hat{p}^G, G) = \sum_{1 \le i \le n} \text{cross-entropy}(x_i, \hat{p}_i^X) + \lambda \sum_{1 \le i,j \le n} \text{cross-entropy}(y_{ij}, \hat{p}_{ij}^Y) \tag{5.14}$$

where $\lambda \in \mathbb{R}^+$ controls the relative importance of nodes and edges. It is noteworthy that, unlike architectures like VAEs which solve complex distribution learning problems that sometimes requires graph matching, our diffusion model simply solves classification tasks on each node and edge.

Once the network is trained, it can be used to sample new graphs. To do so, we need to estimate the reverse diffusion iterations $p_\theta(G^{t-1}|G^t)$ using the network prediction $\hat{p}^G$. We model this distribution as a product over nodes and edges:

$$p_\theta(G^{t-1}|G^t) = \prod_{1 \le i \le n} p_\theta(x_i^{t-1}|G^t) \prod_{1 \le i,j \le n} p_\theta(y_{ij}^{t-1}|G^t) \tag{5.15}$$

To compute each term, we marginalize over the network predictions:

$$p_\theta(x_i^{t-1}|G^t) = \int_{x_i} p_\theta(x_i^{t-1} \mid x_i, G^t)\, dp_\theta(x_i|G^t) = \sum_{x \in \mathcal{X}} p_\theta(x_i^{t-1} \mid x_i = x, G^t)\, \hat{p}_i^X(x)$$

where we choose

$$p_\theta(x_i^{t-1} \mid x_i = x,\ G^t) = \begin{cases} q(x_i^{t-1} \mid x_i = x,\ x_i^t) & \text{if } q(x_i^t | x_i = x) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{5.16}$$

Similarly, we have $p_\theta(y_{ij}^{t-1}|y_{ij}^t) = \sum_{y\in\mathbb{Y}} p_\theta(y_{ij}^{t-1} \mid y_{ij} = y, y_{ij}^t)\, \hat{p}_{ij}^Y(y)$. These distributions are used to sample a discrete G^{t-1} that will be the input of the denoising network at the next time step. These equations can also be used to compute an evidence lower bound on the likelihood, which allows for easy comparison between models. We now recall the corresponding computations.

Figure 5.2 – The graphical model of DiGress and ConGress.

The graphical model associated to our problem is presented in figure 5.2: the graph size is sampled from the training distribution and kept constant during diffusion. One can notice the similarity between this graphical model and hierarchical variational autoencoders (VAEs): diffusion models can in fact be interpreted as a particular instance of VAE where the encoder (i.e., the diffusion process) is fixed. The likelihood of a data point x under the model writes:

$$\log p_\theta(G) = \log \sum_{n\in\mathbb{N}} p(n) \int p(G^T|\, n)\, p_\theta(G^{t-1}, \dots, G^1|G^T)\, p_\theta(G|G^1)\, d(G^1, \dots, G^T) \tag{5.17}$$

$$= \log p(n_G) + \log \int p(G^T|\, n_G) \prod_{t=2}^{T} p_\theta(G^{t-1} \mid G^t)\, p_\theta(G|G^1)\, d(G^1, \dots, G^T) \tag{5.18}$$

As for VAEs, an evidence lower bound (ELBO) for this integral can be computed (Sohl-Dickstein et al., 2015; Kingma et al., 2021). It writes:

$$\log p_\theta(G) \geq \log p(n_G) + \underbrace{D_{\mathrm{KL}}[q(G^T|G) \,||\, q_X(n_G) \times q_Y(n_G)]}_{\text{Prior loss}} + \underbrace{\sum_{t=2}^{T} L_t(x)}_{\text{Diffusion loss}} + \underbrace{\mathbb{E}_{q(G^1|G)}[\log p_\theta(G|G^1)]}_{\text{Reconstruction loss}} \tag{5.19}$$

with

$$L_t(G) = \mathbb{E}_{q(G^t|G)}\big[D_{\mathrm{KL}}[q(G^{t-1}|G^t, G) \,||\, p_\theta(G^{t-1}|G^t)]\big] \tag{5.20}$$

All these terms can be estimated: $\log p(n_G)$ is computed using the frequencies of the number of nodes for each graph in the dataset. The prior loss and the diffusion loss are KL divergences between categorical distribution, and the reconstruction loss is simply computed from the predicted probabilities for the clean graph given the last noisy graph G^1.

5.4.2 Denoising network parametrization

The denoising network takes a noisy graph $G^t = (\boldsymbol{X}, \mathbf{Y})$ as input and outputs tensors $\boldsymbol{X}'$ and $\mathbf{Y}'$ which represent the predicted distribution over clean graphs. To efficiently store information, our

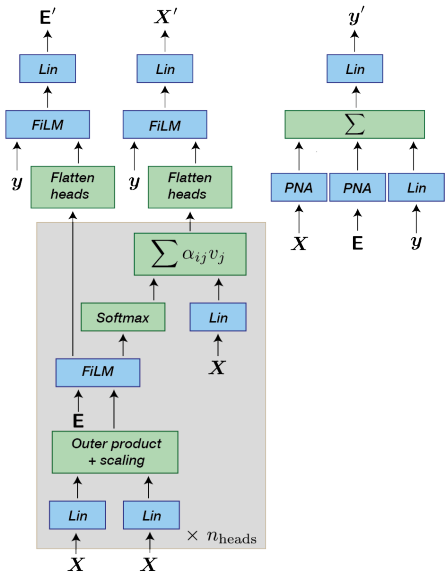


Figure 5.3 – The self-attention module of our graph transformer network. It takes as input node features $\boldsymbol{X}$, edge features $\mathbf{Y}$ and global features $\boldsymbol{y}$, and updates their representation. These features are then passed to normalization layers and a fully connected network, similarly to the standard transformer architecture. $\mathrm{FiLM}(\boldsymbol{M}_1, \boldsymbol{M}_2) = \boldsymbol{M}_1\boldsymbol{W}_1 + (\boldsymbol{M}_1\boldsymbol{W}_2) \odot \boldsymbol{M}_2 + \boldsymbol{M}_2$ for learnable weight matrices $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$, and $\mathrm{PNA}(\boldsymbol{X}) = \mathrm{cat}(\max(\boldsymbol{X}), \min(\boldsymbol{X}), \mathrm{mean}(\boldsymbol{X}), \mathrm{std}(\boldsymbol{X}))\, \boldsymbol{W}$.

layers also manipulate graph-level features $\boldsymbol{w}$. We chose to extend the graph transformer network proposed by Dwivedi & Bresson (2021), as attention mechanisms are a natural model for edge prediction. The proposed transformer layer is depicted in 5.3. At a high-level, it first updates node features using self-attention, incorporating edge features and global features using FiLM layers (Perez et al., 2018). The edge features are then updated using the unnormalized attention scores, and the graph-level features using pooled node and edge features. Our transformer layers also feature residual connections and layer normalization. To incorporate time information, we normalize the timestep to $[0, 1]$ and treat it as a global feature inside $\boldsymbol{w}$. The overall memory and time complexity of our network is $\Theta(n^2)$ per layer, due to the attention scores and the predictions for each edge.

5.4.3 Improving DiGress with marginal probabilities and structural features

Choice of the noise model

The choice of the Markov transition matrices $(\boldsymbol{Q}_t)_{t \leq T}$ defining the graph edit probabilities is arbitrary, and it is a priory not clear what noise model will lead to the best performance. A common model is a uniform transition over the classes $\boldsymbol{Q}^t = \alpha^t \boldsymbol{I} + (1 - \alpha^t)(\mathbf{1}_d \mathbf{1}_d')/d$, which leads to limit distributions q_X and q_Y that are uniform over categories. Graphs are however usually sparse, meaning that the marginal distribution of edge types is far from uniform. Starting from uniform noise, we observe in Figure 5.4 that it takes many diffusion steps for the model

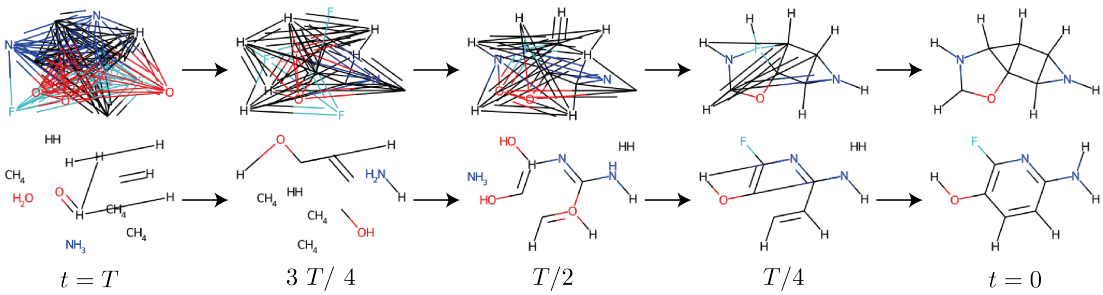


Figure 5.4 – Reverse diffusion chains generated from a model trained on uniform transition noise (top) and marginal noise (bottom). When noisy graphs have the right marginals of node and edge types, they are closer to realistic graphs, which makes training easier.

to produce a sparse graph. To improve upon uniform transitions, we propose the following hypothesis: *using a prior distribution which is close to the true data distribution makes training easier*.

This prior distribution cannot be chosen arbitrarily, as it needs to be permutation invariant to satisfy exchangeability. A natural model for this distribution is therefore a product $\prod_i u \times \prod_{i,j} v$ of a single distribution u for all nodes and a single distribution v for all edges. We propose the following result to guide the choice of u and v:

Theorem 3. *(Optimal prior distribution)*
Consider the class $\mathcal{C} = \{\prod_i u \times \prod_{i,j} v,\ (u, v) \in \mathcal{P}(\mathbb{X}) \times P(\mathbb{Y})\}$ of distributions over graphs that factorize as the product of a single distribution u over $\mathbb{X}$ for the nodes and a single distribution v over $\mathbb{Y}$ for the edges. Let P be an arbitrary distribution over graphs (seen as a tensor of order $n + n^2$) and m_X, m_Y its marginal distributions of node and edge types. Then $\pi^G = \prod_i m_X \times \prod_{i,j} m_Y$ is the orthogonal projection of P on $\mathcal{C}$:

$$\pi^G \in \underset{(u,v)\in\mathcal{C}}{\arg\min}\ ||\ P\ -\ \prod_{1\leq i\leq n} u \times \prod_{1\leq i,j\leq n} v||_2^2$$

Proof. We first prove the following result:

Lemma 5. *Let p be a discrete distribution over two variables. It is represented by a matrix $P \in \mathbb{R}^{a\times b}$. Let m^1 and m^2 the marginal distribution of p: $m_i^1 = \sum_{j=1}^b p_{ij}$ and $m_i^2 = \sum_{i=1}^a p_{ij}$. Then*

$$(m^1, m^2) \in \underset{\substack{u,v \\ u\geq 0, \sum u_i = 1 \\ v\geq 0, \sum v_j = 1}}{\arg\min}\ ||P - u\,v'||_2^2$$

Proof. We define $L(\boldsymbol{u}, \boldsymbol{v}) := ||P - uv'||_2^2 = \sum_{i,j}(p_{ij} - u_i v_j)^2$. We derive this formula to obtain

optimality conditions:

$$\begin{aligned}\frac{L}{\partial u_i} = 0 &\iff \sum_j (p_{ij} - u_i v_j) v_i = 0 \\ &\iff \sum_j p_{ij} v_j = u_i \sum_j v_j^2 \\ &\iff u_i = \sum_j p_{ij} v_j \;/\; \sum_j v_j^2\end{aligned}$$

Similarly, we have $\frac{\partial L}{\partial v_j} = 0 \iff v_j = \sum_i p_{ij} u_i \;/\; \sum_j u_i^2$.

Since p, u and v are probability distributions, we have $\sum_{i,j} p_{i,j} = 1$, $\sum_i u_i = 1$ and $\sum_j v_j = 1$. Combining these equations, we have:

$$\begin{aligned}u_i = \frac{\sum_j p_{ij} v_j}{\sum_j v_j^2} &\implies \sum_i u_i = 1 = \frac{\sum_{i,j} p_{ij} v_j}{\sum_j v_j^2} \\ &\iff \sum_j v_j^2 = \sum_j (\sum_i p_{ij}) v_j \\ &\iff \sum_j v_j^2 = \sum_j b_j v_j\end{aligned}$$

So that:

$$u_{i_0} = \frac{\sum_j p_{i_0 j} v_j}{\sum_j b_j v_j} = \frac{\sum_j p_{i_0 j} \frac{\sum_i p_{ij} u_i}{\sum_i a_i u_i}}{\sum_j b_j \frac{\sum_i p_{ij} u_i}{\sum_i a_i u_i}} = \sum_j p_{i_0 j} = m^1_{i_0}$$

and similarly $v_{j_0} = b_{i_0}$. Conversely, $\boldsymbol{m}^1$ and $\boldsymbol{m}^2$ belong to the set of feasible solutions. □

We have proved that the product distribution that is the closest to the true distribution of two variables is the product of marginals (for l_2 distance). We need to extend this result to a product $\prod_{i=1}^n u \times \prod_{1 \le i,j \le n} v$ of a distribution for nodes and a distribution for edges.

We now view p as a tensor in dimension $a^n b^{n^2}$. We denote p^X the marginalisation of this tensor across the node dimensions ($p^X \in \mathbb{R}^{a^n}$), and p^Y the marginalisation across the edge dimensions ($p^Y \in \mathbb{R}^{b^{n \times n}}$). By flattening the n first dimensions and the n^2 next, p can be viewed as a distribution over two variables (a distribution for the nodes and a distribution for the edges). By application of our Lemma, p^X and p^Y are the optimal approximation of p. However, p^X is a joint distribution for all nodes and not the product $\prod_{i=1}^n u$ of a single distribution for all nodes.

We then notice that:

$$
\begin{aligned}
||\prod_{i=1}^{n} u - p^X||_2^2 &= \sum_i ||u||^2 - 2\sum_i \langle u, p_i^X\rangle + \sum_i ||p_i^X||^2 \\
&= n\left(||u||^2 - 2\,\langle u, \frac{1}{n}\sum_i p_i^X\rangle + \frac{1}{n}\sum_i ||p_i^X||^2\right) \\
&= ||u - \frac{1}{n}\sum_i p_i^X||_2^2 + f(p^X)
\end{aligned}
$$

for a function f that does not depend on u. As $\sum_i p_i^X/n$ is exactly the empirical distribution of node types, the optimal u is the empirical distribution of node types as desired. Overall, we have made two orthogonal projections: a projection from the distributions over graphs to the distributions over nodes, and a projection from the distribution over nodes to the product distributions $u \times \cdots \times u$. Since the product distributions forms a linear space contained in the distributions over nodes, these two projections are equivalent to a single orthogonal projection from the distributions over graphs to the product distributions over nodes. A similar reasoning holds for edges. □

This result means that to get a prior distribution $q_X \times q_Y$ close to the true data distribution, we should define transition matrices such that $\forall i,\ \lim_{T\to\infty} \bar{\boldsymbol{Q}}_X^T \mathbb{1}_i = \boldsymbol{m}_X$ (and similarly for edges). To achieve this property, we propose to use

$$
\boldsymbol{Q}_X^t = \alpha^t \boldsymbol{I} + \beta^t\, \mathbf{1}_{d_x} \boldsymbol{m}_X' \quad \text{and} \quad \boldsymbol{Q}_Y^t = \alpha^t \boldsymbol{I} + \beta^t\, \mathbf{1}_{d_y} \boldsymbol{m}_Y' \tag{5.21}
$$

With this model, the probability of jumping from a state i to a state j is proportional to the marginal probability of category j in the training set. Since $(\mathbf{1}\boldsymbol{m}')^2 = \mathbf{1}\boldsymbol{m}'$, we still have $\bar{\boldsymbol{Q}}^t = \bar{\alpha}^t \boldsymbol{I} + \bar{\beta}^t \mathbf{1}\boldsymbol{m}'$ for $\bar{\alpha}^t = \prod_{\tau=1}^t \alpha^\tau$ and $\bar{\beta}^t = \prod_{\tau=1}^t (1-\alpha^\tau)$. We follow the popular cosine schedule $\bar{\alpha}^t = \cos\left(0.5\pi(t/T+s)/(1+s)\right)^2$ with a small s. Experimentally, these marginal transitions improves over uniform transitions (Appendix B.1).

Structural features augmentation

Generative models for graphs inherit the limitations of graph neural networks, and in particular their limited representation power (Xu et al., 2019; Morris et al., 2019). One example of this limitation is the difficulty for standard message passing networks (MPNNs) to detect simple substructures such as cycles (Chen et al., 2020), which raises concerns about their ability to accurately capture the properties of the data distribution. While more powerful networks have been proposed such as SMP in Chapter 3, they are slower to train and therefore not well suited to large-scale data. Another strategy to overcome this limitation is to augment standard MPNNs with features that they cannot compute on their own. For example, Bouritsas et al. (2022) proposed adding counts of substructures of interest, and Beaini et al. (2021) proposed adding spectral features, which are known to capture important properties of graphs (Chung & Graham, 1997).

DiGress operates on a discrete space and its noisy graphs are not complete, allowing for the computation of various graph descriptors at each diffusion step. These descriptors can be input to the network to aid in the denoising process, resulting in Algorithms 2 and 3 for training DiGress and sampling from it. The inclusion of these additional features experimentally improves performance, but they are not required for building a good model. The choice of which features to include and the computational complexity of their calculation should be considered, especially for larger graphs. In practice, the structural features that we use can be divided in two types: graph-theoretic (cycles and spectral features) and domain specific (molecular features).

Cycles Since message-passing neural networks are unable to detect cycles (Chen et al., 2020), we add cycle counts to our model. Because computing traversals would be impractical on GPUs (all the more as these features are recomputed at every diffusion step), we use formulas for cycles up to size 6. We build node features (how many $k-$cycles does this node belong to?) for up to 5-cycles, and graph-level features (how many $k-$cycles does this graph contain?) for up to $k = 6$. We use the following formulas, where $\boldsymbol{d}$ denotes the vector containing node degrees and $||.||_F$ is Frobenius norm:

$$
\begin{aligned}
\boldsymbol{X}_3 &= \operatorname{diag}(\boldsymbol{A}^3)/2 \\
\boldsymbol{X}_4 &= (\operatorname{diag}(\boldsymbol{A}^4) - \boldsymbol{d}(\boldsymbol{d}-1) - \boldsymbol{A}(\boldsymbol{d}\mathbf{1}_n^T)\mathbf{1}_n)/2 \\
\boldsymbol{X}_5 &= (\operatorname{diag}(\boldsymbol{A}^5) - 2\operatorname{diag}(\boldsymbol{A}^3) \odot \boldsymbol{d} - \boldsymbol{A}(\operatorname{diag}(\boldsymbol{A}^3)\mathbf{1}_n^T)\mathbf{1}_n + \operatorname{diag}(\boldsymbol{A}^3))/2 \\
\boldsymbol{y}_3 &= \boldsymbol{X}_3^T \mathbf{1}_n/3 \\
\boldsymbol{y}_4 &= \boldsymbol{X}_4^T \mathbf{1}_n/4 \\
\boldsymbol{y}_5 &= \boldsymbol{X}_5^T \mathbf{1}_n/5 \\
\boldsymbol{y}_6 &= \operatorname{Tr}(\boldsymbol{A}^6) - 3\operatorname{Tr}(\boldsymbol{A}^3 \odot \boldsymbol{A}^3) + 9||\boldsymbol{A}(\boldsymbol{A}^2 \odot \boldsymbol{A}^2)||_F - 6\,\langle \operatorname{diag}(A^2), \operatorname{diag}(A^4)\rangle \\
&\quad + 6\operatorname{Tr}(\boldsymbol{A}^4) - 4\operatorname{Tr}(\boldsymbol{A}^3) + 4\operatorname{Tr}(\boldsymbol{A}^2\dot{\boldsymbol{A}}^2 \odot \boldsymbol{A}^2) + 3||\boldsymbol{A}^3||_F - 12\operatorname{Tr}(\boldsymbol{A}^2 \odot \boldsymbol{A}^2) + 4\operatorname{Tr}(\boldsymbol{A}^2)
\end{aligned}
$$

Spectral features We also add the option to incorporate spectral features to the model. While this requires a $O(n^3)$ eigendecomposition, we find that it is not a limiting factor for the graphs that we use in our experiments (that have up to 200 nodes). We first compute some graph-level features that relate to the eigenvalues of the graph Laplacian: the number of connected components (given by the multiplicity of eigenvalue 0), as well as the 5 first nonzero eigenvalues. We then add node-level features relative to the graph eigenvectors: an estimation of the biggest connected component (using the eigenvectors associated to eigenvalue 0), as well as the two first eigenvectors associated to non zero eigenvalues.

5.4.4 Conditional generation

While good unconditional generation is a prerequisite, the ability to condition generation on graph-level properties is crucial for many applications. For example, in drug design, molecules that are easy to synthesize and have high activity on specific targets are of particular interest. One

Algorithm 2: Training DiGress

Input: A graph $G = (\boldsymbol{X}, \boldsymbol{E})$
Sample $t \sim \mathcal{U}(1, ..., T)$
Sample $G^t \sim \boldsymbol{X}\bar{\boldsymbol{Q}}^t_X \times \mathbf{Y}\bar{\boldsymbol{Q}}^t_Y$ ▷ Sample a (discrete) noisy graph
$z \leftarrow f(G^t, t)$ ▷ Structural and spectral features
$\hat{p}^X, \hat{p}^Y \leftarrow \phi_\theta(G^t, z)$ ▷ Forward pass
optimizer. step$(l_{CE}(\hat{p}^X, \boldsymbol{X}) + \lambda\, l_{CE}(\hat{p}^Y, \mathbf{Y}))$ ▷ Cross-entropy

Algorithm 3: Sampling from DiGress

Sample n from the training data distribution
Sample $G^T \sim q_X(n) \times q_Y(n)$ ▷ Random graph
for t = T **to** 1 **do**
  $z \leftarrow f(G^t, t)$ ▷ Structural and spectral features
  $\hat{p}^X, \hat{p}^Y \leftarrow \phi_\theta(G^t, z)$ ▷ Forward pass
  $p_\theta(x_i^{t-1}|G^t) \leftarrow \sum_x q(x_i^{t-1}|x_i = x, x_i^t)\, \hat{p}_i^X(x) \quad i \in 1, \dots, n$ ▷ Posterior
  $p_\theta(y_{ij}^{t-1}|G^t) \leftarrow \sum_y q(y_{ij}^{t-1}|y_{ij} = y, y_{ij}^t)\, \hat{p}_{ij}^Y(y) \quad i, j \in 1, \dots, n$
  $G^{t-1} \sim \prod_i p_\theta(x_i^{t-1}|G^t) \prod_{ij} p_\theta(y_{ij}^{t-1}|G^t)$ ▷ Categorical distr.
end
return G^0

way to perform conditional generation is to train the denoising network using the target properties (Hoogeboom et al., 2022), but it requires to retrain the model when the conditioning properties changes.

To overcome this limitation, we propose a new discrete guidance scheme inspired by the classifier guidance algorithm (Sohl-Dickstein et al., 2015). Our method uses a regressor g_η which is trained to predict target properties $\boldsymbol{w}_G$ of a clean graph G from a noisy version of G: $g_\eta(G^t) = \hat{\boldsymbol{w}}$.This regressor guides the unconditional diffusion model ϕ_θ by modulating the predicted distribution at each sampling step and pushing it towards graphs with the desired properties. The equations for the conditional denoising process are given by the following lemma:

Lemma 6. *(Conditional reverse noising process) (Dhariwal & Nichol, 2021)*
Denote $\dot{q}$ the noising process conditioned on $\boldsymbol{w}_G$, q the unconditional noising process, and assume that $\dot{q}(G^t|G, \boldsymbol{w}_G) = \dot{q}(G^t|G)$. Then we have $\dot{q}(G^{t-1}|G^t, \boldsymbol{w}_G) \propto q(G^{t-1}|G^t)\, \dot{q}(\boldsymbol{w}_G|G^{t-1})$.

While we would like to estimate $q(G^{t-1}|G^t)\, \dot{q}(\boldsymbol{w}_G|G^{t-1})$ by $p_\theta(G^{t-1}|G^t)\, p_\eta(\boldsymbol{w}_G|G^{t-1})$, p_η cannot be evaluated for all possible values of G^{t-1}. To overcome this issue, we view G as a continuous tensor of order $n + n^2$ (so that ∇_G can be defined) and use a first-order approximation.

Algorithm 4: Sampling from DiGress with discrete regressor guidance.

Input: Trained model ϕ_θ, property regressor g, target $\boldsymbol{w}$, guidance scale λ, graph size n
Sample $G^T \sim q_X(n) \times q_Y(n)$ ▷ `Random graph`
for $t = T$ **to** 1 **do**
  $z \leftarrow f(G^t, t)$ ▷ `Structural and spectral features`
  $\hat{p}^X, \hat{p}^Y \leftarrow \phi_\theta(G^t, z)$ ▷ `Forward pass`
  $\hat{\boldsymbol{w}} \leftarrow g_\eta(G^t)$ ▷ `Regressor model`
  $p_\eta(\hat{\boldsymbol{w}}|G^{t-1}) \propto \exp(-\lambda \langle \nabla_{G^t} ||\hat{\boldsymbol{w}} - \boldsymbol{w}||^2, G^{t-1}\rangle)$ ▷ `Guidance distribution`
  Sample $G^{t-1} \sim p_\theta(G^{t-1}|G^t)\, p_\eta(\hat{\boldsymbol{w}}|G^{t-1})$ ▷ `Reverse process`
end
return G^0

It gives:

$$\log \dot{q}(\boldsymbol{w}_G|G^{t-1}) \approx \log \dot{q}(\boldsymbol{w}_G|G^t) + \langle \nabla_G \log \dot{q}(\boldsymbol{w}_G|G^t), G^{t-1} - G^t\rangle$$
$$\approx c(G^t) + \sum_{1\leq i\leq n} \langle \nabla_{x_i} \log \dot{q}(\boldsymbol{w}_G|G^t), \boldsymbol{x}_i^{t-1}\rangle + \sum_{1\leq i,j\leq n} \langle \nabla_{y_{ij}} \log \dot{q}(\boldsymbol{w}_G|G^t), \boldsymbol{y}_{ij}^{t-1}\rangle$$

for a function c that does not depend on G^{t-1}. We make the additional assumption that $\dot{q}(\boldsymbol{w}_G|G^t) = \mathcal{N}(g(G^t), \sigma_w^2 \boldsymbol{I})$, where g is estimated by g_η, so that $\nabla_{G^t} \log \dot{q}_\eta(\boldsymbol{w}|G^t) \propto -\nabla_{G^t}||\hat{\boldsymbol{w}} - \boldsymbol{w}_G||^2$. The resulting procedure is presented in Algorithm 4.

In addition to being conditioned on graph-level properties, our model can be used to extend an existing subgraph – a task called molecular scaffold extension in the drug discovery literature (Maziarz et al., 2022). In Appendix C.2, we explain how to do it and demonstrate it on a simple example.

5.4.5 Experiments

In our experiments, we compare the performance of DiGress against several state-of-the-art one-shot graph generation methods on both molecular and non-molecular benchmarks. We compare its performance against the Set2GraphVAE proposed in Chapter 4, SPECTRE (Martinkus et al., 2022), GraphNVP (Madhawa et al., 2019), GDSS (Jo et al., 2022), GraphRNN (You et al., 2018b), GRAN (Liao et al., 2019), JT-VAE (Jin et al., 2018), NAGVAE (Kwon et al., 2020) and GraphINVENT (Mercado et al., 2021). We also build Congress, a model that has the same denoising network as DiGress but Gaussian diffusion (Appendix C.1). Our results are presented without validity correction[2].

[2]Code is available at `github.com/cvignac/DiGress`.

General graph generation

Table 5.2 – Unconditional generation on SBM and planar graphs. VUN: valid, unique & novel graphs.

Model	Deg ↓	Clus ↓	Orb↓	V.U.N. ↑
Stochastic block model				
GraphRNN	6.9	1.7	3.1	5 %
GRAN	14.1	1.7	2.1	25%
GG-GAN	4.4	2.1	2.3	25%
SPECTRE	1.9	1.6	**1.6**	53%
ConGress	34.1	3.1	4.5	0%
DiGress	**1.6**	**1.5**	1.7	**74%**
Planar graphs				
GraphRNN	24.5	9.0	2508	0%
GRAN	3.5	1.4	1.8	0%
SPECTRE	2.5	2.5	2.4	25%
ConGress	23.8	8.8	2590	0%
DiGress	**1.4**	**1.2**	**1.7**	**75%**

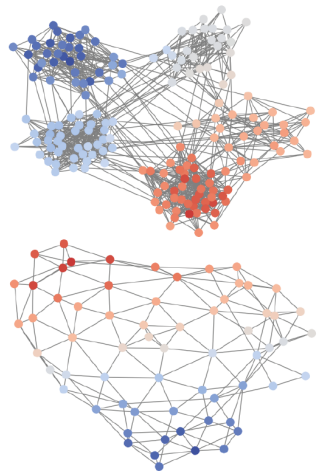


Figure 5.5 – Samples from DiGress trained on SBM and planar graphs.

We first evaluate DiGress on the benchmark proposed in Martinkus et al. (2022), which consists of two datasets of 200 graphs each: one drawn from the stochastic block model (with up to 200 nodes per graph), and another dataset of planar graphs (64 nodes per graph). We evaluate the ability to correctly model various properties of these graphs, such as whether the generated graphs are statistically distinguishable from the SBM model or if they are planar and connected. We refer to Appendix B.1 for a description of the metrics. In Table 5.2, we observe that DiGress is able to capture the data distribution very effectively, with significant improvements over baselines on planar graphs. In contrast, our continuous model, ConGress, performs poorly on these relatively large graphs.

Small molecule generation

We then evaluate our model on the standard QM9 dataset (Wu et al., 2018) that contains molecules with up to 9 heavy atoms. We use a split of 100k molecules for training, 20k for validation and 13k for evaluating likelihood on a test set. We report the negative log-likelihood of our model, validity (measured by RDKit sanitization) and uniqueness over 10k molecules. Novelty results are discussed in Appendix B.1. 95% confidence intervals are reported based on five runs. Results are presented in Figure 5.3. Since ConGress and DiGress both obtain close to perfect metrics on this dataset, we also perform an ablation study on a more challenging version of QM9 where hydrogens are explicitly modeled in Appendix B.1. It shows that the discrete framework is beneficial and that marginal transitions and auxiliary features further boost performance.

Table 5.3 – Molecule generation on QM9. Training time is the time needed to reach 99% validity. On small graphs, DiGress achieves similar results to the continuous model but is faster to train.

Method	NLL	Valid	Unique	Training time (h)
Dataset	–	99.3	100	–
Set2GraphVAE	–	59.9	93.8	–
SPECTRE	–	87.3	35.7	–
GraphNVP	–	83.1	99.2	–
GDSS	–	95.7	**98.5**	–
ConGress (ours)	–	98.9±.1	96.8±.2	7.2
DiGress (ours)	69.6±1.5	**99.0**±.1	96.2±.1	**1.0**

Table 5.4 – Molecule generation on MOSES. DiGress is the first one-shot graph model that scales to this dataset. While all graph-based methods except ours have hard-coded rules to ensure high validity, DiGress outperforms GraphInvent on most other metrics.

Model	Class	Val ↑	Unique↑	Novel↑	Filters↑	FCD↓	SNN↑	Scaf↑
VAE	SMILES	97.7	99.8	69.5	99.7	0.57	0.58	5.9
JT-VAE	Fragment	100	100	99.9	97.8	1.00	0.53	10
GraphINVENT	Autoreg.	96.4	99.8	–	95.0	1.22	0.54	12.7
ConGress (ours)	One-shot	83.4	99.9	96.4	94.8	1.48	0.50	16.4
DiGress (ours)	One-shot	85.7	100	95.0	97.1	1.19	0.52	14.8

Conditional generation experiments

To measure the ability of DiGress to condition the generation on graph-level properties, we propose a conditional generation setting on QM9. We sample 100 molecules from the test set and retrieve their dipole moment μ and the highest occupied molecular orbit (HOMO). The pairs (μ, HOMO) constitute the conditioning vector that we use to generate 10 molecules. To evaluate the ability of a model to condition correctly, we need to estimate the properties of the generated samples. To do so, we use RdKit (Landrum et al., 2006) to produce conformers of the generated graphs, and then Psi4 (Smith et al., 2020) to estimate the values of μ and HOMO. We report the mean absolute error between the targets and the estimated values for the generated molecules (Fig. 5.6).

Figure 5.6 – Mean absolute error on conditional generation with discrete regression guidance on QM9.

Target	μ	HOMO	μ & HOMO
Uncondit.	1.71±.04	0.93±.01	1.34±.01
Guidance	0.81±.04	0.56±.05	0.87±.03

Molecule generation at scale

We finally evaluate our model on two much more challenging datasets made of more than a million molecules: MOSES (Polykovskiy et al., 2020), which contains small drug-like molecules, and GuacaMol (Brown et al., 2019), which contains larger molecules. DiGress is to our knowledge

Table 5.5 – Molecule generation on GuacaMol. We report scores, so that higher is better for all metrics. While SMILES seem to be the most efficient molecular representation, DiGress is the first general graph generation method that achieves correct performance, as visible on the FCD score.

Model	Class	Valid↑	Unique↑	Novel↑	KL div↑	FCD↑
LSTM	Smiles	95.9	100	91.2	99.1	91.3
NAGVAE	One-shot	92.9	95.5	100	38.4	0.9
MCTS	One-shot	100	100	95.4	82.2	1.5
ConGress (ours)	One-shot	0.1	100	100	36.1	0.0
DiGress (ours)	One-shot	85.2	100	99.9	92.9	68.0

the first one-shot generative model that is not based on molecular fragments and that scales to datasets of this size. The metrics used as well as additional experiments are presented in App. C.3. For MOSES, the reported scores for FCD, SNN, and Scaffold similarity are computed on the dataset made of separate scaffolds, which measures the ability of the networks to predict new ring structures. Results are presented in Tables 5.4 and 5.5: they show that DiGress does not yet match the performance of SMILES and fragment-based methods, but performs on par with GraphInvent, an autoregressive model fine-tuned using chemical software and reinforcement learning. DiGress thus bridges the important gap between one-shot methods and autoregressive models that previously prevailed.

5.5 MiDi: Mixed Graph and 3D Denoising Diffusion for Molecule Generation

We now present the Mixed Graph+3D denoising diffusion (MiDi) model. We represent each molecule as a graph $G = (\boldsymbol{x}, \boldsymbol{c}, \boldsymbol{R}, \boldsymbol{Y})$, where $\boldsymbol{x}$ and $\boldsymbol{c}$ are vectors of length n containing the type of each atom and its formal charge, respectively. The $n \times 3$ matrix $\boldsymbol{R} = [\boldsymbol{r}_i]_{1 \leq i \leq n}$ contains the coordinates of each atom, and $\boldsymbol{Y}$ is an $n \times n$ matrix containing the bond types. Similarly to previous diffusion models for graphs, we consider the absence of a bond as a particular bond type and generate dense adjacency tensors. We denote the one-hot encoding of $\boldsymbol{x}$, $\boldsymbol{c}$, and $\boldsymbol{Y}$ by

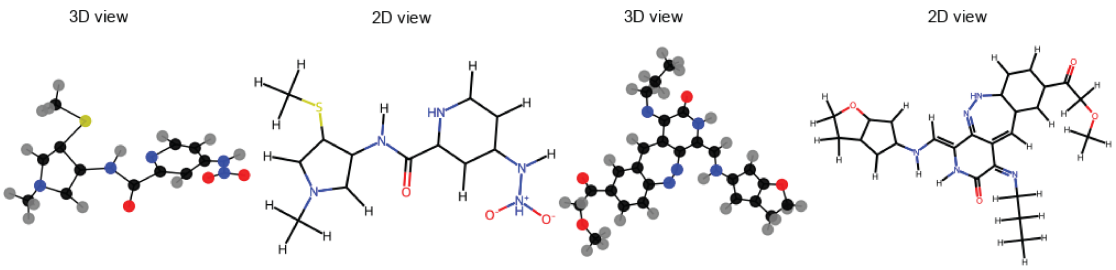


Figure 5.7 – Samples from our model. MiDi generates graphs embedded in 3D, therefore producing realistic 2D structures that are consistent with the 3D conformation.

$\boldsymbol{X}$, $\boldsymbol{C}$, and $\mathbf{Y}$, respectively. Time steps are denoted by superscripts, so, for example, $\boldsymbol{r}_i^t$ denotes the coordinates of atom i at time t. The transpose of matrix $\boldsymbol{X}$ is denoted by $\boldsymbol{X}'$.

5.5.1 Noise Model

Our noise model corrupts the features of each node and edge independently, using a noise model that depends on the data type. For the positions, we use a Gaussian noise within the zero center-of-mass (CoM) subspace of the molecule $\boldsymbol{\epsilon} \sim \mathcal{N}^{\text{CoM}}(\alpha^t \boldsymbol{R}^{t-1},\ (\sigma^t)^2 \boldsymbol{I})$, which is required to obtain a roto-translation equivariant architecture (Xu et al., 2022). This means that the noise follows a Gaussian distribution on the linear subspace of dimension $3(n-1)$ that satisfies $\sum_{i=1}^n \boldsymbol{\epsilon}_i = 0$.

For atom types, formal charges and bond types, we use discrete diffusion, where the noise model is a sequence of categorical distributions. We choose the marginal transition model proposed in DiGress. For instance, when $\boldsymbol{m} \in \mathbb{R}^a$ represents the marginal distribution of atom types in the training set, we define $\boldsymbol{Q}_x^t = \alpha^t \boldsymbol{I} + \beta^t\ \mathbf{1}_a \boldsymbol{m}'$. We similarly define $\boldsymbol{Q}_c^t$ and $\boldsymbol{Q}_y^t$. The resulting noise model is given by:

$$q(G^t|G^{t-1}) \sim \mathcal{N}^{\text{CoM}}(\alpha^t \boldsymbol{R}^{t-1},\ (\sigma^t)^2 \boldsymbol{I}) \times \mathcal{C}(\boldsymbol{X}^{t-1}\boldsymbol{Q}_x^t) \times \mathcal{C}(\boldsymbol{C}^{t-1}\boldsymbol{Q}_c^t) \times \mathcal{C}(\mathbf{Y}^{t-1}\boldsymbol{Q}_y^t).$$

When generating new samples, we define the posterior as a product as well:

$$p_\theta(G^{t-1}|G^t) = \prod_{1\le i\le n} p_\theta(\boldsymbol{r}_i^{t-1}|G^t) p_\theta(\boldsymbol{x}_i^{t-1}|G^t) p_\theta(\boldsymbol{h}_i^{t-1}|G^t) \prod_{1\le i,j\le n} p_\theta(\mathbf{Y}_{ij}^{t-1}|G^t),$$

We calculate each term by marginalizing over the network predictions. For instance,

$$\begin{aligned} p_\theta(x_i^{t-1}|G^t) &= \int_{\boldsymbol{x}_i} p_\theta(x_i^{t-1} \mid x_i, G^t)\ dp_\theta(x_i|G^t) \\ &= \sum_{x\in\mathcal{X}} q(x_i^{t-1}|x_i = x, G^t)\ p_\theta^X(x_i = x), \end{aligned}$$

where $p_\theta^X(x_i = x)$ is the neural network estimate for the probability that node v_i in the clean graph G is of type x.

Adaptive Noise Schedule

Although the MiDi model corrupts the coordinates, atom types, bond types and formal charges simultaneously, these components do not play a symmetrical role. For instance, while the 2D connectivity structure can be predicted relatively well from the 3D conformation, the converse is not true as the conformation is not unique for a given structure. Similarly, the formal charges serve as an adjustable variable used to match the valency of each atom with its electronic structure, but they do not constitute a very fundamental property of the molecules.

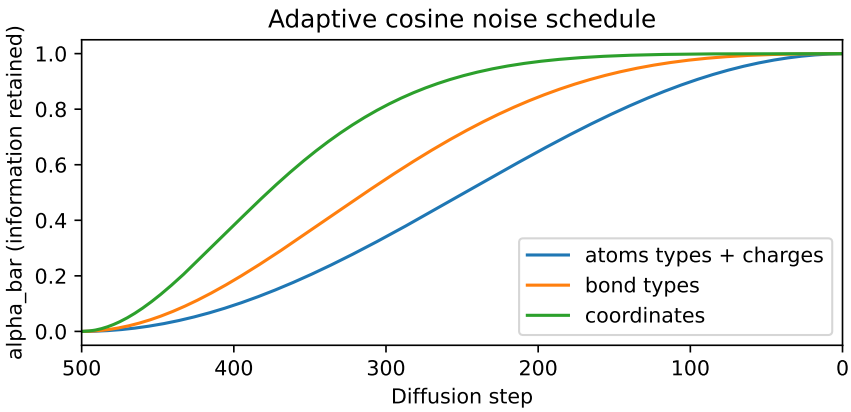


Figure 5.8 – The noise schedule is tuned separately for each component. The atom coordinates and bond types are denoised earlier during sampling, while the atom types and formal charges are mostly tuned afterwards.

Based on these observations, we propose an adaptation of the noise model in order to encourage the denoising network to first generate correctly the most important components, namely the atom coordinates and bond types, before moving on to predict the atom types and formal charges. To achieve this, we modify the noise schedule to vary according to the component. We modify the popular cosine schedule by adding an exponent ν that controls the rate at which the noise is added to the model:

$$\bar{\alpha}^t = \cos\left(\frac{\pi}{2}\frac{(t/T+s)^\nu}{1+s}\right)^2$$

where the parameter ν can take the form of ν_r, ν_x, ν_y and ν_c for the atom coordinates, types, bond types, and charges, respectively. By tuning ν on the QM9 dataset, we find that the following configuration works best: $\nu_r = 2.5,\ \nu_y = 1.5,\ \nu_x = \nu_c = 1$. The corresponding noise schedule is shown in Fig. 5.8. This choice implies that rough estimates for the atom coordinates and the bond types are first generated during inference, before the other components start to play a significant role. This aligns with previous work on 2D molecular graph generation which found that predicting the bond types before the atom types is beneficial (Madhawa et al., 2019), which we also observed in Chapter 4.

5.5.2 Denoising Network

The denoising network takes a noisy graph as input and learns to predict the corresponding clean graph. It manipulates graph-level features $\boldsymbol{w}$, node coordinates $\boldsymbol{R}$, node features (atom types and formal charges, treated together in the matrix $\boldsymbol{X}$), and edge features $\mathbf{Y}$. Coordinates are treated separately from the other node features in order to guarantee SE(3) equivariance. The neural network architecture is summarized in Figure 5.9. It consists of a Transformer architecture (Vaswani et al., 2017), with a succession of self-attention module followed by normalization layers and feedforward networks. We give more details about the different blocks below.

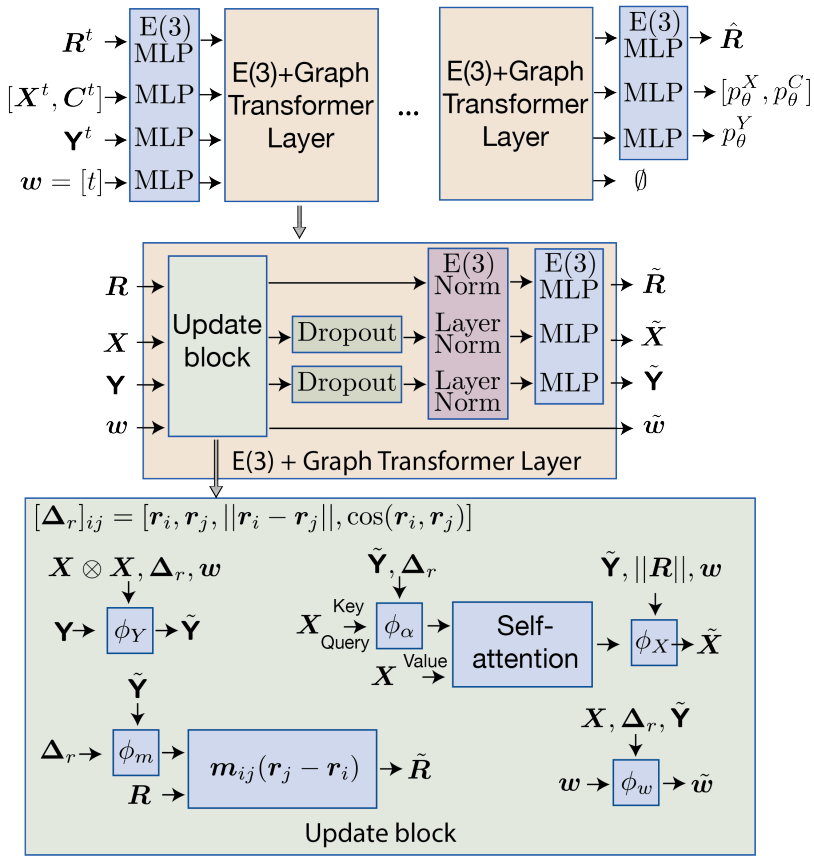


Figure 5.9 – The denoising neural network of MiDi jointly predicts the 2D graph and 3D coordinates of the clean graph from a noisy input. It follows a graph Transformer architecture with layers tailored to maintain SE(3) equivariance. In the update block, each component is updated using the other features. While the graph-level features $\boldsymbol{w}$ do not play a direct role in the final prediction, they serve as an effective means of storing and organizing pertinent information throughout the transformer layers.

Relaxed Equivariant Graph Neural Networks (rEGNNs)

In our proposed method, we leverage the effective yet affordable EGNN layers (Satorras et al., 2021b) for processing the coordinates. However, we enhance these layers by exploiting the fact that, when the data and the noise reside in the zero Center-Of-Mass subspace, it is not necessary for the neural network to be translation invariant. This can be interpreted as defining a canonical pose for the translation group, which is a valid way to achieve equivariance (Jaderberg et al., 2015; Kaba et al., 2022).

Rather than simply relying on pairwise distances $||\boldsymbol{r}_i - \boldsymbol{r}_j||_2$, we can therefore use other rotation invariant descriptors such as $||\boldsymbol{r}_i||_2$ or $\cos(\boldsymbol{r}_i, \boldsymbol{r}_j)$. We therefore propose the following

relaxedEGNN (rEGNN) layer:

$$[\boldsymbol{\Delta}_r]_{ij} = \mathrm{cat}(||\boldsymbol{r}_i - \boldsymbol{r}_j||_2, ||\boldsymbol{r}_i||_2, ||\boldsymbol{r}_j||_2, \cos(\boldsymbol{r}_i, \boldsymbol{r}_j))$$
$$\boldsymbol{r}_i \leftarrow \boldsymbol{r}_i + \sum_j \phi_m(\boldsymbol{X}_i, \boldsymbol{X}_j, [\boldsymbol{\Delta}_r]_{ij}, \mathbf{Y}_{ij})\,(\boldsymbol{r}_j - \boldsymbol{r}_i)$$

Similar to EGNN layers, the rEGNN layer combines a rotation-invariant message function with a linear update in $\boldsymbol{r}_j - \boldsymbol{r}_i$, which guarantees rotation equivariance. Notably, the additional features $||\boldsymbol{r}_i||_2$, $||\boldsymbol{r}_j||_2$, and $\cos(\boldsymbol{r}_i, \boldsymbol{r}_j)$ are computed relative to the center-of-mass of the molecule, which is set to 0 by definition. In our experiments, we have observed that these features facilitate the generation of a higher proportion of connected molecules, thereby mitigating an issue previously observed with both the EDM model and DiGress

Update Block To improve our model's ability to process node features, edge features, graph level features and coordinates, our new rEGNN layer is integrated into a larger update block. The edge features are first updated using $\boldsymbol{\Delta}_r$, the node features, and the global features. The node features are updated using a self-attention mechanism, where the attention coefficients also use the edge features and $\boldsymbol{\Delta}_r$. After the attention heads have been flattened, the obtained values are modulated by the pooled edge features, the norm of the coordinates and the global features. The global features are updated by pooling all other features at the graph level. Finally, the coordinates are updated using a rEGNN update, where the message function takes as input $\boldsymbol{\Delta}_r$ and the updated edge features $\mathbf{Y}'$. Note that we do not use the normalization term of EGNN: our layers are integrated in a Transformer architecture as discussed next, and we empirically found SE(3) normalization layers to be more effective than the EGNN normalisation term at controlling the magnitude of the activations.

Integration into a Transformer Architecture Transformers have proved to be a very efficient way to stabilize the self-attention mechanism over many layers. We describe below the changes to the feed-forward neural network and normalization layers that are required to ensure SE(3)-equivariance.

Our feed-forward neural network processes each component using MLPs applied in parallel on each node and each edge. As the coordinates cannot be treated separately (it would break SE(3)-equivariance), we define

$$\mathrm{PosMLP}(\boldsymbol{R}) = \Pi^{\mathrm{CoM}}(\mathrm{MLP}(||\boldsymbol{R}||)\frac{\boldsymbol{R}}{||\boldsymbol{R}|| + \delta}) \in \mathbb{R}^{n\times 3},$$

where $||\boldsymbol{R}|| \in \mathbb{R}^{ntimes1}$ contains the norm $||\boldsymbol{r}_i||^2$ of each point, $\mathrm{MLP}(||\boldsymbol{R}||) \in \mathbb{R}^{n\times 1}$ as well, δ is a small positive constant, and Π^{CoM} is the projection of the coordinates on the linear subspace

with center-of-mass at 0:

$$\Pi^{\text{CoM}}(\boldsymbol{R})_i = \boldsymbol{r}_i - \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{r}_i.$$

The choice of the normalization layer also depends on the problem symmetries: while batch normalization (Ioffe & Szegedy, 2015) is used in some graph transformer models (Dwivedi & Bresson, 2021), this layer is not equivariant in contrast to Set Normalization (Zhang et al., 2022) or Layer Normalization (Ba et al., 2016). For SE(3) equivariance, the normalization of (Liao & Smidt, 2022) should be used. Applied to 3D coordinates, it writes

$$\text{E3Norm}(\boldsymbol{R}) = \gamma \frac{||\boldsymbol{R}||}{\bar{n}+\delta} \frac{\boldsymbol{R}}{||\boldsymbol{R}||} = \gamma \frac{\boldsymbol{R}}{\bar{n}+\delta} \quad \text{with} \quad \bar{n} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} ||\boldsymbol{r}_i||^2},$$

with a learnable parameter $\gamma \in \mathbb{R}$ initialized at 1.

Training Objective

The denoising network of MiDi is trained to predict the clean molecule from a noisy input G^t, which is reflected in the choice of loss function used during model training. The estimation of the coordinates $\boldsymbol{R}$ is a regression problem that can simply be solved with mean-squared error, whereas the prediction p_θ^X for the atom types, p_θ^C for the formal charges and p_θ^Y for the bond types corresponds to a classification problem which can be addressed through a cross-entropy loss (CE in the equations). Note that the network's position predictions result in pointwise estimates $\hat{\boldsymbol{R}}$, while for the other terms, the prediction is a distribution over classes. The final loss is a weighted sum of these components:

$$l(G, \hat{p}^G) = \lambda_r ||\hat{\boldsymbol{R}} - \boldsymbol{R}||^2 + \lambda_x \,\text{CE}(\boldsymbol{X}, p_\theta^X) + \lambda_c \,\text{CE}(\boldsymbol{C}, p_\theta^C) + \lambda_y \,\text{CE}(\mathbf{Y}, p_\theta^Y)$$

The (λ_i) were initially chosen in order to balance the contribution of each term and cross-validated starting from this initial value. Our final experiments use $\lambda_r = 3$, $\lambda_x = 0.4$, $\lambda_c = 1$, $\lambda_y = 2$.

5.5.3 Experiments

Settings

We evaluate MiDi's performance on unconditional molecule generation tasks. To the best of our knowledge, MiDi is the first method to generate both the graph structure and the conformer simultaneously, leaving no end-to-end differentiable method to compare to. We therefore compare MiDi to 3D models on top of which a bond predictor is applied. We consider two such predictors: either a simple lookup table, as used in Hoogeboom et al. (2022), or the optimization procedure of

OpenBabel[3] O'Boyle et al. (2011) used in other works such as Igashov et al. (2022); Schneuing et al. (2022). The latter algorithm optimizes the bond orders of neighboring atoms in order to create a valid molecule, removing all control on the generated graphs. In terms of dataset comparison, EDM Hoogeboom et al. (2022) was previously the only method that could scale up to the large GEOM-DRUGS dataset, so it is our only direct competitor in that case. For the QM9 dataset, we also compare MiDi's performance to that of the GSchNet method (Gebauer et al., 2019), which employed the OpenBabel algorithm and achieved good results.

To facilitate comparison with previous methods, such as (Satorras et al., 2021a) and Hoogeboom et al. (2022), we benchmark our models on the full molecular graphs that include explicit hydrogens atoms. However, we acknowledge that, for most practical applications, hydrogen atoms can be inferred from the heavy atoms in the structure, and thus can be removed. In fact, methods trained solely on heavy atoms usually perform better since they consider smaller graphs.

We measure validity using the success rate of RDKit sanitization over 10,000 molecules. Uniqueness is the proportion of valid molecules with different canonical SMILES. Atom and molecule stability are metrics proposed in Satorras et al. (2021a) – they are similar to validity but, in contrast to RDKit sanitization, they do not allow for adding implicit hydrogens to satisfy the valency constraints. Novelty is the proportion of unique canonical SMILES strings obtained that are not in the training set. Since all the molecules that we consider have a single connected component, we also measure the proportion of the generated molecules that are connected.

We also compare the histograms of several properties of the generated set with a test set. The atom and bond total variations (AtomTV and BondTV) measure the l1 distance between the marginal distribution of atom types and bond types, respectively, in the generated set and test set. The Wasserstein distance between valencies is a weighted sum over the valency distributions for each atom types: $\text{ValencyW}_1 = \sum_{x \in \text{atom types}} p(x) \mathcal{W}_1(\hat{D}_{\text{val}}(x), D_{\text{val}}(x))$, where $p^X(x)$ is the marginal distribution of atom types in the training set, $\hat{D}_{\text{val}}(x)$ is the marginal distribution of valencies for atoms of type x in the generated set, $D_{\text{val}}(x)$ the same distribution in the test set. Here, the Wasserstein distance between histograms is used rather than total variation, as it allows to better respect the structure of ordinal data.

In previous methods, graph-based metrics were predominantly used. However, in our approach, we also introduce 3D metrics based on histograms of bond lengths and bond angles. This allows us to evaluate the efficacy of our approach not only in terms of the graph structure but also in generating accurate conformers. To this end, we report a weighted sum of the distance between bond lengths for each bond type:

$$\text{BondLenghtsW}_1 = \sum_{y \in \text{bond types}} p(y) \mathcal{W}_1(\hat{D}_{\text{dist}}(y), D_{\text{dist}}(y)),$$

where $p^Y(y)$ is the proportion of bonds of type y in the training set, $\hat{D}_{\text{dist}}(y)$ is the generated

[3] http://openbabel.org/wiki/Bond_Orders

Table 5.6 – Unconditional generation on QM9 with explicit hydrogens with uniform and adaptive noise schedules. While MiDi outperforms the base EDM model on graph-based metrics, the Open Babel optimization procedure is very effective on this simple dataset, as the structures are simple enough for the bonds to be determined unambiguously from the conformation.

Metrics (↑)	Mol stable	At stable	Validity	Uniqueness	Novelty	Connected
Data	98.7	99.8	98.9	99.9	–	100.0
GSchNet	92.0	98.7	98.1	94.5	**80.5**	97.1
EDM	90.7	99.2	91.7	**98.5**	76.0	99.3
EDM + OBabel	**97.9**	**99.9**	**99.0**	**98.5**	77.8	**99.7**
MiDi (uniform)	96.1±.2	**99.7**±.0	96.6±.2	97.6±.1	64.9±.5	**99.8**±.0
MiDi (adaptive)	**97.5**±.1	**99.8**±.0	97.9±.1	97.6±.1	67.5±.3	**99.9**±.0
Metrics (↓)	Valency(e-2)	Atom(e-2)	Bond(e-2)	Angles	Bond Lengths (e-2)	
Data	0.1	0.3	~ 0	0.12	~ 0	
GSchNet	4.9	4.2	1.1	1.68	0.5	
EDM	1.1	2.1	0.2	**0.44**	**0.1**	
EDM + OBabel	1.1	2.1	**0.1**	**0.44**	**0.1**	
MiDi (uniform)	0.4±.0	0.9±.0	**0.1**±0.0	0.67±.02	1.6±.7	
MiDi (adaptive)	**0.3**±.0	**0.3**±.1	**0.0**±.0	0.62±.02	0.3±.1	

distribution of bond lengths for bond of type y, and $D_{\text{dist}}(y)$ is the same distribution computed over the test set. The output is value in Angstrom.

Finally, BondAnglesW_1 (in degrees) compares the distribution of bond angles (in degrees) for each atom type. We compute a weighted sum of these values using the proportion of each atom type in the dataset. This calculation is restricted to atoms with two or more neighbors to ensure that angles can be defined:

$$\text{BondAnglesW}_1(\text{generated}, \text{target}) = \sum_{x \in \text{atom types}} \tilde{p}(x) \mathcal{W}_1(\hat{D}_{\text{angles}}(x), D_{\text{angles}}(x)),$$

where $\tilde{p}^X(x)$ denotes the proportion of atoms of type x in the training set, restricted to atoms with two neighbors or more, and $D_{\text{angles}}(x)$ is the distribution of geometric angles of the form $\angle(\boldsymbol{r}_k - \boldsymbol{r}_i, \boldsymbol{r}_j - \boldsymbol{r}_i)$, where i is an atom of type x, and k and j are neighbors of i. The reported metrics are mean and 95% confidence intervals on 5 different samplings from the same checkpoint.

QM9

We first evaluate our model on the standard QM9 dataset Wu et al. (2018) containing molecules with up to 9 heavy atoms. We split the dataset into 100k molecules for training, 20k for validation, and 13k for testing. Results are presented in Table 5.6. The *data* line represents the results of the training set compared with the test set, while the other entries compare the generated molecules to the test molecules. As we observe in Table 5.6, predicting the bonds only from the inter-atomic

Table 5.7 – Unconditional generation on GEOM-Drugs with explicit hydrogens with uniform and adaptive noise schedules. EDM was previously the only method that scaled to this dataset. On this complex dataset, the benefits of an integrated models are very clear, as MiDi generates a much higher proportion of stable molecules. All models achieve 100% uniqueness rate over 10,000 molecules. For MiDi (adaptive), 95% confidence intervals are reported on five checkpoints of the same run.

2D metrics	Mol stable	At stable	Validity	AtomTV	BondTV	Valency W1
Data	99.9	99.9	99.8	0.001	0.025	0.001
EDM	5.5	92.9	34.8	0.212	0.049	0.112
EDM + OBabel	40.3	97.8	35.3	0.212	0.048	0.285
MiDi (uniform)	66.5	98.9	66.9	**0.060**	**0.024**	**0.032**
MiDi (adaptive)	**81.0**$\pm$1.4	**99.4**$\pm$.0	**71.4**$\pm$1.4	0.086$\pm$.009	**0.024**$\pm$.000	**0.032**$\pm$.002

3D metrics	Bond Lengths W1 (Å)	Bond Angles W1(deg)
Data	~ 0	0.05
EDM (3D)	**0.002**	6.23
EDM + O. Babel	**0.002**	6.42
MiDi (uniform)	0.006	5.42
MiDi (adaptive)	0.006$\pm$.003	**3.60**$\pm$.016

distances and atom types has limited performance. Therefore, MiDi outperforms EDM on 2D metrics, while obtaining similar 3D metrics for the generated conformers. It is worth noting that our list of allowed bonds is not identical to that used in Satorras et al. (2021a); Hoogeboom et al. (2022), which may explain why our results for EDM Hoogeboom et al. (2022) do not match those of the original paper perfectly. Nonetheless, the optimization algorithm of Open Babel performs very well on this dataset of simple molecules. As QM9 contains molecules with only up to 9 atoms, the molecular conformations are easy to understand and the bonds can be determined easily.

GEOM-DRUGS

We then assess our model on the much larger GEOM-DRUGS dataset (Axelrod & Gomez-Bombarelli, 2020) which comprises 430,000 drug-sized molecules with an average of 44 atoms and up to 181 atoms. As this dataset features drug-like compounds, it is therefore better suited for downstream applications than QM9. We split the dataset into 80% for training, 10% for validation, and 10% for testing. For each molecule, we extract the 5 lowest energy conformations to build the dataset. Results are presented in Table 5.7.

As this dataset contains molecules that are much more complex than those in QM9, the bonds in the molecules cannot be determined solely from pairwise distances. This explains why EDM, which performs relatively well on 3D-based metrics, produces very few valid and stable molecules. Furthermore, many structures in this dataset are too complex for the Open Babel

algorithm. While the latter achieves good atom stability, there is at least one invalid atom in most molecules, leading to low molecular stability. The advantages of an end-to-end model that generates both a graph structure and its conformation are evident on this dataset: MiDi not only generates better molecular graphs, but also predicts more realistic conformers that better reflect the real angles between molecular bonds.

5.6 Discussion

While most previous graph generation methods were limited to small point clouds and graphs, we have seen that the proposed architectures scale significantly better. In this discussion, we propose explanations to this phenomenon, but also describe limitations of our current models.

What makes denoising diffusion models superior to other architectures?

As shown in this study, denoising diffusion models demonstrate superiority over other architectures such as VAEs, GANs, and normalizing flows for generating large-scale graphs. In comparison to GANs, diffusion models exhibit a more stable training process since they maximize the evidence lower bound (ELBO) instead of using a min-max objective. This is very clear experimentally, as little hyper-parameter tuning is needed to make diffusion models work well. Additionally, diffusion models use an iterative denoising process that allows them to correct their own predictions, which is not possible with GANs. This feature is particularly useful for generating high-quality outputs.

In contrast to normalizing flows, such as those used in (Satorras et al., 2021a), diffusion models are more efficient thanks to their better training process. Normalizing flows require a computationally expensive change of variable formula, which requires to either constrain the function class they learn or rely on neural ordinary differential equations (Neural ODEs). While both diffusion models and Neural ODEs iteratively apply a neural network to approximate the data distribution, diffusion models are trained on a single time step, which allows for easy parallelization, whereas Neural ODEs apply the network recursively many times before back-propagating. As a result, training Neural ODEs is slower and more challenging, particularly for early time steps.

In summary, the success of denoising diffusion models for large-scale graph generation can be attributed to their stable training process, iterative denoising scheme, and efficient training compared to other generative models. However, it is important to note that the success of denoising diffusion models also comes at a cost: they require a large number of diffusion steps, which can make them computationally expensive for large graphs.

What makes denoising diffusion suited to equivariant learning

While diffusion models have demonstrated superiority over GANs on other data modalities such as images (Dhariwal & Nichol, 2021), the gap in performance is even more impressive for graphs. To understand why, we need to consider the specificity of permutation equivariant generation.

In contrast to the models used in previous chapter, denoising diffusion models maintain a fixed number of nodes throughout the diffusion process, preserving the identity of each node and obviating the need for graph matching in computing the loss function. This is a significant advantage, as no polynomial-time algorithm for graph matching is currently known. Furthermore, we have seen that encoding and decoding graphs in a permutation equivariant way is as difficult as graph canonization. In contrasts, denoising diffusion models only operate at the node and edge-level, therefore avoiding all the complications of graph pooling and up-sampling.

Moreover, the iterative denoising scheme employed in denoising diffusion models acts as a powerful symmetry-breaking mechanism. When working with graphs without continuous features, such as the graphs used in DiGress, automorphisms are likely to be present. This can be problematic for equivariant graph neural networks, as they cannot distinguish between automorphic nodes, potentially leading to label noise during training. However, this issue is mitigated in denoising diffusion models because node and edge types are repeatedly sampled throughout the diffusion process. While symmetries may be present at certain points in the process, they should eventually be broken by the sampling steps.

Scalability

One limitation of the proposed diffusion models is their $O(n^2)$ complexity, which hinders their scalability to graphs and point clouds with thousands of nodes. In the case of EDM, this limitation can be overcome by restricting the message-passing updates to neighboring nodes, rather than using a fully-connected graph as done in EDM. This avoids the need to compute predictions for all pairs of nodes.

However, for DiGress, the limitation is more severe. DiGress operates on dense tensors and computes predictions for all pairs of nodes. Without breaking permutation equivariance, it is difficult to restrict the message-passing computations, as there is no clear notion of locality. This makes it challenging to scale non-auto-regressive graph generation models, and to our knowledge, no sub-quadratic permutation equivariant model has been proposed.

In the case of MiDi, sub-quadratic models could be obtained more easily. While MiDi currently makes a prediction for all pairs of atoms, in practice, atoms that are far enough are disconnected. Therefore, there is no need to make these predictions, and a cutoff could be used to improve model scalability.

Internal symmetries

Although EDM and MiDi are both equivariant to SE(3) transformations, these transformations do not account for all possible symmetries of molecules. One prominent example is the existence of rotatable bonds, which allow parts of the molecule to rotate freely, leading to non-unique conformations that cannot be captured by SE(3) transformations alone. Incorporating internal symmetries in the model is a challenging task, as computing the locations of rotatable bonds requires information about both the connectivity structure and bond types. One possible approach is to first generate the graph structure and then generate the conformer afterwards, but this approach would break the overall differentiability of the model and limit its performance.

Conclusion

In summary, we have introduced in this chapter two diffusion models for graph generation. These models operate in a similar fashion: they are trained on corrupted data to predict the clean data, and sample new objects by iteratively denoising pure noise. At each step, these diffusion models require to marginalize over the network predictions in order to sample the next time step. This is only possible for some noise models, but is important for efficient training.

The noise models needs to be chosen depending on the data modality. In this chapter, we have shown that there is a clear advantage to using discrete diffusion for the generation of categorical features. Gaussian noise is well suited to continuous features, but it needs to be slightly adapted in the case of coordinates in order to account for SE(3) equivariance.

The network parametrization also depends on the data modality. For example, DiGress uses additional features to overcome the limited expressive power of graphs networks, while the MiDi model, which additionally has access to 3D coordinates, does not need such features.

Overall, these models outperform previous architectures by a large margin. In addition to the general benefits of denoising diffusion models, which define a very stable training procedure that is easy to parallelize, casting generation as an iterative denoising problem is particularly interesting for unordered data. In denoising diffusion models, there is no need to perform graph coarsening, graph pooling or graph up-sampling, which are three tasks that are difficult for existing models. The denoising networks simply operate at the node and edge level, which lifts the challenges of permutation equivariance.

6 Conclusion and future directions

6.1 Conclusion

This thesis has contributed to the development of deep generative models and graph neural networks by exploring the concept of permutation equivariance and its implications on graph representation learning, especially in applications related to point cloud and graph generation.

With the Structural Message-Passing model, we have shown how to build higher order message-passing schemes that have a better representation power without sacrificing on the inductive bias of MPNNs. In SMP, we have shown that it is not the message-passing scheme that limits the expressive power of graph networks, but rather the order of the tensors that are used to manipulate the data. Despite a representation power that is similar to other architectures, SMP is much more effective at learning structural information of graphs, which makes it well suited to tasks with few node attributes.

We have then investigated equivariance in set and graph generation tasks, showing the constraints that equivariance impose on the probabilistic decoder of different architectures. We have reviewed layers that map vectors to sets, and proposed a novel effective layer for this task.

Unfortunately, learning to generate graphs with a vector-shaped latent space is difficult, as it require to compress graphs into a single vector and learn to decode them. This problem does not exist in diffusion models, which perform node and edge-level predictions without compressing the data representation. We presented DiGress, a diffusion model for graph generation, and its extension MiDi for nodes that additionally have 3D coordinates. These models achieve state-of-the-art performance, outperforming previous architectures such as GANs or VAEs by a large margin. While previous methods could only consider very small graphs, drug-sized molecules can now be generated. This opens the way for many applications, particularly in drug discovery and computation biology. In addition, diffusion models for graphs are still at their infancy, and there are many avenues of research to develop these models further. We discuss below some future directions that we would like to explore in the future.

6.2 Future directions

6.2.1 Directions for graph representation learning

Understanding the inductive bias of graph neural networks

In Chapter 2, we have explored the relationship between the iterative propagation scheme of MPPNs and the Weisfeiler-Lehman test, which has allowed us to gain insights into the expressive power of MPNNs. However, understanding the effectiveness of different graph neural networks requires us to not only comprehend their expressive power but also their inductive bias. To this end, it may be more relevant to consider continuous Weisfeiler-Lehman iterations (Togninalli et al., 2019) rather than the standard Weisfeiler-Lehman test, which employs hash functions at each step. Although all MPNN architectures share the same discrete Weisfeiler-Lehman test, the variations in their aggregation factors and normalization terms could be captured by continuous Weisfeiler-Lehman iterations. By defining an embedding for each node in a graph, these iterations provide a means to compare the embeddings learned by different graph networks.

6.2.2 Graph generation extensions

Diffusion models with indefinite time horizons

The idea of applying iteratively a denoising neural network has been proposed outside of diffusion models. In the context of molecule generation, Mahmood et al. (2021) proposed a model that iteratively masks part of a molecule and learns to recover the masked part. However, this model lacks a fundamental property of diffusion models, which is the ability to parallelize the training on different time steps, which is key to good performance. Nevertheless, the idea of viewing the denoising network as an agent that updates a graph until it is satisfied with the result is very interesting. It calls for the development of denoising diffusion models that do not have a fixed horizon but can still be trained efficiently. These models would enable the diffusion process to continue until convergence, instead of being limited by a fixed number of steps, thus allowing for more flexibility and potentially better performance.

Scaling denoising diffusion models

As discussed in previous chapter, the ability to scale diffusion models to point clouds and graphs with thousands of points is crucial for many applications. Currently, the quadratic complexity of the proposed models limits their scalability. One way to achieve sub-quadratic generation is to develop a hierarchical diffusion model that utilizes graph coarsening and graph up-sampling operators such as those proposed in (Ying et al., 2018; Gao & Ji, 2019). The main challenge probably lies in the design of a noise model that compresses the clean graph into a smaller, noisy graph, while preserving the closed-form formula required for efficient training of diffusion models.

Diffusion models for particular classes of graphs

In certain applications, it may be necessary to generate only particular subclasses of graphs, such as trees or planar graphs. To achieve this, diffusion models with specialized architectures can be developed to exploit the unique properties of these graphs. By combining established results in graph theory with diffusion models, it may be feasible to identify the appropriate representation of the data, which ensures that all generated graphs conform to the desired properties, such as being trees or planar graphs. The development of such specialized architectures requires the integration of insights from various domains, such as graph theory, combinatorics, and geometry.

6.2.3 Directions for molecule generation

Fragment-based diffusion

In the context of molecule generation, the use of fragment-based diffusion models, such as those proposed in (Jin et al., 2020) and (Maziarz et al., 2022), has proved to be very effective. By focusing on substructures, these models can effectively generate drug-like molecules by combining only a few fragments. While these models generally utilize autoregressive architectures, it may be worthwhile to explore other types of models. For instance, discrete diffusion models could be a useful alternative, as some expert knowledge on the similarity between molecular fragments could be incorporated in the Markov transition matrices.

Molecule generation with rich representations

In recent years, there has been a growing interest in the development of large-scale generative models for molecules which can perform well on a range of tasks. While MiDi considers molecules represented by their 2D connectivity structure and 3D arrangement of atoms, other representations such as SMILES strings have proven to be effective for various molecular tasks. One potential avenue for further research is therefore to consider different data modalities simultaneously. In addition to SMILES, we could for example consider multiple conformers of a molecule and their corresponding energies, which would provide additional information for the generative models.

Dynamics modeling

While existing models for generating physical systems such as molecule in 3D assume that the systems have a unique stable state (up to symmetries), this is not the case in practice. For example, molecules in 3D should not be defined by one single conformation, but rather by a probability distribution over the conformations. Moreover, the true systems evolve in time, which is currently not taken into account. Designing models that do not output one single state is a challenging direction for the future. It probably requires to integrate some physical knowledge to reduce the

size of the search space.

6.2.4 General directions

In contrast to text or image data, we have yet to develop large-scale architectures for few-shot learning on graphs that can be fine-tuned. Such architectures would be particularly useful for datasets with limited data, such as those commonly encountered in medical applications. However, current graph neural network architectures are highly task-specific, and the choice of aggregation function can have significant ramifications on downstream performance. Thus, the development of more flexible architectures is a necessary precondition for the creation of large-scale graph models.

In summary, our research has contributed to the exciting and rapidly evolving field of graph machine learning. As we have discussed, this area encompasses a broad range of research directions, and, in addition to the topics discussed in this thesis, there are still many open questions to be addressed in self-supervised learning, knowledge graphs, and large-scale graphs. However, we are confident that the graph machine learning community will continue to make significant progress in these areas and beyond, and we are proud to have been a part of this thriving and dynamic research community.

A Structural message-passing

A.1 Proof of Theorem 2

We first present the formal version of the theorem:

Theorem 4 (Representation power – formal). *Consider the class S of simple graphs with n nodes, diameter at most Δ and degree at most d_{max}. We assume that these graphs have respectively d_x and d_y attributes on the nodes and the edges. There exists a permutation equivariant SMP network $f : \mathbb{R}^{n\times n} \mapsto \mathbb{R}^{n\times n\times c}$ of depth at most $\Delta + 1$ and width at most $2d_{max} + d_x + n\ d_y$ such that, for any two graphs G and G' in S with respective adjacency matrices, node and edge features $\boldsymbol{A}, \boldsymbol{X}, \mathbf{Y}$ and $\boldsymbol{A}', \boldsymbol{X}', \mathbf{Y}'$, the following statements hold for every $v_i \in V$ and $v_j \in V'$:*

- *If G and G' are not isomorphic, then for all $\pi \in \mathfrak{S}_n$,*

$$\mathbf{\Pi}^T f(\boldsymbol{A}, \boldsymbol{X}, \mathbf{Y})[i, :, :] \neq f(\boldsymbol{A}', \boldsymbol{X}', \mathbf{Y}')[j, :, :].$$

- *If G and G' are isomorphic, then for some $\pi \in \mathfrak{S}_n$ independent of i and j,*

$$\mathbf{\Pi}^T f(\boldsymbol{A}, \boldsymbol{X}, \mathbf{Y})[i, :, :] = f(\boldsymbol{A}', \boldsymbol{X}', \mathbf{Y}')[j, :, :].$$

The fact that embeddings produced by isomorphic graphs are permutations one of another is a consequence of equivariance, so we are left to prove the first point. To do so, we will first ignore the features and prove that there is an SMP that maps the initial one-hot encoding of each node to an embedding that allows to reconstruct the adjacency matrix. The case of attributed graphs and the statement of the theorem will then follow easily.

Consider a simple connected graph $G = (V, E)$. For any layer $l \in \mathbb{N}$ and node $v_i \in V$, we denote by $G_i^{(l)} = (V, E_i^{(l)})$ the graph with node set V and edge set

$$E_i^{(l)} = \{(v_p, v_q) \in E,\ d(v_i, v_p) \leq l,\ d(v_i, v_q) \leq l,\ d(v_i, v_p) + d(v_i, v_q) < 2l\}.$$

These edges correspond to the receptive field of node v_i after l layers of message-passing. We denote by $\boldsymbol{A}_i^{(l)}$ the adjacency matrix of $G_i^{(l)}$.

Warm up: nodes manipulate n x n matrices

To build intuition, it is useful to first consider the case where $\boldsymbol{U}_i$ are $n \times n$ matrices (rather than $n \times c$ as in SMP). In this setting, messages are $n \times n$ matrices as well. If the initial state of each node v_i is its one-hop neighbourhood ($\boldsymbol{U}_i^{(1)} = \boldsymbol{A}_i^{(1)}$), then each node can easily recover the full adjacency matrix by updating its internal state as follows:

$$\boldsymbol{U}_i^{(l+1)} = \max_{v_j \in N(v_i) \,\cup\, v_i} \{\boldsymbol{U}_j^{(l)}\}, \tag{A.1}$$

where the max is taken element-wise.

Lemma 7. *Recursion A.1 yields* $\boldsymbol{U}_i^{(l)} = \boldsymbol{A}_i^{(l)}$.

Proof. We prove the claim by induction. It is true by construction for $l = 1$. For the inductive step, suppose that $\boldsymbol{U}_i^{(l)} = \boldsymbol{A}_i^{(l)}$. Then,

$$\begin{aligned}
\boldsymbol{U}_i^{l+1}[p,q] = 1 \iff & \exists\, v_j \in \{N(v_i) \cup v_i\} \quad \text{such that} \quad \boldsymbol{A}_j^{(l)}[p,q] = 1 \\
\iff & (v_p, v_q) \in E \text{ and } \exists\, v_j \in \{N(v_i) \cup v_i\},\ d(v_j, v_p) \leq l,\ d(v_j, v_q) \leq l,\ d(v_j, v_p) + d(v_j, v_q) < 2l \\
\implies & (v_p, v_q) \in E,\ d(v_i, v_p) \leq l+1,\ d(v_i, v_q) \leq l+1,\ d(v_i, v_p) + d(v_i, v_q) < 2(l+1) \\
\implies & \boldsymbol{A}_i^{(1+1)}[p,q] = 1
\end{aligned}$$

Conversely, if $\boldsymbol{A}_i^{(l+1)}[p,q] = 1$, then there exists either a path of length l of the form $(v_i, v_j, \ldots, v_p)$ or $(v_i, v_j, \ldots, v_q)$. This node v_j will satisfy $\boldsymbol{U}_j^{(l)}[p,q] = 1$ and thus $\boldsymbol{U}_i^{(l+1)}[p,q] = 1$. □

It is an immediate consequence that, for every connected graph of diameter Δ, we have $\boldsymbol{U}_i^{(\Delta)} = \boldsymbol{A}$.

SMP: nodes manipulate node embeddings

We now shift to the case of SMP. We will start by proving that we can find an $n \times 2d_{\text{max}}$ embedding matrix (rather than $n \times n$) that still allows to reconstruct $\boldsymbol{A}_i^{(l)}$. For this purpose, we will use the following result:

Lemma 8 (Maehara & Rödl (1990))**.** *For any simple graph $G = (V, E)$ of n nodes and maximum degree d_{max}, there exists a unit-norm embedding of the nodes $\boldsymbol{X} \in \mathbb{R}^{n \times 2d_{max}}$ such that*

$$\forall (v_i, v_j) \in V^2,\ (v_i, v_j) \in E \iff \boldsymbol{X}_i \perp \boldsymbol{X}_j.$$

In the following we assume the perspective of some node $v_i \in V$. Let $\boldsymbol{U}_i^{(l)} \in \mathbb{R}^{n \times c_l}$ be the context of v_i. Further, write $\boldsymbol{u}_j^{(l)} = \boldsymbol{U}_i^{(l)}[j,:] \in \mathbb{R}^{c_l}$ to denote the embedding of v_j at layer l from the perspective of v_i. Note that, for simplicity, the index i is omitted.

Lemma 9. *There exists a sequence $(f_l)_{l \geq 1}$ of permutation equivariant SMP layers defining $\boldsymbol{U}_i^{(l+1)} = f^{(l+1)}(\boldsymbol{U}_i^{(l)}, \{\boldsymbol{U}_j^{(l)}\}_{j \in N(v_i)})$ such that $\boldsymbol{u}_j^{(l)} \perp \boldsymbol{u}_k^{(l)} \iff (v_j, v_k) \in E_i^{(l)}$ for every layer l and nodes $v_j, v_k \in V$. These functions do not depend on the choice of $v_i \in V$.*

Proof. We use an inductive argument. An initialization (layer $l = 1$), we have $\boldsymbol{U}_j^{(0)} = \boldsymbol{\delta}_j$ for every v_j. We need to prove that there exists $\boldsymbol{U}_i^{(1)} = f^{(1)}(\boldsymbol{U}_i^{(0)}, \{\boldsymbol{U}_j^{(0)}\}_{v_j \in N(v_i)})$ which satisfies

$$\forall (v_j, v_k) \in V^2,\ \boldsymbol{u}_j^{(1)} \perp \boldsymbol{u}_k^{(1)} \iff (v_j, v_k) \in E_i^{(1)}.$$

Rewritten in matrix form, it is sufficient to show that there exists $\boldsymbol{U}_i^{(1)}$ such that $\boldsymbol{U}_i^{(1)}(\boldsymbol{U}_i^{(1)})^\top = \mathbf{1}\mathbf{1}^\top - \boldsymbol{A}_i^{(1)}$, with $\mathbf{1}$ being the all-ones vector. $\boldsymbol{A}_i^{(1)}$ is the adjacency matrix of a star consisting of v_i at the center and all its d_i neighbors at the spokes. Further, it can be constructed in an equivariant manner from the layer's input as follows:

$$\boldsymbol{A}_i^{(1)} = \sum_{v_j \in N(v_i)} \boldsymbol{\delta}_i \boldsymbol{\delta}_j^\top + \sum_{v_j \in N(v_i)} \boldsymbol{\delta}_j \boldsymbol{\delta}_i^\top.$$

Since the rank of $\boldsymbol{A}_i^{(1)}$ is at most d_i (there are d_i non-zero rows), the rank of $\mathbf{1}\mathbf{1}^\top - \boldsymbol{A}_i^{(1)}$ is at most $d_i + 1 \leq 2d_i \leq 2d_{max}$. It directly follows that there exists a matrix $\boldsymbol{U}_i^{(1)}$ of dimension $n \times 2d_{max}$ which satisfies $\boldsymbol{U}_i^{(1)}(\boldsymbol{U}_i^{(1)})^\top = \mathbf{1}\mathbf{1}^\top - \boldsymbol{A}_i^{(1)}$. Further, as the construction of this matrix is based on the eigendecomposition of $\boldsymbol{A}_i^{(1)}$, it is permutation equivariant as desired.

Inductive step. According to the inductive hypothesis, we suppose that:

$$\boldsymbol{u}_j^{(l)} \perp \boldsymbol{u}_k^{(l)} \iff (v_j, v_k) \in E_i^{(l)} \text{ for all } v_j, v_k \in V$$

The function $f^{(l+1)}$ builds the embedding $\boldsymbol{U}_i^{(l+1)}$ from $(\boldsymbol{U}_i^{(l)}, \{\boldsymbol{U}_j^{(l)},\ v_j \in N(v_i)\})$ in three steps:

Step 1. Each node $v_j \in N(v_i)$ sends its embedding $\boldsymbol{U}_j^{(l)}$ to node v_i. This is done using the message function $m^{(l)}$.

Step 2. The aggregation function ϕ reconstructs the adjacency matrix $\boldsymbol{A}_j^{(l)}$ of $G_j^{(l)}$ from $\boldsymbol{U}_j^{(l)}$ for each $v_j \in N(v_i) \cup \{v_i\}$. This is done by testing orthogonality conditions, which is a permutation equivariant operation. Then, it computes $\boldsymbol{A}_i^{(l+1)}$ as in Lemma 7 using $\boldsymbol{A}_i^{(l+1)} = \max(\{\boldsymbol{A}_j^{(l+1)}\}_{v_j \in N(v_i) \cup \{v_i\}})$, with the maximum taken entry-wise.

Step 3. The update function $u^{(l)}$ constructs an embedding matrix $\boldsymbol{U}_i^{(l+1)} \in \mathbb{R}^{n \times 2d_{\max}}$ that allows to reconstruct $\boldsymbol{A}_i^{(l+1)}$ through orthogonality conditions. The existence of such an embedding is guaranteed by Lemma 1. This operation can be performed in a permutation equivariant manner by ensuring that the order of the rows of $\boldsymbol{U}_i^{(l+1)}$ is identical with that of $\boldsymbol{A}_i^{(l+1)}$.

Therefore, the constructed embedding matrix $\boldsymbol{U}_i^{(l+1)}$ satisfies

$$\boldsymbol{u}_j^{(l+1)} \perp \boldsymbol{u}_k^{(l+1)} \iff (v_j, v_k) \in E_i^{(l+1)} \text{ for all } v_j, v_k \in V$$

and the function $f^{(l+1)}$ is permutation equivariant (as a composition of equivariant functions). □

It is a direct corollary of Lemma 1 that, when the depth is at least as large as the graph diameter, such that $E_i^{(l)} = E$ for all v_i and the width is at least as large as $2d_{\text{max}}$, then there exist a permutation equivariant SMP $f = f^{(L)} \circ \ldots \circ f^{(1)}$ that induces an injective mapping from the adjacency matrix $\boldsymbol{A}$ to the local context $\boldsymbol{U}_i$ of each node v_i. As a result, given two graphs G and G', if there are two nodes $v_i \in V$ and $v'_j \in V'$ and a permutation $\pi \in \mathfrak{S}_n$ such that $\boldsymbol{U}_i^{(L)} = \boldsymbol{\pi}^T \boldsymbol{U}_j'^{(L)}$, then the orthogonality conditions will yield $\boldsymbol{A} = \boldsymbol{\pi}^T \boldsymbol{A}' \boldsymbol{\pi}$. The contraposition is that if two nodes belong to graphs that are not isomorphic, their embedding will belong to two different equivalence classes (i.e. they will be different even up to permutations).

Extension for attributed graphs

For attributed graphs, the reasoning is very similar: we are looking for a SMP network that maps the attributes to a set of local context matrices such that all the attributes of the graph can be recovered from the context matrix at any node. We treat the case of node and edge attributes separately:

Node attributes Using d_x extra channels in SMP is sufficient to create the desired embedding. We recall that the input to the SMP is a local context such that the i-th row of v_i contains $[1, \boldsymbol{x}_i]$, where $\boldsymbol{x}_i$ is the vector of attributes of v_i, while the other rows are zero. Ignoring the first entry of this vector (which was used to reconstruct the topology), we propose the following update rule:

$$\boldsymbol{U}_i^{(l+1)} = \boldsymbol{U}_{j_0}^{(l)} \quad \text{where} \quad j_0 = \text{argmax}(\{|\boldsymbol{U}_j^{(l)}|\}_{j \in \{v_i \cup N(v_i)\}}) \tag{A.2}$$

where the max is taken element-wise on each entry of the matrix. This function is simply an extension of the max aggregator that allows to replace the zeros of the local context by non zero values, even if they are negative. Using it, each node can progressively fill the rows corresponding to nodes that are more and more distant. With the assumption that the graph is connected, each node will eventually have access to all node features.

Edge attributes As each edge attribute can be seen as a $n \times n$ matrix, edges attributes are handled in a very similar way as the adjacency matrix of unattributed graphs. If nodes could send $n \times n$ matrices as messages, they would be able to recover all the edge features using the previous update rule (equation A.2). However, SMP manipulates embeddings that transform under the action of a permutation as $\pi \,.\, \boldsymbol{U}_i = \boldsymbol{\pi}^T \boldsymbol{U}_i$, whereas a $n \times n$ matrix $\boldsymbol{M}$ transforms

as $\pi \,.\, \boldsymbol{M} = \boldsymbol{\pi}^T \boldsymbol{M}\, \boldsymbol{\pi}$. As a result, we cannot directly pass the incomplete edge features as messages, and we need to embed them into a matrix that permutes in the right way.

The construction of a SMP that embeds the input to local contexts that allow to reconstruct an edge feature matrix $\boldsymbol{E}$ is the same as for the adjacency matrix, except for one difference: lemma 1, which was used to embed adjacency matrices into a smaller matrix cannot be used anymore, as it is specific to unweighted graphs. Therefore, we propose another way to embed each matrix $\boldsymbol{E}_i^{(l)}$ obtained at node v_i after l message passing layers:

- For undirected graphs, $\boldsymbol{E}_i^{(l)}$ is symmetric. We can therefore compute its eigendecomposition $\boldsymbol{E}_i^{(l)} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T$.
- We add a given value λ to the diagonal of $\boldsymbol{\Lambda}$ to make sure that all coefficients are non-negative.
- We compute the square root matrix $\boldsymbol{U} = \boldsymbol{V}(\boldsymbol{\Lambda} + \lambda \boldsymbol{I})^{1/2}$. This matrix permutes as desired under the action of a permutation: $\pi \,.\, \boldsymbol{U} = \boldsymbol{\pi}^T \boldsymbol{U}$. In addition, it allows to reconstruct the matrix $\boldsymbol{E}_i^{(l)} = \boldsymbol{U}\boldsymbol{U}^T$, so that it constitutes a valid embedding for the rest of the proof.

Note that the square root matrix permutes as desired, but that it does not compress the representation of $\boldsymbol{E}_i^{(l)}$: for each edge features, n additional channels are needed, so that a SMP should have $n \times d_y$ more channels to be able to reconstruct all edge features.

Conclusion

We have shown that there exists an SMP that satisfies the conditions of the theorem, and specifically, we demonstrated that each layer can be decomposed in a message, aggregation and update functions that should be able to internally manipulate $n \times n$ matrices in order to produce embeddings of size $n \times 2d_{\text{max}} + d_x + n\, d_y$.

The main assumption of our proof is that the aggregation and update functions can *exactly* compute any function of their input — this is impossible in practice. An extension of our argument to a universal approximation statement would entail substituting the aggregation and update functions by appropriate universal approximators. In particular, the aggregation function manipulates a set of $n \times c$ matrices, which can be represented as a $n \times n \times c$ tensor with some lines zeroed out. Some universal approximators of equivariant functions for these tensors are known (Keriven & Peyré, 2019), but they have large memory requirements. Therefore, proving that a given parametrization of an SMP can be used to approximately reconstruct the adjacency matrix hinges on the identification of a simple universal approximator of equivariant functions on $n \times n \times c$ tensors.

B Top-n creation

B.1 Details about the experimenal setting of Top-n

Set MNIST and CLEVR

Since we use existing code for this task, we refer to the respective papers (Zhang et al., 2019) and (Kosiorek & Kim, 2020) for details on the model used and the loss function. The code used for TSPN is not the original code (which is not available) but a reimplementation (not by one of the authors of the present paper). The reader will notice that our results for DSPN are approximately 3 times worse than in the original paper. The reason is that we fixed what we believe to be a bug in the implementation of Chamfer's loss in DSPN: a mean over channels was used instead of a sum, which explains this difference of a factor 3. We also note that the results for TSPN are around 3 times worse than the original paper. One possible reason could be that the authors of TSPN used the code of DSPN and had a similar bug.

For Top-n generation, we set the number of points in the reference set to twice the cardinality n of the generated sets. We also experimented with $n_0 = n$ which resulted in better performance for DSPN (with a Chamfer loss of 6.14 ± 0.56 e-5), but not for TSPN (16.07 ± 0.47). We observed that reducing the learning rate improves results for all methods: TSPN was therefore trained for 100 epochs with a learning rate of 5e-4, and DSPN with a learning rate of 1e-4 for 200 epochs. No other hyper-parameter was tuned.

Synthetic set generation

Dataset generation procedure Each set is created via rejection sampling: points are drawn iteratively from a uniform distribution within a bounding box in $\mathbb{R}^3$. The first point is always accepted, and the next ones are accepted only if they satisfy a predefined set of constraints: i) they are not closer to any other point than a given threshold *min-distance*. ii) they are connected to the rest of the set, i.e., have at least one neighbor than a distance *neighbor-distance* iii) they do not have too many neighbors. Our dataset is made of 2000 sets that have between 2 and 35

points, with 9 points per set on average. It is a simplification of real molecules in several aspects: there are only single bonds, angles between bonds are not constrained and the atom types are only defined by the valency, which reflects the fact that atoms with the same valency tend to play a similar role and be more interchangeable.

Model The set encoder is made of a 2-layer MLP, 3 transformer layers followed by a PNA global pooling layer (that computes the sum, mean, max and standard deviation over each channel) (Corso et al., 2020) and a 2-layer MLP. The decoder is made of a set generator followed by a linear layer, 3 transformer layers and a 2-layer MLP. We use residual connections when possible and batch normalization between each layer. The reference set contains 35 points.

We experimented with the two ways to sample the number of points presented in Section 4.2, but we found the results to be quite similar. We therefore opted sampling the number of points from the data distribution, which is the simplest method.

Loss function We use a standard variational autoencoder loss with a Wasserstein reconstruction term and two additional regularizers. Given an input set $\boldsymbol{X}$ and its reconstruction $\hat{\boldsymbol{X}}$, the loss can be written:

$$L(\boldsymbol{X}, \hat{\boldsymbol{X}}) = d_{W_2}(\boldsymbol{X}, \hat{\boldsymbol{X}}) + \lambda_1 \, KL(p(\boldsymbol{z} \mid \boldsymbol{X}), \mathcal{N}(0, \boldsymbol{I}_l)) + \lambda_2 reg_2\,(\hat{\boldsymbol{X}}) + \lambda_3 \, reg_3(\hat{\boldsymbol{X}})$$

where

$$reg_2(\boldsymbol{X}) = \sum_{1 \leq i < j \leq n} (d_0 - ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2)_+ \quad \text{with} \quad d_0 = 1$$

prevents atoms from being generated too close to each other. $reg_3(\boldsymbol{X})$ penalizes atoms that have either no neighbor, or a too large valency. It is computed in the following way:

- for each point i, compute $s^i = \text{sort}((d_{ij})_{j \leq n,\, j \neq i})$. This vector contains the sorted distances between i and all other points. Points that are at distance less than $d_1 =$ *neighbor-distance* from i are considered as its neighbours.
- Compute $l_1(i) = (s_0^i - d_1)_+$. This term penalizes atoms that have no neighbour.
- Compute $l_2(i) = \sum_{j=\textit{max-valency}}^{n-1} (d_1 - s_j^i)_+$. This term penalizes atoms that have too many neighbors.
- $reg_3(X)$ is defined as $\sum_{1 \leq i \leq n} l_1(i) + l_2(i)$.

Training details In order to train the model efficiently, mini-batches have to be used. This may not be easy when dealing with sets and graphs, since they do not have all the same shape.

Table B.1 – Train reconstruction error and valency loss in the generated sets over 5 runs for a modified version of our dataset, where the cardinality varies less across sets. We observe a tradeoff between reconstruction performance and generalization.

Reference points	11	13	15	20	30	50
$\mathcal{W}_2$ train loss	$\mathbf{0.75}\pm.04$	$0.78\pm.03$	$0.79\pm.04$	$0.87\pm.05$	$0.93\pm.04$	$1.03\pm.06$
Valency loss (e-1)	$2.8\pm.7$	2.2 ± 0.7	$2.1\pm.9$	2.4 ± 1.2	$\mathbf{1.6}\pm.2$	$2.8\pm.8$

To circumvent this issue, we reorganise the training data in order to ensure that all sets inside a mini-batch have the same size. At generation time, this method cannot be applied, so we simply generate sets one by one.

The optimizer is Adam with its default parameters. We use a learning rate of $2e^{-4}$ and a scheduler that halves it when reconstruction performance does not improve significantly after 750 epochs. Experimentally, we found the learning rate decay to be important to achieve good reconstruction.

We also run a study with different reference set sizes. For this purpose, we slightly modify our dataset so that each set has only up to 11 points (still with 9 points on average). The reason is that there is more flexibility in the choice of the reference size if the maximal size is not too large. By training a Top-n network with several reference set sizes, we obtain the results of Table B.1.

Molecular graph generation on QM9

Model Our encoder is a graph neural network is made of 3 message-passing layers followed by a PNA global pooling layer and a final MLP. For the decoder, we use a set creation method followed by transformer layers. The resulting representation is then processed by i) a Set2Graph layer (Serviansky et al., 2020) followed by two MLPs to generate edge probabilities and edge features a MLP which generates node features ii) a MLP that takes as input the set representation and the valencies predicted for each atom, and returns an atom type.

Graph matching and loss function As explained in Section 4.3, the loss function of the variational autoencoder should solve a graph matching problem, which is hard in general. Instead of using a proper graph matching method, we propose to use the atom types to perform an imperfect but much cheaper alignment between the target and the predicted molecules.

For both molecules, we compute a score for each atom i defined as:

$$s(i) = 10^5 \textit{atom-type}(i) + 10^4 \textit{num-edges}(i) + \sum_{j \in \mathcal{N}(i)} \textit{edge-type}(i,j) * \textit{atom-type(j)}$$

This score cannot differentiate between all atoms in each molecule, but it reduces drastically the number of permutations that can represent the same input. It is motivated by the fact that

empirically, we observe that our method quickly learns to reconstruct the molecular formula very well. Once they are all computed, we use these scores to sort the atoms reorder the adjacency matrix and the atom and edge types.

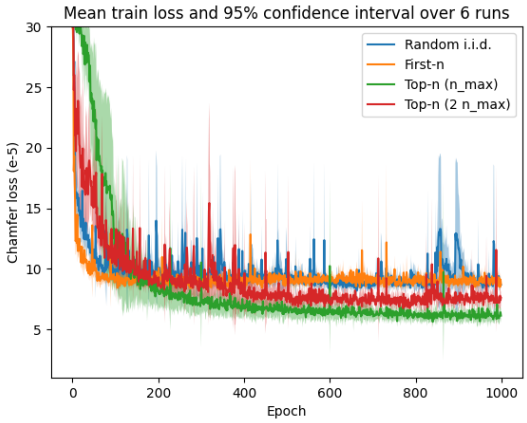
Our model learns to predict a probabilistic model for the atom types, edge presence and edge types. For this purpose, we use standard cross entropy loss between the predicted probabilities and the ground truth. However, these metrics can be hard to optimize because of the imperfect graph matching algorithm. We therefore regularize these metrics with several other measures at the graph level, that do not depend on matching:

- The mean squared error between the real atomic formula and the predicted one.
- The mean squared error between the average number of edges per atom in the input and predicted molecule.
- The mean squared error between the distribution of edge types in the real and predicted molecule.

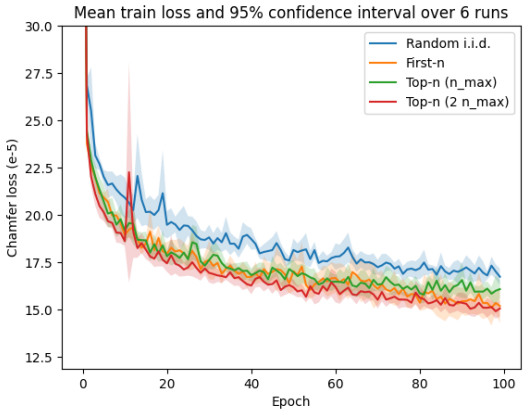
Finally, we add a matching dependent term, which is the mean squared error between the valencies of the input and the target molecule.

Training details The model is trained over 600 epochs with a batch size of 512 and a learning rate of $2e^{-3}$. It is halved after 100 epochs when the loss does not improve anymore. The optimizer is Adam with default parameters. The reference set has 12 points. When using more points, we obtained overall similar results, but with a larger variance.

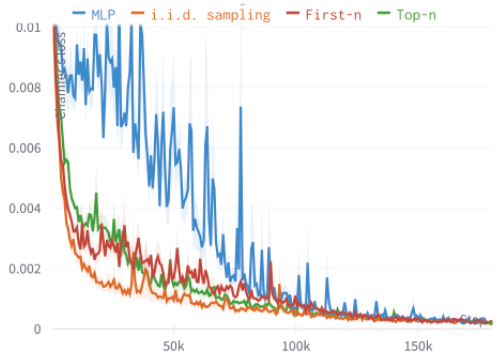
B.2 Training curves



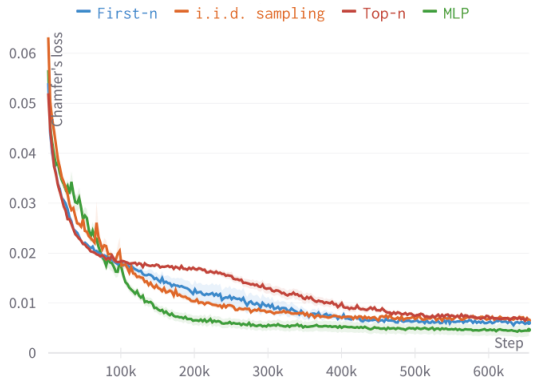
(a) DSPN training on Set-MNIST



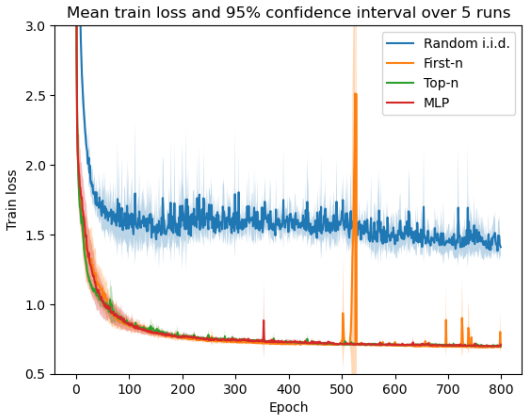
(b) TSPN training on Set-MNIST



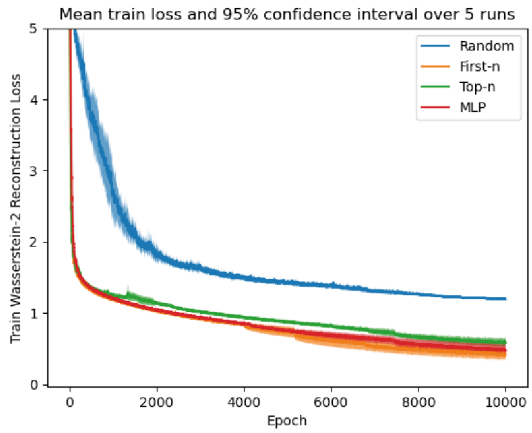
(c) DSPN training on CLEVR for bounding box prediction.



(d) DSPN training on Set-MNIST for full state prediction.



(e) Molecule generation on QM9. First-n, Top-n and MLP mostly overlap.



(f) Synthetic molecule-like dataset in 3d.

Figure B.1 – Training curves for all models. We observe that random i.i.d. generation is in general harder to train than the other models, while the differences between the other methods are smaller.

C DiGress

C.1 Continuous Graph Denoising Diffusion Model (ConGress)

In this section we present a diffusion model for graphs that uses Gaussian noise rather than a discrete diffusion process. Its denoising network is the same as the one of our discrete model. Our goal is to show that the better performance obtained with DiGress is not only due to the neural network design, but also to the discrete process itself.

Diffusion process

Consider a graph $G = (\boldsymbol{X}, \mathbf{Y})$. Similarly to the discrete diffusion model, this diffusion process adds noise independently on each node and each edge, but this time the noise considered is Gaussian:

$$q(\boldsymbol{X}^t|\boldsymbol{X}^{t-1}) = \mathcal{N}(\alpha^{t|t-1}\boldsymbol{X}^{t-1},\ (\sigma^{t|t-1})^2\mathbf{I}) \quad \text{and} \quad q(\mathbf{Y}^t|\mathbf{Y}^{t-1}) = \mathcal{N}(\alpha^{t|t-1}\mathbf{Y}^{t-1},\ (\sigma^{t|t-1})^2\mathbf{I}) \tag{C.1}$$

This process can equivalently be written:

$$q(\boldsymbol{X}^t|\boldsymbol{X}) = \mathcal{N}(\boldsymbol{X}^t|\alpha^t\boldsymbol{X}, \sigma^t\boldsymbol{I}) \qquad q(\mathbf{Y}^t|\mathbf{Y}) = \mathcal{N}(\mathbf{Y}^t|\alpha^t\mathbf{Y}, \sigma^t\mathbf{I}) \tag{C.2}$$

where $\alpha^{t|t-1} = \alpha^t/\alpha^{t-1}$ and $(\sigma^{t|t-1})^2 = (\sigma^t)^2 - (\alpha^{t|t-1})^2(\sigma^{t-1})^2$.

The variance is chosen as $(\sigma^t)^2 = 1 - (\alpha^t)^2$ in order to obtain a *variance-preserving* process (Kingma et al., 2021). Similarly to DiGress, when we consider undirected graphs, we only apply the noise on the upper-triangular part of $\mathbf{Y}$ without the main diagonal, and then symmetrize the matrix. The true denoising process can be computed in closed-form:

$$q(\boldsymbol{X}^{t-1}|\boldsymbol{X}, \boldsymbol{X}^t) = \mathcal{N}(\boldsymbol{\mu}^{t\to t-1}(\boldsymbol{X}, \boldsymbol{X}^t), (\sigma^{t\to t-1})^2\ \mathbf{I}) \quad \text{(and similarly for } \mathbf{Y}\text{)}, \tag{C.3}$$

Algorithm 5: Training ConGress

Input: A graph $G = (\boldsymbol{X}, \mathbf{Y})$
Sample $t \sim \mathcal{U}(1, ..., T)$
Sample $\epsilon_X \sim \mathcal{N}(0, \boldsymbol{I}_n)$
Sample $\epsilon_Y \sim \mathcal{N}(0, \boldsymbol{I}_{n(n-1)/2})$ and symmetrize if needed
$z^t \leftarrow \alpha^t(\boldsymbol{X}, \mathbf{Y}) + \sigma_t\,(\epsilon_X, \epsilon_Y)$ ▷ `Add noise`
Minimize $||(\epsilon_X, \epsilon_Y) - \phi_\theta(z^t, t)||^2$

Algorithm 6: Sampling from ConGress

Sample n from the training data distribution
Sample $\epsilon_X \sim \mathcal{N}(0, \boldsymbol{I}_n)$
Sample $\epsilon_Y \sim \mathcal{N}(0, \boldsymbol{I}_{n(n-1)/2})$ and symmetrize if needed
$z_T \leftarrow (\epsilon_X, \epsilon_Y)$
for t = T **to** 1 **do**
  Sample $\epsilon_X \sim \mathcal{N}(0, \boldsymbol{I}_n)$
  Sample and symmetrize $\epsilon_Y \sim \mathcal{N}(0, \boldsymbol{I}_{n(n-1)/2})$
  $z^{t-1} \leftarrow \frac{1}{\alpha_{t|t-1}} z^t - \frac{\sigma^2_{t|t-1}}{\alpha_{t|t-1}\sigma^t}\phi_\theta(z^t, t) + \sigma_{t\to t-1}(\epsilon_X, \epsilon_Y)$ ▷ `Reverse iterations`
end
return $\mathrm{argmax}(\boldsymbol{X}^0), \mathrm{argmax}(\mathbf{Y}^0)$

with

$$\boldsymbol{\mu}^{t\to t-1}(\boldsymbol{X}, \boldsymbol{X}^t) = \frac{\alpha_{t|t-1}\,(\sigma^{t-1})^2}{\sigma_t^2}\boldsymbol{X}^t + \frac{\alpha^{t-1}\,(\sigma^{t|t-1})^2}{(\sigma^t)^2}\boldsymbol{X} \quad \text{and} \quad \sigma^{t\to t-1} = \frac{\sigma_{t|t-1}\,\sigma_{t-1}}{\sigma_t}. \tag{C.4}$$

As commonly done for Gaussian diffusion models, we train the denoising network to predict the noise components $\hat{\epsilon}_X, \hat{\epsilon}_Y$ instead of $\hat{\boldsymbol{X}}$ and $\hat{\mathbf{Y}}$ themselves (Ho et al., 2020). Both relate as follows:

$$\alpha^t\,\hat{\boldsymbol{X}} = \boldsymbol{X}^t - \sigma^t\hat{\boldsymbol{\epsilon}}_X \quad \text{and} \quad \hat{\alpha}^t\mathbf{Y} = \mathbf{Y}^t - \sigma^t\,\hat{\boldsymbol{\epsilon}}_Y \tag{C.5}$$

To optimize the network, we minimize the mean squared error between the predicted noise and the true noise, which results in Algorithm 5 for training ConGress. Sampling is done similarly to standard Gaussian diffusion models, except for the last step: since continuous valued features are obtained, they must be mapped back to categorical values in order to obtain a discrete graph. For this purpose, we then take the argmax of $\boldsymbol{X}^0, \mathbf{Y}^0$ across node and edge types (Algorithm 6).

Overall, ConGress is very close to the GDSS model proposed in Jo et al. (2022), as it is also a Gaussian-based diffusion model for graphs. An important difference is that we define a diffusion process that is independent for each node and edge, while GDSS uses a more complex noise model that does not factorize. We observe empirically that a simple noise model does not hurt performance, since ConGress outperforms GDSS on QM9 (Table 4.5).

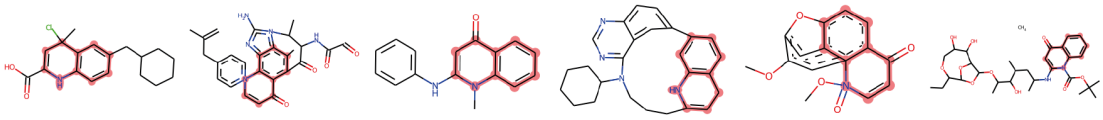


Figure C.1 – An example of molecular scaffold extension. We sometimes observe long-range consistency issues in the generated samples, which is in line with the observations of (Lugmayr et al., 2022) for image data. A resampling strategy similar to theirs could be used to solve this issue.

C.2 Substructure conditioned generation

Given a subgraph $S = (\boldsymbol{X}_S, \mathbf{Y}_S)$ with n_s nodes, we can condition the generation on S by masking the generated node and edge feature tensor at each reverse iteration step (Lugmayr et al., 2022). As our model is permutation equivariant, it does not matter what entries are masked: we therefore choose the first n_s ones. After sampling G^{t-1}, we update $\boldsymbol{X}$ and $\mathbf{Y}$ using

$$\boldsymbol{X}^{t-1} = \boldsymbol{M}_X \odot \boldsymbol{X}_s + (1 - \boldsymbol{M}_X) \odot \boldsymbol{X}^{t-1} \quad \text{and} \quad \mathbf{Y}^{t-1} = \mathbf{M}_Y \odot \boldsymbol{E}_s + (1 - \mathbf{M}_Y) \odot \mathbf{Y}^{t-1},$$

where $\boldsymbol{M}_X \in \mathbb{R}^{n \times a}$ and $\boldsymbol{M}_Y \in \mathbb{R}^{n \times n \times b}$ are masks indicating the n_s first nodes. In Figure C.1, we showcase an example for molecule generation: we follow the setting proposed by (Maziarz et al., 2022) and generate molecules starting from a particular motif called 1,4-Dihydroquinoline[1].

C.3 Experimental details and additional results

Abstract graph generation

Metrics The reported metrics compare the discrepancy between the distribution of some metrics on a test set and the distribution of the same metrics on a generated graph. The metrics measured are degree distributions, clustering coefficients, and orbit counts (it measures the distribution of all substructures of size 4). We do not report raw numbers but ratios computed as follows:

$$r = \mathrm{MMD}(\textit{generated}, \textit{test})^2 \;/\; \mathrm{MMD}(\textit{training}, \textit{test})^2$$

The denominator $\mathrm{MMD}(\textit{training}, \textit{test})^2$ is taken from the results table of SPECTRE (Martinkus et al., 2022). Note that what the authors report as MMD is actually MMD squared.

Community-20 In Table C.1, we also provide results for the smaller Community-20 dataset which contains 200 graphs drawn from a stochastic block model with two communities. We

[1] https://pubchem.ncbi.nlm.nih.gov/compound/1_4-Dihydroquinoline

Table C.1 – Results on the small Community-20 dataset.

	Degree↓	Clustering↓	Orbit↓	Ratio↓
GraphRNN	4.0	1.7	4.0	3.2
GRAN	3.0	1.6	1.0	1.9
GG-GAN	4.0	3.1	8.0	5.5
SPECTRE	0.5	2.7	2.0	1.7
DiGress	1.0	0.9	1.0	1.0

Table C.2 – Ablation study on QM9 with explicit hydrogens. Marginal transitions improve over uniform transitions, and spectral and structural features further boost performance.

Model	Valid↑	Unique↑	Atom stable↑	Mol stable↑
Dataset	97.8	100	98.5	87.0
ConGress	86.7±1.8	**98.4**±0.1	97.2±0.2	69.5±1.6
DiGress (uniform)	89.8±1.2	97.8±0.2	97.3±0.1	70.5±2.1
DiGress (marginal)	92.3±2.5	97.9±0.2	**97.3**±0.8	66.8±11.8
DiGress (marg. + features)	**95.4**±1.1	97.6±0.4	**98.1**±0.3	**79.8**±5.6

observe that DiGress performs very well on this small dataset.

QM9

Metrics Because it is the metric reported in most papers, the validity metric we report is computed by building a molecule with RdKit and trying to obtain a valid SMILES string out of it. As explained by Jo et al. (2022), this method is not perfect because QM9 contains some charged molecules which would be considered as invalid by this method. They thus compute validity using a more relaxed definition that allows for some partial charges, which gives them a small advantage.

Ablation study We perform an ablation study in order to highlight the role of marginal transitions and auxiliary features. In this setting, we also measure atom stability and molecule stability as defined in (Hoogeboom et al., 2022). Results are presented in Figure C.2.

Novelty As in Chapter 4 and don't report novelty for QM9 in the main table. The reason is that since QM9 is an exhaustive enumeration of the small molecules that satisfy a given set of constrains, generating molecules outside this set is not necessarily a good sign that the network has correctly captured the data distribution. For the interested reader, DiGress achieves on average a novelty of 33.4% on QM9 with implicit hydrogens, while ConGress obtains 40.0%.

MOSES and GuacaMol

Datasets For both MOSES and GuacaMol, we convert the generated graphs to SMILES using the code of Jo et al. (2022) that allows for some partial charges.

We note that GuacaMol contains complex molecules that are difficult to process, for example because they contain formal charges or fused rings. As a result, mapping the train smiles to a graph and then back to a train SMILES does not work for around 20% of the molecules. Even if our model is able to correctly model these graphs and generate graphs that are similar, these graphs cannot be mapped to SMILES strings to be evaluated by GuacaMol. More efficient tools for processing complex molecules as graphs are therefore needed to truly achieve good performance on this dataset.

Metrics Since MOSES and Guacamol are benchmarking tools, they come with their own set of metrics that we use to report the results. We briefly describe this metrics: Validity measures the proportion of molecules that pass basic valency checks. Uniqueness measures the proportion of molecules that have different SMILES strings (which implies that they are non-isomorphic). Novelty measures the proportion of generated molecules that are not in the training set. The filter score measures the proportion of molecules that pass the same filters that were used to build the test set. The Frechet ChemNetDistance (FCD) measures the similarity between molecules in the training set and in the test set using the embeddings learned by a neural network. SNN is the similarity to a nearest neighbor, as measured by Tanimoto distance. Scaffold similarity compares the frequencies of Bemis-Murcko scaffolds. The KL divergence compares the distribution of various physicochemical descriptors.

Likelihood Since other methods did not report likelihood for GuacaMol and MOSES, we did not include our NLL results in the table neither. We obtain a test NLL of 129.7 on QM9 with explicit hydrogens, 205.2 on MOSES (on the separate scaffold test set) and 308.1 on GuacaMol.

Size extrapolation While the vast majority of molecules in QM9 have the same number of atoms, molecules in MOSES and Guacamol have varying sizes. On these datasets, we would like to know if DiGress can generate larger molecules than it has been trained on. This problem is usually called size extrapolation in the graph neural network literature.

To measure the network ability to extrapolate, we set the number of atoms to generate to $n_{\text{max}} + k$, where n_{max} is the maximal graph size in the dataset and $k \in [5, 10, 20]$. We generate 24 batches of 256 molecules (=6144 molecules) in each setting and measure the proportion of valid and unique molecules – all these molecules are novel since they are larger than the training set.

The results are presented in Table C.3. We observe an important discrepancy between the two datasets: DiGress is very capable of extrapolation on GuacaMol, but completely fails on MOSES.

Table C.3 – Proportion of valid and unique molecules obtained when sampling larger molecules than the maximal size in the training set. Interestingly, DiGress performs very well on GuacaMol and poorly on MOSES. We hypothesize that this is due to GuacaMol being a more diverse dataset, which forces the network to learn to generate good molecules of all sizes.

	Dataset statistics			Valid and unique (%)		
	n_{min}	n_{average}	n_{max}	$n_{\text{max}} + 5$	$n_{\text{max}} + 10$	$n_{\text{max}} + 20$
MOSES	8	21.7	27	2.6	2.2	0.0
GuacaMol	2	27.8	88	87.3	85.6	80.5

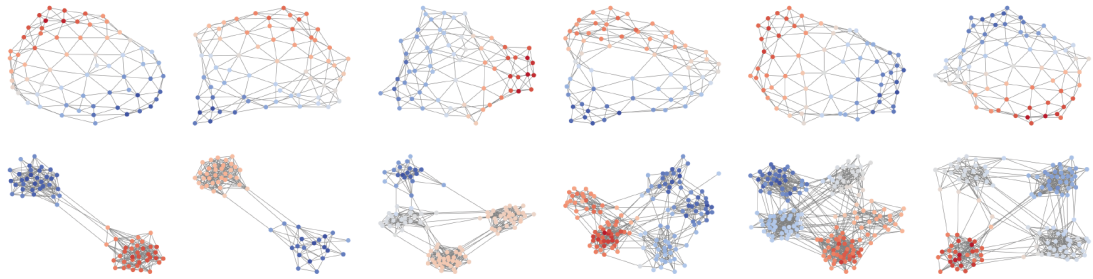


Figure C.2 – Non curated samples generated by DiGress trained on planar graphs (top) and graphs drawn from the stochastic block model (bottom).

This can be explained by the respective statistics of the datasets: MOSES features molecules that are relatively homogeneous in size. On the contrary, GuacaMol features molecules that are much larger than the dataset average. The network is therefore trained on more diverse examples, which we conjecture is why it learns some size invariance properties. The major difference in extrapolation ability that we obtain clearly highlights the value of large and diverse datasets.

We finally note that our denoising network was not designed to be size invariant, as it for example features sum aggregation functions at each layer. Specific techniques such as SizeShiftReg (Buffelli et al., 2022) could also be used to improve the size-extrapolation ability of DiGress if needed for downstream applications.

C.4 Samples from our model

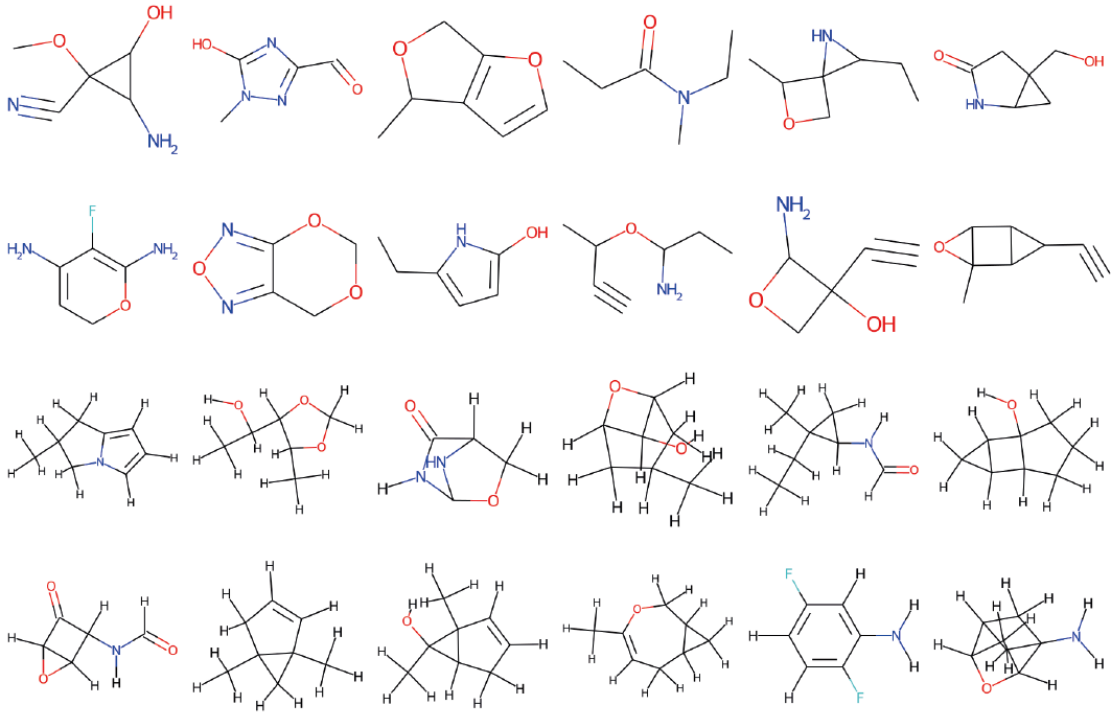


Figure C.3 – Non curated samples generated by DiGress, trained on QM9 with implicit hydrogens (top), and explicit hydrogens (bottom).

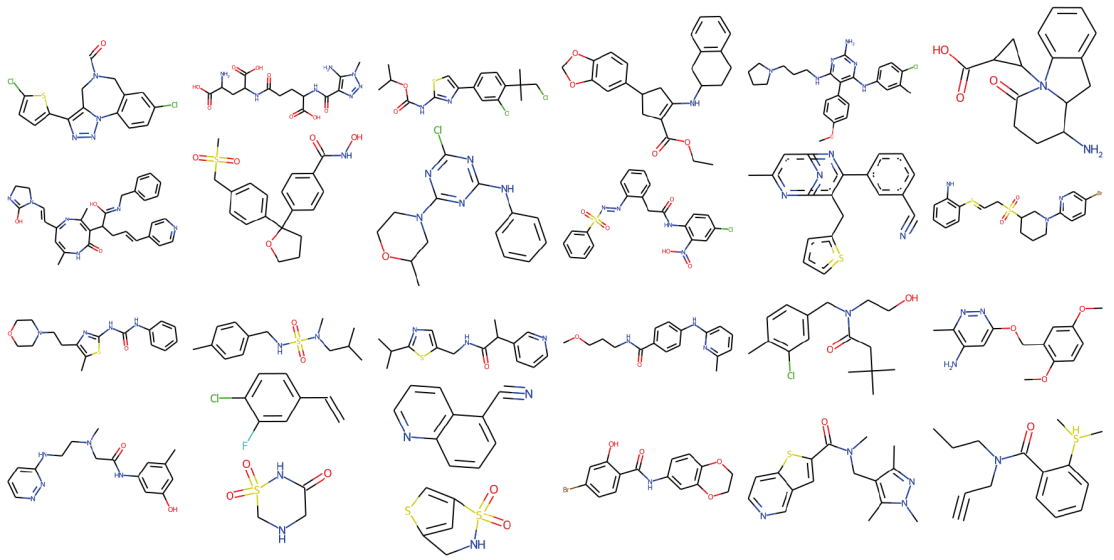


Figure C.4 – Non curated samples generated by Guacamol (top) and Moses (bottom). While there are some failure cases (disconnected molecules or invalid molecules), our model is the first non autoregressive method that scales to these datasets that are much more complex than the standard QM9.

D MiDi

D.1 Additional Results

QM9 with Implicit Hydrogens

The results are presented in Table D.1. Overall, all methods achieve good metrics on this small dataset. Part of the reason why MiDi achieves higher validity than EDM is because it can handle formal charges.

The results are presented in Table D.1 for 5 samplings of the same checkpoint. While all methods overall achieve good results on this simple dataset, we observe that the lookup table of EDM sometimes fails to predict the bond type, resulting in much more invalid molecules than our model. Interestingly, the adaptive noise schedule that allowed for important improvements on the GEOM-DRUG dataset is not effective on this simpler dataset, and the uniform schedule seems to perform better. The reasons for this phenomenon. We finally observe that the bond length predictions are overall good for all methods, but that MiDi is not as precise as EDM, which can be explained by the fact that EDM uses both learning rate decay and an exponential moving average.

D.2 Samples from our models

Table D.1 – Unconditional generation on QM9 with implicit hydrogens. On this simple dataset, all methods acheve very good results, although the lookup table of EDM sometimes fail to generate correct edge types, resulting in invalid molecules.

Metric (↑)	Validity	Uniqueness	Novelty	Connected
Data	99.5	99.9	–	100
EDM	96.8	**96.6**	45.5	**100.0**
EDM + OBabel	**100.0**	96.1	45.4	**100.0**
MiDi (uniform)	99.5±.1	95.8±.2	**49.2**±0.4	**100**±.0
MiDi (adaptive)	99.7±.0	93.9±.2	44.7±.5	**100**±.0
Metric (↓)	Valency(e-2)	Atom(e-2)	Bond(e-2)	Bond Lengths(e-2)
Data	0.6	0.1	0.1	0.3
EDM	4.3	**2.9**	0.9	**0.2**
EDM + OBabel	3.8	**2.9**	**0.3**	**0.2**
MiDi (uniform)	**2.3**±1.7	3.1±3.7	**0.4**±.1	0.8±.3
Midi (adaptive)	5.9±.2	11.7±0.2	1.1±.0	1.6±.7

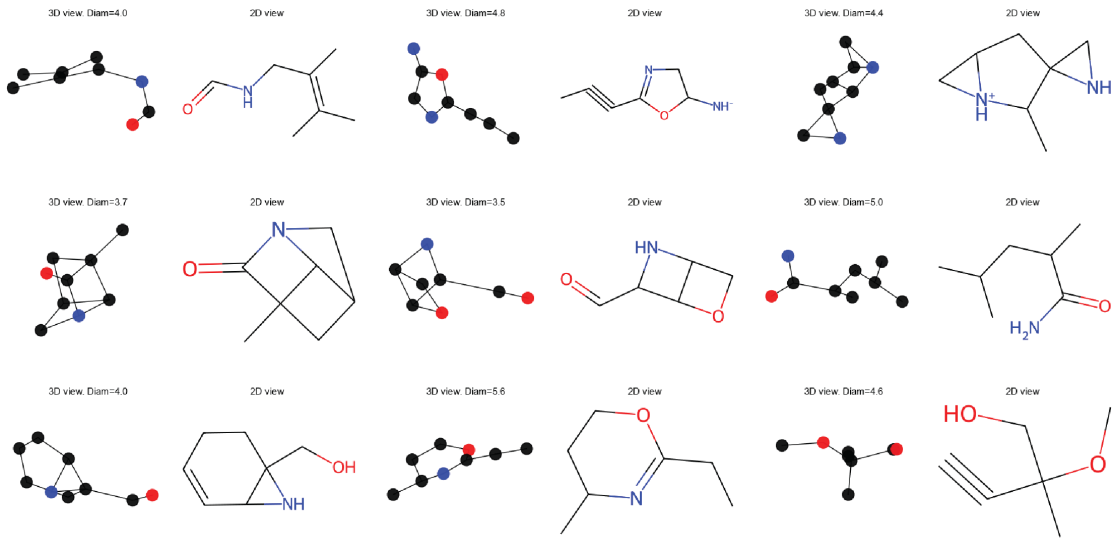


Figure D.1 – Non-curated samples on QM9 with implicit hydrogens.

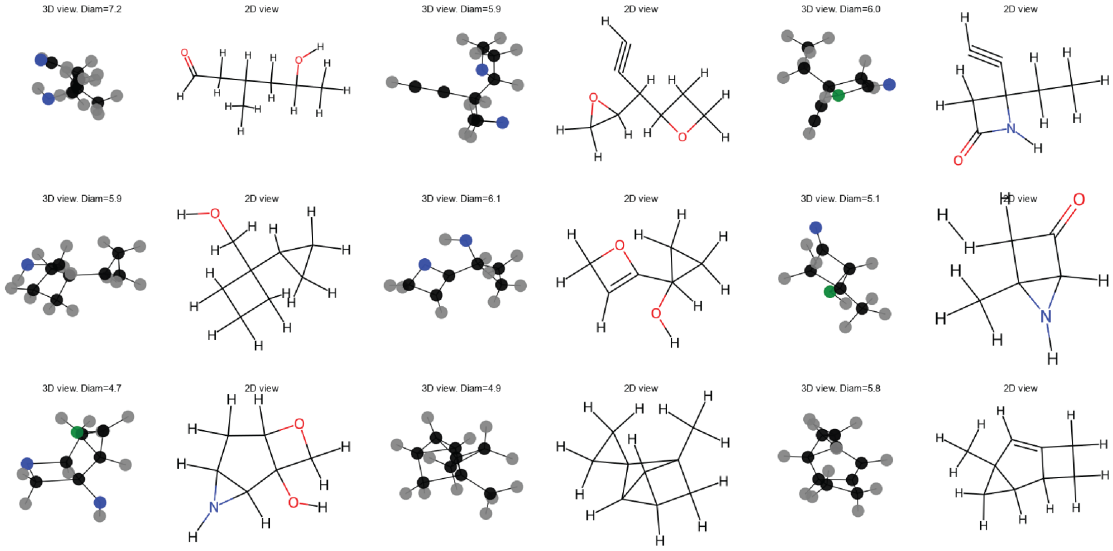


Figure D.2 – Non-curated samples on QM9 with explicit hydrogens.

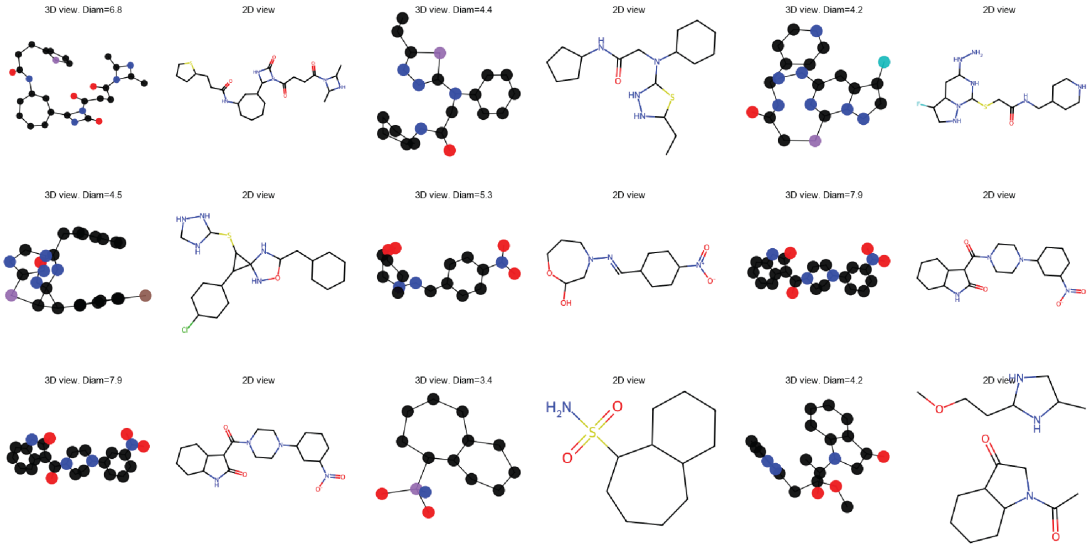


Figure D.3 – Non-curated samples on GEOM-drugs with implicit hydrogens.

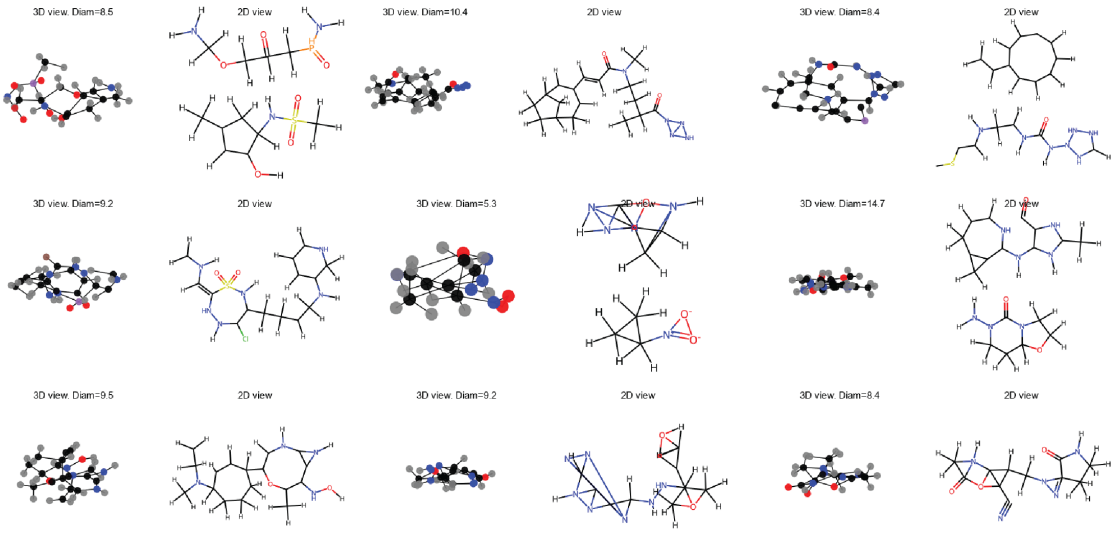


Figure D.4 – Non-curated samples on GEOM-drugs with explicit hydrogens. This dataset contains large graphs and is used to challenge the limits of our model: although our model performs better than previous methods (Table 5.7), we observe that many conformations are not realistic. For practical applications, we recommend using the model with implicit hydrogens when possible.

Bibliography

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pp. 40–49. PMLR, 2018. 6, 56

Amir Hosein Khas Ahmadi. *Memory-based graph networks*. University of Toronto (Canada), 2020. 29

Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4): 844–856, 1995. 37

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020. 27, 41

Jacob Austin, Daniel 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, 2021. 70, 75, 76

Simon Axelrod and Rafael Gomez-Bombarelli. Geom: Energy-annotated molecular conformations for property prediction and molecular generation. *arXiv preprint arXiv:2006.05531*, 2020. 97

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 94

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 21

Fan Bao, Min Zhao, Zhongkai Hao, Peiyao Li, Chongxuan Li, and Jun Zhu. Equivariant energy-guided sde for inverse molecular design. *arXiv preprint arXiv:2209.15408*, 2022. 72

Pablo Barceló, Egor V Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*, 2019. 36

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016. 17, 35

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. 38, 57

Dominique Beaini, Saro Passaro, Vincent Létourneau, Will Hamilton, Gabriele Corso, and Pietro Liò. Directional graph networks. In *International Conference on Machine Learning*, pp. 748–758. PMLR, 2021. 26, 51, 52, 83

Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M Bronstein, and Haggai Maron. Equivariant subgraph aggregation networks. *arXiv preprint arXiv:2110.02910*, 2021. 52

Marin Biloš and Stephan Günnemann. Equivariant normalizing flows for point processes and sets, 2021. URL https://openreview.net/forum?id=LIR3aVGIlln. 56

Aleksandar Bojchevski and Stephan Günnemann. Certifiable robustness to graph perturbations. *Advances in Neural Information Processing Systems*, 32, 2019. 28

Giorgos Bouritsas, Fabrizio Frasca, Stefanos P Zafeiriou, and Michael Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. 36, 51, 83

Johannes Brandstetter, Rob Hesselink, Elise van der Pol, Erik J Bekkers, and Max Welling. Geometric and physical quantities improve e (3) equivariant message passing. *arXiv preprint arXiv:2110.02905*, 2021. 13

Xavier Bresson and Thomas Laurent. A two-step graph convolutional decoder for molecule generation. *arXiv preprint arXiv:1906.03412*, 2019. 57

Marc Brockschmidt, Miltiadis Allamanis, Alexander L. Gaunt, and Oleksandr Polozov. Generative code modeling with graphs. In *International Conference on Learning Representations (ICLR)*, 2019. 2

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021. 22

Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021. 11

Jennifer AN Brophy and Christopher A Voigt. Principles of genetic circuit design. *Nature methods*, 11(5):508–520, 2014. 2

Nathan Brown, Marco Fiscato, Marwin HS Segler, and Alain C Vaucher. Guacamol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling*, 59 (3):1096–1108, 2019. 88

Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886, 2013. 11

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013. 1

Davide Buffelli, Pietro Liò, and Fabio Vandin. Sizeshiftreg: a regularization method for improving size-generalization in graph neural networks. *arXiv preprint arXiv:2207.07888*, 2022. 122

Chen Cai and Yusu Wang. A simple yet effective baseline for non-attributed graph classification. *arXiv preprint arXiv:1811.03508*, 2018. 2, 25

Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020. 27

Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identifications. *Combinatorica*, 12(4):389–410, 1992. 23, 26

Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *arXiv preprint arXiv:2102.09544*, 2021. 20, 27

Chaoqi Chen, Yushuang Wu, Qiyuan Dai, Hong-Yu Zhou, Mutian Xu, Sibei Yang, Xiaoguang Han, and Yizhou Yu. A survey on graph neural networks and graph transformers in computer vision: A task-oriented perspective. *arXiv preprint arXiv:2209.13232*, 2022. 22

Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. *Advances in neural information processing systems*, 32, 2019. 4, 5, 26, 36, 37, 43

Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? *Advances in neural information processing systems*, 33:10383–10395, 2020. 2, 23, 25, 35, 83, 84

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. 50

Fan RK Chung and Fan Chung Graham. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997. 83

Taco Cohen et al. *Equivariant convolutional networks*. PhD thesis, Taco Cohen, 2021. 11, 13

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020. xii, xv, 2, 16, 20, 46, 47, 50, 51, 112

Gabriele Corso, Hannes Stärk, Bowen Jing, Regina Barzilay, and Tommi Jaakkola. Diffdock: Diffusion steps, twists, and turns for molecular docking. *arXiv preprint arXiv:2210.01776*, 2022. 72

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26:2292–2300, 2013. 60

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. 1, 63

George Dasoulas, Ludovic Dos Santos, Kevin Scaman, and Aladin Virmaux. Coloring graph neural networks for node disambiguation. *arXiv preprint arXiv:1912.06058*, 2019. 37

Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018. 3, 6, 56, 59, 66, 71

Pim de Haan, Taco S Cohen, and Max Welling. Natural graph networks. *Advances in neural information processing systems*, 33:3636–3646, 2020. 2, 52

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016. 1, 20, 29, 31, 32

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021. 69, 85, 99

Gabriel dos Passos Gomes, Robert Pollice, and Alán Aspuru-Guzik. Navigating through the maze of homogeneous catalyst design with machine learning. *Trends in Chemistry*, 3(2):96–110, 2021. 2

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021. 80, 94

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*, 2021. 52

Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023. URL http://jmlr.org/papers/v24/22-0567.html. 21, 51

Daniel C Elton, Zois Boukouvalas, Mark D Fuge, and Peter W Chung. Deep learning for molecular design—a review of the state of the art. *Molecular Systems Design & Engineering*, 4(4):828–849, 2019. 35

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019. 18, 48

Matthias Fey, Jan-Gin Yuen, and Frank Weichert. Hierarchical inter-message passing for learning on molecular graphs. *arXiv preprint arXiv:2006.12179*, 2020. 51

Jean Feydy, Thibault Séjourné, François-Xavier Vialard, Shun-ichi Amari, Alain Trouve, and Gabriel Peyré. Interpolating between optimal transport and mmd using sinkhorn divergences. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 2681–2690, 2019. 60

Fabrizio Frasca, Beatrice Bevilacqua, Michael Bronstein, and Haggai Maron. Understanding and extending subgraph gnns by rethinking their symmetries. *Advances in Neural Information Processing Systems*, 35:31376–31390, 2022. 52

Octavian-Eugen Ganea, Lagnajit Pattanaik, Connor W Coley, Regina Barzilay, Klavs F Jensen, William H Green, and Tommi S Jaakkola. Geomol: Torsional geometric generation of molecular 3d conformer ensembles. *arXiv preprint arXiv:2106.07802*, 2021. 72

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pp. 2083–2092. PMLR, 2019. 54, 61, 102

Victor Garcia and Joan Bruna. Few-shot learning with graph neural networks. *arXiv preprint arXiv:1711.04043*, 2017. 2

Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning*, pp. 3419–3430. PMLR, 2020. 2, 36

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018. 27, 28, 32

Niklas WA Gebauer, Michael Gastegger, and Kristof T Schütt. Symmetry-adapted generation of 3d point sets for the targeted discovery of molecules. *arXiv preprint arXiv:1906.00957*, 2019. 70, 72, 95

Niklas WA Gebauer, Michael Gastegger, Stefaan SP Hessmann, Klaus-Robert Müller, and Kristof T Schütt. Inverse design of 3d molecular structures with conditional generative neural networks. *arXiv preprint arXiv:2109.04824*, 2021. 72

Floris Geerts, Filip Mazowiecki, and Guillermo A Pérez. Let's agree to degree: Comparing graph convolutional networks in the message-passing framework. *arXiv preprint arXiv:2004.02593*, 2020. 36

Mario Geiger and Tess Smidt. e3nn: Euclidean neural networks. *arXiv preprint arXiv:2207.09453*, 2022. 13

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*, pp. 1263–1272. PMLR, 2017. 1, 19, 46

Francesco Grassi, Andreas Loukas, Nathanaël Perraudin, and Benjamin Ricaud. A time-vertex signal processing framework: Scalable processing and meaningful representations for time-series on graphs. *IEEE Transactions on Signal Processing*, 66(3):817–829, 2017. 26

Nicolas Grelier, Bastien Pasdeloup, Jean-Charles Vialatte, and Vincent Gripon. Neighborhood-preserving translations on graphs. In *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 410–414. IEEE, 2016. 29

Jiaqi Guan, Wesley Wei Qian, qiang liu, Wei-Ying Ma, Jianzhu Ma, and Jian Peng. Energy-inspired molecular conformation optimization. In *International Conference on Learning Representations*, 2022. 72

Michael Guarino, Alexander Shah, and Pablo Rivas. Dipol-gan: generating molecular graphs adversarially with relational differentiable pooling. *Under review*, 2017. 56

Kilian Konstantin Haefeli, Karolis Martinkus, Nathanaël Perraudin, and Roger Wattenhofer. Diffusion models for graphs benefit from discrete state spaces. *arXiv preprint arXiv:2210.01549*, 2022. 71

Philip J Hajduk and Jonathan Greer. A decade of fragment-based drug design: strategic advances and lessons learned. *Nature reviews Drug discovery*, 6(3):211–219, 2007. 71

Matthias Hein, Jean-Yves Audibert, and Ulrike Von Luxburg. From graphs to manifolds-weak and strong pointwise consistency of graph laplacians. In *COLT*, volume 3559, pp. 470–485. Springer, 2005. 30

Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015. 31

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf. 69, 73, 74, 75, 118

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *arXiv preprint arXiv:2204.03458*, 2022. 69

Moritz Hoffmann and Frank Noé. Generating valid euclidean distance matrices. *arXiv preprint arXiv:1910.03131*, 2019. 72

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, 2021. 75, 76

Emiel Hoogeboom, Vicctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *International Conference on Machine Learning*, pp. 8867–8887. PMLR, 2022. 70, 72, 85, 94, 95, 97, 120

Han Huang, Leilei Sun, Bowen Du, Yanjie Fu, and Weifeng Lv. Graphgdp: Generative diffusion processes for permutation invariant graph generation. *arXiv preprint arXiv:2212.01842*, 2022a. 71

Lei Huang, Hengtong Zhang, Tingyang Xu, and Ka-Chun Wong. Mdm: Molecular diffusion model for 3d molecule generation. *arXiv preprint arXiv:2209.05710*, 2022b. 72

Ilia Igashov, Hannes Stärk, Clément Vignac, Victor Garcia Satorras, Pascal Frossard, Max Welling, Michael Bronstein, and Bruno Correia. Equivariant 3d-conditional diffusion models for molecular linker design. *arXiv preprint arXiv:2210.05274*, 2022. 72, 95

John Ingraham, Max Baranov, Zak Costello, Vincent Frappier, Ahmed Ismail, Shan Tie, Wujie Wang, Vincent Xue, Fritz Obermeyer, Andrew Beam, et al. Illuminating protein space with a programmable generative model. *bioRxiv*, pp. 2022–12, 2022. 15, 72

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015. 94

Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems*, 28, 2015. 92

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pp. 2323–2332. PMLR, 2018. 86

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical generation of molecular graphs using structural motifs. In *International Conference on Machine Learning*, pp. 4839–4848. PMLR, 2020. 71, 103

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. 71, 86, 118, 120, 121

Daniel D Johnson, Jacob Austin, Rianne van den Berg, and Daniel Tarlow. Beyond in-place corruption: Insertion and deletion in denoising probabilistic models. *arXiv preprint arXiv:2107.07675*, 2021. 75

Chaitanya Joshi. Transformers are graph neural networks. *The Gradient*, 12, 2020. 17

Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019. 35

Chaitanya K Joshi, Cristian Bodnar, Simon V Mathis, Taco Cohen, and Pietro Liò. On the expressive power of geometric graph neural networks. *arXiv preprint arXiv:2301.09308*, 2023. 52

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021. 15

Sékou-Oumar Kaba, Arnab Kumar Mondal, Yan Zhang, Yoshua Bengio, and Siamak Ravanbakhsh. Equivariance with learned canonicalization functions. In *NeurIPS 2022 Workshop on Symmetry and Geometry in Neural Representations*, 2022. URL https://openreview.net/forum?id=pVD1k8ge25a. 92

Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *Advances in Neural Information Processing Systems*, 2020. 35

Henry Kenlay, Dorina Thanou, and Xiaowen Dong. Interpretable stability bounds for spectral graph filters. In *International conference on machine learning*, pp. 5388–5397. PMLR, 2021. 31

Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. *Advances in Neural Information Processing Systems*, 32, 2019. 25, 42, 109

Nicolas Keriven, Alberto Bietti, and Samuel Vaiter. Convergence and stability of graph convolutional networks on large random graphs. *Advances in Neural Information Processing Systems*, 33:21512–21523, 2020. 52

Nicolas Keriven, Alberto Bietti, and Samuel Vaiter. On the universality of graph neural networks on large random graphs. *Advances in Neural Information Processing Systems*, 34:6960–6971, 2021. 52

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pp. 6348–6358, 2017. 35

Renata Khasanova and Pascal Frossard. Graph-based isometry invariant representation learning. In *International conference on machine learning*, pp. 1847–1856. PMLR, 2017. 1, 29

Jinwoo Kim, Jaehoon Yoo, Juho Lee, and Seunghoon Hong. Setvae: Learning hierarchical composition for generative modeling of set-structured data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15059–15068, June 2021. 56

Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707, 2021. 74, 75, 79, 117

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 2, 20, 27

Jonas Köhler, Leon Klein, and Frank Noé. Equivariant flows: Exact likelihood generative learning for symmetric densities. In *Proceedings of the 37th International Conference on Machine Learning, ICML*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5361–5370. PMLR, 2020. 56, 59, 72, 77

Imre Risi Kondor. *Group theoretical methods in machine learning*. Columbia University, 2008. 11, 12, 33, 58

Risi Kondor and Shubhendu Trivedi. On the generalization of equivariance and convolution in neural networks to the action of compact groups. In *International Conference on Machine Learning*, pp. 2747–2755. PMLR, 2018. 16

Risi Kondor, Hy Truong Son, Horace Pan, Brandon Anderson, and Shubhendu Trivedi. Covariant compositional networks for learning graphs. *arXiv preprint arXiv:1801.02144*, 2018. 26, 37

Adam R Kosiorek and Danilo J Kim, Hyunjik and@articlexu2018powerful, title=How powerful are graph neural networks?, author=Xu, Keyulu and Hu, Weihua and Leskovec, Jure and Jegelka, Stefanie, journal=arXiv preprint arXiv:1810.00826, year=2018 Rezende. Conditional set generation with transformers. *Workshop on Object-Oriented Learning at ICML 2020*, 2020. xv, 6, 55, 56, 57, 64, 71, 111

Igor Krawczuk, Pedro Abranches, Andreas Loukas, and Volkan Cevher. Gg-gan: A geometric graph generative adversarial network, 2021. URL https://openreview.net/forum?id=qiAxL3Xqx1o. 54, 56, 57, 71

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. 26, 29

Youngchun Kwon, Jiho Yoo, Youn-Suk Choi, Won-Joon Son, Dongseon Lee, and Seokho Kang. Efficient learning of non-autoregressive graph variational autoencoders for molecular graph generation. *Journal of Cheminformatics*, 11(1):1–10, 2019. 66

Youngchun Kwon, Dongseon Lee, Youn-Suk Choi, Kyoham Shin, and Seokho Kang. Compressed graph representation for scalable molecular graph generation. *Journal of Cheminformatics*, 12 (1):1–8, 2020. 86

Greg Landrum et al. Rdkit: Open-source cheminformatics. *Unknown journal*, 2006. 88

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pp. 3744–3753. PMLR, 2019. 17, 57

Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2018. xi, 26, 27

Jintang Li, Jiaying Peng, Liang Chen, Zibin Zheng, Tingting Liang, and Qing Ling. Spectral adversarial training for robust graph neural network. *IEEE Transactions on Knowledge and Data Engineering*, 2022. 28

Yang Li, Haidong Yi, Christopher Bender, Siyuan Shan, and Junier B Oliva. Exchangeable neural ode for set modeling. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 6936–6946. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/4db73860ecb5533b5a6c710341d5bbec-Paper.pdf. 56

Yibo Li, Liangren Zhang, and Zhenming Liu. Multi-objective de novo drug design with conditional graph generative model. *Journal of cheminformatics*, 10(1):1–24, 2018a. 66

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pp. 539–548, 2018b. 35

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Charlie Nash, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In *NeurIPS*, 2019. 13, 54, 71, 86

Yi-Lun Liao and Tess Smidt. Equiformer: Equivariant graph attention transformer for 3d atomistic graphs. *arXiv preprint arXiv:2206.11990*, 2022. 13, 94

Derek Lim, Joshua Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and basis invariant networks for spectral graph representation learning. *arXiv preprint arXiv:2202.13013*, 2022. 15

Lu Lin, Ethan Blaser, and Hongning Wang. Graph structural attack by perturbing spectral distance. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 989–998, 2022. 28

Phillip Lippe and Efstratios Gavves. Categorical normalizing flows via continuous transformations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=-GLNZeVDuik. 71

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 338–348, 2020. 27

Meng Liu, Keqiang Yan, Bora Oztekin, and Shuiwang Ji. GraphEBM: Molecular graph generation with energy-based models. In *Energy Based Models Workshop - ICLR 2021*, 2021. URL https://openreview.net/forum?id=Gc51PtL_zYw. 56

Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. Constrained graph variational autoencoders for molecule design. *Advances in neural information processing systems*, 31, 2018. 54, 71

Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, 2020a. URL https://openreview.net/forum?id=B1l2bp4YwS. 4, 25, 35, 37, 46, 50

Andreas Loukas. How hard is to distinguish graphs with graph neural networks? In *Advances in Neural Information Processing Systems*, 2020b. 23, 37, 46

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11461–11471, 2022. xiv, 119

Shitong Luo, Jiaqi Guan, Jianzhu Ma, and Jian Peng. A 3d generative model for structure-based drug design. *Advances in Neural Information Processing Systems*, 34, 2021a. 72

Shitong Luo, Chence Shi, Minkai Xu, and Jian Tang. Predicting molecular conformation via dynamic graph score matching. *Advances in Neural Information Processing Systems*, 34, 2021b. 72

Shitong Luo, Yufeng Su, Xingang Peng, Sheng Wang, Jian Peng, and Jianzhu Ma. Antigen-specific antibody design and optimization with diffusion-based generative models. *bioRxiv*, 2022a. 70

Tianze Luo, Zhanfeng Mo, and Sinno Jialin Pan. Fast graph generative model via spectral diffusion. *arXiv preprint arXiv:2211.08892*, 2022b. 71

Youzhi Luo and Shuiwang Ji. An autoregressive flow model for 3d molecular geometry generation from scratch. In *International Conference on Learning Representations*, 2021. 72

Youzhi Luo, Keqiang Yan, and Shuiwang Ji. Graphdf: A discrete flow model for molecular graph generation. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pp. 7192–7203, 2021c. 71

Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. Is homophily a necessity for graph neural networks? *arXiv preprint arXiv:2106.06134*, 2021. 27

Kaushalya Madhawa, Katushiko Ishiguro, Kosuke Nakago, and Motoki Abe. Graphnvp: An invertible flow model for generating molecular graphs. *arXiv preprint arXiv:1905.11600*, 2019. 56, 71, 86, 91

Hiroshi Maehara and Vojtech Rödl. On the dimension to represent a graph by a unit distance graph. *Graphs and Combinatorics*, 6(4):365–367, 1990. 41, 106

Abram Magner, Mayank Baranwal, and Alfred O Hero III. The power of graph convolutional networks to distinguish random graph models: Short version. *arXiv preprint arXiv:2002.05678*, 2020. 36

Omar Mahmood, Elman Mansimov, Richard Bonneau, and Kyunghyun Cho. Masked graph modeling for molecule generation. *Nature communications*, 12(1):3156, 2021. 102

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018. 16, 25, 36, 37, 44

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019a. 4, 17, 26, 37, 43, 46

Haggai Maron, Ethan Fetaya, Nimrod Segol, and Yaron Lipman. On the universality of invariant networks. In *International conference on machine learning*, pp. 4363–4371. PMLR, 2019b. 26

Karolis Martinkus, Andreas Loukas, Nathanaël Perraudin, and Roger Wattenhofer. Spectre: Spectral conditioning helps to overcome the expressivity limits of one-shot graph generators. *arXiv preprint arXiv:2204.01613*, 2022. 86, 87, 119

Krzysztof Maziarz, Henry Richard Jackson-Flux, Pashmina Cameron, Finton Sirockin, Nadine Schneider, Nikolaus Stiefl, Marwin Segler, and Marc Brockschmidt. Learning to extend molecular scaffolds with structural motifs. In *International Conference on Learning Representations (ICLR)*, 2022. 71, 86, 103, 119

Søren Ager Meldgaard, Jonas Köhler, Henrik Lund Mortensen, Mads-Peter Verner Christiansen, Frank Noé, and Bjork Hammer. Generating stable molecules using imitation and reinforcement learning. *Machine Learning: Science and Technology*, 2021. 67

Facundo Mémoli. On the use of gromov-hausdorff distances for shape comparison. *Unknown*, 2007. 60

Rocío Mercado, Tobias Rastemo, Edvard Lindelöf, Günter Klambauer, Ola Engkvist, Hongming Chen, and Esben Jannik Bjerrum. Graph networks for molecular design. *Machine Learning: Science and Technology*, 2(2):025023, 2021. 71, 86

Diego Mesquita, Amauri Souza, and Samuel Kaski. Rethinking pooling in graph neural networks. *Advances in Neural Information Processing Systems*, 33:2220–2231, 2020. 2

Erxue Min, Runfa Chen, Yatao Bian, Tingyang Xu, Kangfei Zhao, Wenbing Huang, Peilin Zhao, Junzhou Huang, Sophia Ananiadou, and Yu Rong. Transformer for graphs: An overview from architecture perspective. *arXiv preprint arXiv:2202.08455*, 2022. 22

Joshua Mitton, Hans M Senn, Klaas Wynne, and Roderick Murray-Smith. A graph vae and graph transformer approach to generating molecular graphs. *arXiv preprint arXiv:2104.04345*, 2021. 56, 66, 67

Alex Morehead and Jianlin Cheng. Geometry-complete diffusion for 3d molecule generation. *arXiv preprint arXiv:2302.04313*, 2023. 72

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 4602–4609, 2019. 2, 4, 24, 25, 26, 35, 36, 37, 46, 57, 83

Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. *Advances in Neural Information Processing Systems*, 33:21824–21840, 2020. 25, 36

Christopher Morris, Yaron Lipman, Haggai Maron, Bastian Rieck, Nils M Kriege, Martin Grohe, Matthias Fey, and Karsten Borgwardt. Weisfeiler and leman go machine learning: The story so far. *arXiv preprint arXiv:2112.09992*, 2021. 2

Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampášek. Attending to graph transformers. *arXiv preprint arXiv:2302.04181*, 2023. 22

Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 4663–4673, 2019. URL http://proceedings.mlr.press/v97/murphy19a.html. 4, 14, 25, 35, 37, 48

Charlie Nash, Yaroslav Ganin, SM Ali Eslami, and Peter Battaglia. Polygen: An autoregressive generative model of 3d meshes. In *International Conference on Machine Learning*, pp. 7220–7229. PMLR, 2020. 54

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*, pp. 4474–4484. PMLR, 2020. 71

Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457), 2019. 72

Hoang Nt and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019. 27, 32

Noel M O'Boyle, Michael Banck, Craig A James, Chris Morley, Tim Vandermeersch, and Geoffrey R Hutchison. Open babel: An open chemical toolbox. *Journal of cheminformatics*, 3 (1):1–14, 2011. 72, 95

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. xiii, 20, 61, 62, 80

Daniil Polykovskiy, Alexander Zhebrak, Benjamin Sanchez-Lengeling, Sergey Golovanov, Oktai Tatanov, Stanislav Belyaev, Rauf Kurbanov, Aleksey Artamonov, Vladimir Aladinskiy, Mark Veselov, et al. Molecular sets (moses): a benchmarking platform for molecular generation models. *Frontiers in pharmacology*, 11:565644, 2020. 88

Omri Puny, Matan Atzmon, Heli Ben-Hamu, Ishan Misra, Aditya Grover, Edward J Smith, and Yaron Lipman. Frame averaging for invariant and equivariant network design. *arXiv preprint arXiv:2110.03336*, 2021. 15

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017. 22, 39

Matthew Ragoza, Tomohide Masuda, and David Ryan Koes. Learning a continuous representation of 3d molecular structures with deep generative models. *arXiv preprint arXiv:2010.08687*, 2020. 72

Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014. 66

Lars Ruddigkeit, Ruud Van Deursen, Lorenz C Blum, and Jean-Louis Reymond. Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17. *Journal of chemical information and modeling*, 52(11):2864–2875, 2012. 66

T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023. 27

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 4470–4479, 2018. URL http://proceedings.mlr.press/v80/sanchez-gonzalez18a.html. 35

Aliaksei Sandryhaila and Jose MF Moura. Big data analysis with signal processing on graphs: Representation and processing of massive data sets with irregular structure. *IEEE signal processing magazine*, 31(5):80–90, 2014. 32, 39

Ryoma Sato. A survey on the expressive power of graph neural networks. *ArXiv*, abs/2003.04078, 2020. 36

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Approximation ratios of graph neural networks for combinatorial problems. In *Advances in Neural Information Processing Systems*, pp. 4083–4092, 2019. 36

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. *arXiv preprint arXiv:2002.03155*, 2020. 4, 25, 37

Victor Garcia Satorras, Zeynep Akata, and Max Welling. Combining generative and discriminative models for hybrid inference. In *Advances in Neural Information Processing Systems*, pp. 13802–13812, 2019. 35

Victor Garcia Satorras, Emiel Hoogeboom, Fabian Fuchs, Ingmar Posner, and Max Welling. E(n) equivariant normalizing flows. *Advances in Neural Information Processing Systems*, 34, 2021a. 70, 72, 95, 97, 98

Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. *arXiv preprint arXiv:2102.09844*, 2021b. 56, 67, 71, 92

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008. 1, 19, 30

Franco Scarselli, Ah Chung Tsoi, and Markus Hagenbuchner. The vapnik–chervonenkis dimension of graph and recursive neural networks. *Neural Networks*, 108:248–259, 2018. 2

Arne Schneuing, Yuanqi Du, Charles Harris, Arian Jamasb, Ilia Igashov, Weitao Du, Tom Blundell, Pietro Lió, Carla Gomes, Max Welling, et al. Structure-based drug design with equivariant diffusion models. *arXiv preprint arXiv:2210.13695*, 2022. 72, 95

Nimrod Segol and Yaron Lipman. On universal equivariant set networks. *arXiv preprint arXiv:1910.02421*, 2019. 22

Jean-Pierre Serre. *Linear representations of finite groups*, volume 42. Springer, 1977. 12

Hadar Serviansky, Nimrod Segol, Jonathan Shlomi, Kyle Cranmer, Eilam Gross, Haggai Maron, and Yaron Lipman. Set2graph: Learning graphs from sets. *Advances in Neural Information Processing Systems*, 33, 2020. 6, 53, 57, 113

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018. 22, 27

Chence Shi, Shitong Luo, Minkai Xu, and Jian Tang. Learning gradient fields for molecular conformation generation. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML*, 2021. 72

Chence Shi, Chuanrui Wang, Jiarui Lu, Bozitao Zhong, and Jian Tang. Protein sequence and structure co-design with equivariant translation. *arXiv preprint arXiv:2210.08761*, 2022. 72

David I Shuman, Pierre Vandergheynst, and Pascal Frossard. Chebyshev polynomial approximation for distributed signal processing. In *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, pp. 1–8. IEEE, 2011. 32

Gregor N. C. Simm, Robert Pinsler, Gábor Csányi, and José Miguel Hernández-Lobato. Symmetry-aware actor-critic for 3d molecular design. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=jEYKjPE1xYN. 72

Gregor NC Simm and José Miguel Hernández-Lobato. A generative model for molecular distance geometry. *arXiv preprint arXiv:1909.11459*, 2019. 72

Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International conference on artificial neural networks*, pp. 412–422. Springer, 2018. 3, 6, 56, 59, 66, 71

Daniel GA Smith, Lori A Burns, Andrew C Simmonett, Robert M Parrish, Matthew C Schieber, Raimondas Galvelis, Peter Kraus, Holger Kruse, Roberto Di Remigio, Asem Alenaizan, et al. Psi4 1.4: Open-source software for high-throughput quantum chemistry. *The Journal of chemical physics*, 152(18):184108, 2020. 88

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis R. Bach and David M. Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML*, 2015. 69, 74, 79, 85

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019. 73

Karl Stelzner, Kristian Kersting, and Adam R Kosiorek. Generative adversarial set transformers. In *Workshop on Object-Oriented Learning at ICML*, volume 2020, 2020. 56, 57

Mengying Sun, Sendong Zhao, Coryandar Gilvary, Olivier Elemento, Jiayu Zhou, and Fei Wang. Graph convolutional networks for computational drug development and discovery. *Briefings in bioinformatics*, 2019. 35

Yiwei Sun, Suhang Wang, Xianfeng Tang, Tsung-Yu Hsieh, and Vasant Honavar. Adversarial attacks on graph neural networks via node injections: A hierarchical reinforcement learning approach. In *Proceedings of the Web Conference 2020*, pp. 673–683, 2020a. 28

Yongbin Sun, Yue Wang, Ziwei Liu, Joshua Siegel, and Sanjay Sarma. Pointgrow: Autoregressively learned point cloud generation with self-attention. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 61–70, 2020b. 54

Jukka Suomela. Survey of local algorithms. *ACM Computing Surveys (CSUR)*, 45(2):1–40, 2013. 4, 37

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013. 28

Behrooz Tahmasebi, Derek Lim, and Stefanie Jegelka. Counting substructures with higher-order graph neural networks: Possibility and impossibility results. *arXiv preprint arXiv:2012.03174*, 2020. 23, 25

Dorina Thanou, David I Shuman, and Pascal Frossard. Learning parametric dictionaries for signals on graphs. *IEEE Transactions on Signal Processing*, 62(15):3849–3862, 2014. 32

Nathaniel Thomas, Tess Smidt, Steven M. Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. *CoRR*, abs/1802.08219, 2018. 13

Matteo Togninalli, Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten Borgwardt. Wasserstein weisfeiler-lehman graph kernels. *Advances in neural information processing systems*, 32, 2019. 102

Brian L Trippe, Jason Yim, Doug Tischer, Tamara Broderick, David Baker, Regina Barzilay, and Tommi Jaakkola. Diffusion probabilistic modeling of protein backbones in 3d for the motif-scaffolding problem. *arXiv preprint arXiv:2206.04119*, 2022. 72

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 11, 17, 57, 91

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 21

Petar Velickovic, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. In *International Conference on Learning Representations*, 2020. 44

Clément Vignac and Pascal Frossard. Learning anisotropic filters on product graphs. In *Proceedings of the ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 26

Clement Vignac, Andreas Loukas, and Pascal Frossard. Building powerful and equivariant graph neural networks with structural message-passing. *Advances in Neural Information Processing Systems*, 33:14143–14155, 2020. 2, 17

Ulrike von Luxburg and Olivier Bousquet. Distance-based classification with lipschitz functions. *J. Mach. Learn. Res.*, 5(Jun):669–695, 2004. 1

Edward Wagstaff, Fabian Fuchs, Martin Engelcke, Ingmar Posner, and Michael A Osborne. On the limitations of representing functions on sets. In *International Conference on Machine Learning*, pp. 6487–6494. PMLR, 2019. 22

Minjie Yu Wang. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR workshop on representation learning on graphs and manifolds*, 2019. 18

Siyue Wang, Xiao Wang, Pu Zhao, Wujie Wen, David Kaeli, Peter Chin, and Xue Lin. Defensive dropout for hardening deep neural networks under adversarial attacks. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8. IEEE, 2018. 28

Zichao Wang, Weili Nie, Zhuoran Qiao, Chaowei Xiao, Richard Baraniuk, and Anima Anandkumar. Retrieval-based controllable molecule generation. *arXiv preprint arXiv:2208.11126*, 2022. 71

Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borst, Robert J Ragotte, Lukas F Milles, et al. Broadly applicable and accurate protein design by integrating structure prediction networks and diffusion generative models. *bioRxiv*, 2022. 72

Boris Weisfeiler. *On construction and identification of graphs*, volume 558. Springer, 2006. 23

Boris Weisfeiler and Andrei A Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968. 2

Daniel Worrall and Max Welling. Deep scale-spaces: Equivariance over scale. *Advances in Neural Information Processing Systems*, 32, 2019. 11

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019. 2, 27, 32

Kevin E Wu, Kevin K Yang, Rianne van den Berg, James Y Zou, Alex X Lu, and Ava P Amini. Protein structure generation via folding diffusion. *arXiv preprint arXiv:2209.15611*, 2022. 72

Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018. 87, 96

Xiaolin Xia, Jianxing Hu, Yanxing Wang, Liangren Zhang, and Zhenming Liu. Graph-based generative models for de novo drug design. *Drug Discovery Today: Technologies*, 32:45–53, 2019. 2

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km. 2, 4, 20, 21, 24, 35, 44, 46, 70, 83

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *International Conference on Learning Representations*, 2020a. URL https://openreview.net/forum?id=rJxbJeHFPS. 23, 35

Keyulu Xu, Mozhi Zhang, Jingling Li, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. *arXiv preprint arXiv:2009.11848*, 2020b. 1

Minkai Xu, Wujie Wang, Shitong Luo, Chence Shi, Yoshua Bengio, Rafael Gomez-Bombarelli, and Jian Tang. An end-to-end framework for molecular conformation generation via bilevel programming. *arXiv preprint arXiv:2105.07246*, 2021. 72

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*, 2022. URL https://openreview.net/forum?id=PzcvxEMzvQC. 72, 77, 90

Carl Yang, Peiye Zhuang, Wenhan Shi, Alan Luu, and Pan Li. Conditional structure generation through graph variational generative adversarial nets. In *NeurIPS*, pp. 1338–1349, 2019a. 56

Che Yang, Fabian Sesterhenn, Jaume Bonet, Eva A van Aalen, Leo Scheller, Luciano A Abriata, Johannes T Cramer, Xiaolin Wen, Stéphane Rosset, Sandrine Georgeon, et al. Bottom-up de novo design of functional proteins with complex structural features. *Nature Chemical Biology*, 17(4):492–500, 2021a. 2

Dongchao Yang, Jianwei Yu, Helin Wang, Wen Wang, Chao Weng, Yuexian Zou, and Dong Yu. Diffsound: Discrete diffusion model for text-to-sound generation. *arXiv e-prints*, pp. arXiv–2207, 2022. 70, 75, 76

Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4541–4550, 2019b. 56

Liang Yang, Mengzhe Li, Liyang Liu, Chuan Wang, Xiaochun Cao, Yuanfang Guo, et al. Diverse message passing for attribute with heterophily. *Advances in Neural Information Processing Systems*, 34:4751–4763, 2021b. 27

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018. 29, 39, 102

KiJung Yoon, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard Zemel, and Xaq Pitkow. Inference in probabilistic graphical models by graph neural networks. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pp. 868–875. IEEE, 2019. 35

Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *arXiv preprint arXiv:1806.02473*, 2018a. 71

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pp. 5708–5717. PMLR, 2018b. 86

Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. *arXiv preprint arXiv:1906.04817*, 2019. 41

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017. 1, 17, 22, 39, 41, 57

David W Zhang, Gertjan J. Burghouts, and Cees G. M. Snoek. Set prediction without imposing structure as conditional density estimation. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=04ArenGOz3. 56

Kaiyi Zhang, Ximing Yang, Yuan Wu, and Cheng Jin. Attention-based transformation from latent features to point clouds. *arXiv preprint arXiv:2112.05324*, 2021b. 56

Lily H Zhang, Veronica Tozzo, John M. Higgins, and Rajesh Ranganath. Set norm and equivariant skip connections: Putting the deep in deep sets, 2022. URL https://openreview.net/forum?id=MDT30TEtaVY. 94

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 5165–5175, 2018. 41

Yan Zhang, Jonathon Hare, and Adam Prugel-Bennett. Deep set prediction networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alche-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/6e79ed05baec2754e25b4eac73a332d2-Paper.pdf. xv, 57, 64, 111

Yan Zhang, Jonathon Hare, and Adam Prügel-Bennett. Fspool: Learning set representations with featurewise sort pooling. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJgBA2VYwH. 56

Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020. 27

Jiong Zhu, Ryan A Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K Ahmed, and Danai Koutra. Graph neural networks with heterophily. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 11168–11176, 2021. 27

Yanqiao Zhu, Yuanqi Du, Yinkai Wang, Yichen Xu, Jieyu Zhang, Qiang Liu, and Shu Wu. A survey on deep graph generation: Methods and applications. *arXiv preprint arXiv:2203.06714*, 2022. 71

Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2847–2856, 2018. 28

Clément Vignac

clement.vignac@epfl.ch
+ 33 6 13 17 04 10
https://cvignac.github.io/
Github: cvignac
Google Scholar

About me

I am a researcher in generative AI for point cloud and graph data. I have just completed my PhD at EPFL under the guidance of Pascal Frossard and co-supervision of Max Welling within the ELLIS program. During my PhD, I developed several denoising diffusion models for molecules, namely EDM, DiGress and MiDi.

Education

2018 - (Sept 2023) **PhD in Machine learning**
EPFL , Lausanne, Switzerland

2017 - 2018 **MSc. "Mathematics, Vision, Learning"**
MVA, ENS Paris-Saclay, France

2014 - 2017 **Engineering Degree (equivalent to MSc.) in Applied Mathematics**
École Polytechnique, Paris, France

2012 - 2014 **Preparatory Classes** (intensive program to enter top universities)
Lycée Henri IV and Lycée Louis Le Grand, Paris, France

Research experience

2018 - (Sept 2023) **Doctoral Assistant** at EPFL, Switzerland
My research advances methodology on the generation of unordered data (sets and graphs), with a particular focus on molecule generation.

2021 (Sept - Dec) **Visiting researcher** at the University of Amsterdam, Netherlands
Supervision by Max Welling as part of the ELLIS PhD program.
We developed the first denoising diffusion model for molecule generation in 3D.

2018 (Apr - Sept) **Master thesis** at EPFL, Switzerland
Studied anisotropic graph neural networks with Pascal Frossard

2017 (Apr - Aug) **Master thesis** at University of Genoa, Italy
Developed an algorithm for learning-to-rank with A. Rudi and L. Rosasco

Software skills

Deep learning: PyTorch, Tensorflow
Languages: Python, Javascript, Java
Misc: Unix, Git, Docker, Kubernetes, Azure

Languages

French: ●●●●● German: ●●●○○

English: ●●●●◐ Italian: ●●○○○

Scholarships

ELLIS Society PhD program

Swiss Data Science Center fellowship

Other work experience

June - Aug 2016 **Software developer intern** at Pzartech, Tel Aviv, Israel
- Developed a Javascript platform and a API to 3D print spare parts.
- Handled the cloud infrastructure (Azure) and copyright issues in collaboration with layers.

Oct 14 - Apr 15 **Education teacher** at Apprentis d'Auteuil, Toulouse, France
Worked in a school for teenagers with behavioural disorders. I developed projects such an album recording and helped them in their everyday life.

Teaching

Teaching assistant for "Network Machine learning" (2022), "Digital Signal Processing" (2020, 2022) and "A network tour of Data science" (2019).
During my PhD, I supervised 13 students, including 2 junior PhD students and 4 master theses.

First-author publications

Point cloud and graph generation

- MiDi: Mixed Graph and 3D Denoising Diffusion for Molecule Generation
  CV*, N. Osman, L. Toni, P. Frossard, *ECML KDD 2023*

- Digress: Discrete denoising diffusion for graph generation
  CV*, I. Krawczuk*, A. Siraudin, B. Wang, V. Cevher, P. Frossard, *ICLR 2023*

- *Equivariant Diffusion for Molecule Generation in 3d*
  E. Hoogeboom, V. Garcia Sattoras*,* CV*, *M. Welling*, *Spotlight at ICML 2022*

- *Top-N: Equivariant set and graph generation without exchangeability*
  CV, P. Frossard, *ICLR 2022*

Graph neural networks

- *Building powerful and equivariant graph neural networks with structural message-passing*
  CV, A. Loukas, P. Frossard, *NeurIPS 2020*

- On the choice of Graph neural network architectures
  C.V., G. Ortiz-Jiménez, P. Frossard, *ICASSP 2020*

Community service

Reviewer for NeurIPS, ICML and AISTATS

Personal interests

Judo, climbing, hiking, agronomy, chess