GraphSPNs: Sum-Product Networks Benefit From Canonical Orderings

Milan Papež[1] **Martin Rektoris**[1] **Václav Šmídl**[1] **Tomáš Pevný**[1]

[1]Artificial Intelligence Center, Czech Technical University, Prague, Czech Republic

Abstract

Deep generative models have recently made remarkable progress in capturing complex probability distributions over graphs. However, they are intractable and thus unable to answer even the most basic probabilistic inference queries without resorting to approximations. Therefore, we propose graph sum-product networks (GraphSPNs), a tractable deep generative model that provides exact and efficient inference over (arbitrary parts of) graphs. We investigate different principles to make SPNs permutation invariant. We demonstrate that GraphSPNs can (conditionally) generate novel and chemically valid molecular graphs, being competitive to, and sometimes even better than, existing intractable models. We find out that (Graph)SPNs benefit from ensuring the permutation invariance via canonical ordering.

1 INTRODUCTION

Graphs are a fundamental framework for representing real or abstract objects and their hierarchical interactions in a diverse range of scientific and engineering applications, such as discovering new materials [Choudhary et al., 2022], developing personalized diagnostic strategies [Chandak et al., 2023], and estimating time of arrival [Derrow-Pinion et al., 2021]. Nonetheless, designing probabilistic models for graphs is challenging. Graphs can exhibit highly complex and combinatorial structures, making it difficult to capture their probabilistic behavior effectively. While traditional approaches struggle to handle this problem, deep generative models—which rely on expressive graph neural networks [Wu et al., 2020, Zhang et al., 2020]—have recently made significant progress in this direction [You et al., 2018, Simonovsky and Komodakis, 2018, De Cao and Kipf, 2018, Shi et al., 2020, Jo et al., 2022].

The complication with these models is their limited ability to carry out probabilistic inference tasks beyond mere sampling and, typically, exact likelihood evaluation. For example, arbitrary conditioning with an already trained model allows us to sample conditionally on an existing (part of a) graph. The conditioning significantly reduces the space of possible solutions, which is enormous in some domains [Reymond et al., 2012]. Other inference mechanisms, including marginalization, maximum a posteriori estimation, and expectation, have great potential in designing graphs with desired profiles.

Sum-product networks (SPNs) [Poon and Domingos, 2011] are deep generative models for *fixed-size* tensor data. Their essential feature is that they are tractable, which means that—under certain assumptions—they guarantee to answer a large family of complex probabilistic queries exactly and efficiently [Vergari et al., 2021, Choi et al., 2020]. The existing work on SPNs for graphs is limited to representation learning [Zheng et al., 2018, Errica and Niepert, 2023]. Therefore, we propose GraphSPNs, deep (one-shot) generative models for tractable probabilistic inference on graphs. There are two main challenges in designing such models. (i) SPNs are probability distributions defined on a *fixed-dimensional* space. Hence, we have to deal with the fact that each instance of a graph has a different number of nodes and edges. (ii) Graphs are permutation invariant objects [Veitch and Roy, 2015]. Consequently, our models must be agnostic to re-ordering the nodes in a graph. In other words, the probability of a graph has to remain unchanged when permuting the nodes (we want to recognize the same graph up to all its permutations). We address (i) in a pragmatic way, using virtual nodes. This principle is frequently adopted in deep generative models for graphs [Madhawa et al., 2019]. However, the real difficulty is to deal with (ii). Indeed, learning a permutation invariant distribution is hard since the number of modes of the target data distribution is much higher than a non-invariant, canonical distribution. Therefore, this paper investigates different techniques to make GraphSPNs permutation invariant.

Accepted for the 7th Workshop on Tractable Probabilistic Modeling at UAI (TPM 2024).

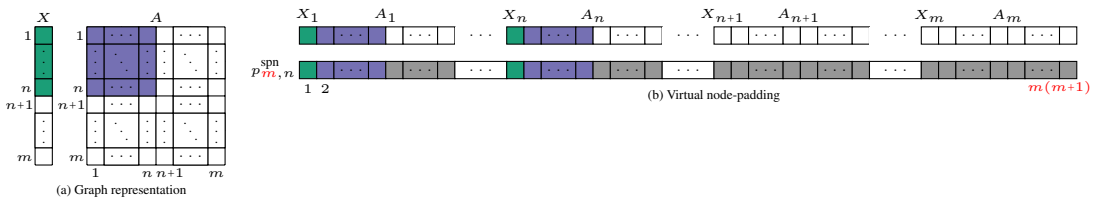


Figure 1: *Graph representation.* (a) Let G be a graph represented by a feature matrix, $X \in \mathcal{X}^n$, and an adjacency tensor, $A \in \mathcal{A}^{n \times n}$. We consider each instance of G (highlighted in green and blue) to have a random number of nodes, $n \in (0, 1, \ldots, m)$, but we expect it to have at most m nodes. The remaining places (white) are empty and are not included in the training data. (b) Virtual node-padding fills in the empty places with virtual nodes (grey), which requires us to extend $\mathcal{X}$ by an extra category, $\mathcal{X} := (0, 1)^{q+1}$.

2 BACKGROUND

Sum-product networks. A tensorized SPN [Peharz et al., 2020a,b, Loconte et al., 2024] is a deep learning model of a probability distribution, $p(X)$, over a *fixed-size* tensor, $X \in \mathcal{X}$. The network contains several layers of computational units (similar to neural networks [Vergari et al., 2019]). Each layer is defined over its *scope*, $\psi \subseteq X$, i.e., a subset of the input. There are three types of layers, depending on the units they encapsulate: sum layer L_S, product layer L_P, and input layer L_I. The units of *input* layers are user-defined probability distributions, $p_i(\psi)$. For n_I units, an input layer computes $p_i(\psi)$ for $i \in (1, \ldots, n_I)$ and outputs an n_I-dimensional vector of probabilities $\mathbf{l}$. The units of *product* layers are factored distributions, applying conditional independence over a pair-wise disjoint partition of their scope. A product layer receives outputs from n layers, $(\mathbf{l}_1, \ldots, \mathbf{l}_n)$, and computes either an Hadamard product, $\mathbf{l} = \odot_{i=1}^{n} \mathbf{l}_i$, or Kronecker product, $\mathbf{l} = \otimes_{i=1}^{n} \mathbf{l}_i$. The units of *sum* layers are mixture distributions. For n_S units, a sum layer receives an n-dimensional input, $\mathbf{l}$, from a previous layer and computes $\mathbf{W}\mathbf{l}$, where $\mathbf{W}$ is an $n_S \times n$ matrix of row-normalized weights. The output (layer) of a tensorized SPN is typically a sum layer.

The key benefit of SPNs is that they provide *tractable* inference over arbitrary subsets of X. However, to this end, the sum units have to satisfy *smoothness*, the product units have to satisfy *decomposability*, and the input units have to be *tractable* [Choi et al., 2020, Vergari et al., 2021].

Permutation invariance. Exchangeable data structures, including sets, graphs, partitions, and arrays [Orbanz and Roy, 2014], have a factorial number of possible configurations (orderings). Permutation invariance says that no matter a selected configuration, the probability of an exchangeable data structure has to remain the same. To define the permutation invariance of a probability distribution, let $\mathbb{S}_n$ be a finite symmetric group of a set of n elements. This is a set of all $n!$ permutations of $[n] := (1, \ldots, n)$, where any of its members, $\pi \in \mathbb{S}_n$, permutes an n-dimensional vector, $X := (X_1, \ldots, X_n)$, as follows: $\pi X = (X_{\pi(1)}, \ldots, X_{\pi(n)})$. The probability distribution p is permutation invariant iff $p(X) = p(\pi X)$ for all $\pi \in \mathbb{S}_n$. X is permutation invariant if $p(X)$ is.

SPNs are only partially permutation invariant [Papež et al., 2024], and so, for this paper, we consider SPNs as permutation-sensitive distributions.

Problem definition. Let $G := (X, A)$ be an n-node graph characterized by a feature matrix, $X \in \mathcal{X}^n$, and an adjacency tensor, $A \in \mathcal{A}^{n \times n}$. For example, to express G with q types of nodes and r types of edges, we use one-hot encoding to define $\mathcal{X} := (0, 1)^q$ and $\mathcal{A} := (0, 1)^{r+1}$, where the extra category (+1) in $\mathcal{A}$ is to reflect the fact that there can be no connections between two nodes. Our objective is to learn a *tractable* probability distribution over graphs, $p(G)$, given a collection of observed graphs, $(G_1, \ldots, G_N)$, where each G_i has a different number of nodes and edges. We would like our GraphSPNs to inherit the ability to tractably answer the same inference quires as the conventional SPNs.

3 GRAPH SUM-PRODUCT NETWORKS

A graph sum-product network (GraphSPN) is a probability distribution over a graph, $p(G)$, where G is random in its values and size. We conveniently use the fact that G is represented by the feature matrix and the adjacency tensor, $G := (X, A)$, and design the GraphSPN as a joint probability distribution of X and A,

$$p(G) := g_n(p^{\text{spn}}_{m,n}(X, A)), \tag{1}$$

where g_n is an operator ensuring the permutation invariance, and $p^{\text{spn}}_{m,n}$ is a permutation-sensitive SPN, as described in the next paragraph. The joint support of $p^{\text{spn}}_{m,n}$ leaves us with many choices to reorganize the elements of X and A. We first concatenate X and A to form an n by $n+1$ matrix and then vectorize this matrix in a row-wise manner (Figure 1). In this way, there is always a node followed by its associated edges, $(X, A) := (X_1, A_1, \ldots, X_n, A_n)$, where $X_i \in \mathcal{X}$ and $A_i \in \mathcal{A}^n$ are the ith slice of X and A along the first dimension, respectively.

Virtual node-padding. SPNs assume fixed-size inputs (Section 2). To deal with the varying size of G, we design the SPN in (1) as $p^{\text{spn}}_{m,n}(X, A) := p^{\text{spn}}_{m}(\texttt{padding}_{m,n}(X, A))$.

It is a composition of a padding operator, $\texttt{padding}_{m,n}$, and a permutation-sensitive SPN, p^{spn}_m, whose support (the root scope) has the size $m(m+1)$. $\texttt{padding}_{m,n}$ extends $\mathcal{X}$ by an extra category to express that there is no node at a specific position in the graph. Consequently, each n-node instance of G is filled in with virtual nodes (and associated virtual edges) to match the maximal number of nodes, m, which we expect in the training data (Figure 1b). This technique—which we refer to as *virtual node-padding*—is often used in deep generative models for graphs [Madhawa et al., 2019].

Exact permutation invariance. The general way to design permutation invariant $p(G)$ is to make g_n in (1) an average over all permutations $\pi \in \mathbb{S}_n$ of (X, A),

$$p(G) \coloneqq \frac{1}{n!} \sum_{\pi \in \mathbb{S}_n} p^{\text{spn}}_{m,n}(\pi X, \pi A), \tag{2}$$

where $\pi A \coloneqq (\pi A_{\pi_1}, \ldots, \pi A_{\pi_n})$. This approach can be seen as artificially extending each instance of (X, A) to $n!$ of its permutations. All the components $p^{\text{spn}}_{m,n}$ of the mixture (2) are an identical SPN, i.e., the components share the same parameterization. This approach is computationally costly, as it requires $n!$ passes through $p^{\text{spn}}_{m,n}$, the scope of which has $m(m+1)$ entries. Note that (2) is known as the Janossy distribution [Daley and Vere-Jones, 2003].

Sorting. Any graph G has up to $n!$ equivalent permutations, πG, $\pi \in \mathbb{S}_n$. If we ensure that $p^{\text{spn}}_{m,n}$ always sees the same, user-defined, canonical representation of G, then the permutation invariance of (1) is guaranteed. One way to find such a representation is to *sort* each instance of G before entering $p^{\text{spn}}_{m,n}$. Then, composing a sorting function, $\texttt{sort}$, with $p^{\text{spn}}_{m,n}$ avoids the summation over $n!$ terms in (2), and thus circumvents the use of g_n, as follows:

$$p(G) \coloneqq p^{\text{spn}}_{m,n}(\texttt{sort}(X, A)), \tag{3}$$

where it holds that $\texttt{sort}(\pi X, \pi A) = \texttt{sort}(X, A)$ for all $\pi \in \mathbb{S}_n$. The computational complexity of (3) depends on a specific sorting algorithm, e.g., $\mathcal{O}(n \log n)$. After the sorting, there is the need for only a single forward pass through $p^{\text{spn}}_{m,n}$, a significant reduction compared to (2). However, certain orderings are more suitable than others [Montavon et al., 2012, Niepert et al., 2016, Defferrard et al., 2016], and the right ordering often depends on domain knowledge.

Approximate permutation invariance. The exact invariance (2) is computationally infeasible for all but small graphs. Indeed, even for G with, say, 8 nodes, we have to perform 8! (40320) passes through $p^{\text{spn}}_{m,n}$, which is prohibitive when $p^{\text{spn}}_{m,n}$ is large. This permutation mechanism will slow down the inference as well. For example, marginalizing out certain node(s) has to be done 8! times. Therefore, we consider different ways to relax the exact permutation invariance, which have been used to design approximately permutation invariant neural networks, i.e., transformations $\mathbb{R}^u \to \mathbb{R}^v$ [Murphy et al., 2019b,a, Wagstaff et al., 2022]. We cast them into their probabilistic interpretation, such that they are probability distributions $\mathbb{R}^u \to \mathbb{R}^+$.

k-ary sub-graphs. The k-ary permutation invariance reduces the complexity of (2) by averaging over all k-node induced sub-graphs of G,

$$p(G) \approx \frac{(n-k)!}{n!} \sum_{\mathbf{t} \in \mathbb{T}^k_n} p^{\text{spn}}_{k,n}(X_{\mathbf{t}}, A_{\mathbf{tt}}). \tag{4}$$

The k-node induced sub-graphs are represented by the feature sub-matrix, $X_{\mathbf{t}} \coloneqq (X_{t_1}, \ldots, X_{t_k}) \in \mathcal{X}^k$, and the adjacency sub-tensor, $A_{\mathbf{tt}} \coloneqq (A_{t_1 t_1}, \ldots, A_{t_k t_k}) \in \mathcal{A}^{k \times k}$. They are selected by a set of k indices, $\mathbf{t} \coloneqq (t_1, \ldots, t_k) \in \mathbb{T}^k_n$, where $\mathbb{T}^k_n$ is the set of all ways to choose k elements out of n unique elements (i.e., the set of all k-tuples of $[n]$). The main benefit of this method is that (i) the scope of $p^{\text{spn}}_{k,n}$ has a fixed size, $k(k+1)$, which also reduces the overall size of $p^{\text{spn}}_{k,n}$, and (ii) the number of passes through $p^{\text{spn}}_{k,n}$ is lower, $n!/(n-k)! < n!$. The expressivity of this approach grows with k. For $k = 1$, we obtain a probabilistic version of Deep Sets [Zaheer et al., 2017], whereas for $k = n$, we recover (2). For high k, we capture more dependencies among the nodes, whereas for low k, we sacrifice some of them.

Random sampling. The exact invariance (2) is a marginal distribution, $p(G) \coloneqq \sum_{\pi \in \mathbb{S}_n} p(\pi) p^{\text{spn}}_{m,n}(\pi X, \pi A)$, where the permutation π is the latent variable, and $p(\pi) \coloneqq 1/n!$ is the uniform distribution on $\mathbb{S}_n$. This interpretation suggests to reduce the complexity of the full permutation invariance by approximating (2) with the random average,

$$p(G) \approx \frac{1}{N} \sum_{\pi \in \mathbb{S}^N_n} p^{\text{spn}}_{m,n}(\pi X, \pi A). \tag{5}$$

Here, $\mathbb{S}^N_n$ is a subset of $N < n!$ random permutations from the full set $\mathbb{S}_n$, which are sampled without repetition. The expressivity of (5) depends on N. Even for $N = 1$, we can still achieve sufficient approximate permutation invariance, assuming we draw a new sample at each training step and perform the training sufficiently long. For $N = n!$, we recover (2). In the inference regime, it is important to use $N \gg 1$. However, we can suffer from the risk of drawing less meaningful permutations for a given graph.

4 EXPERIMENTS

We evaluate GraphSPNs in terms of their capacity to capture the underlying data distribution and their ability to generate realistic and novel graphs. We perform the experiments in the context of the computational design of molecular graphs. We provide the code at `https://github.com/mlnpapez/GraphSPN`.

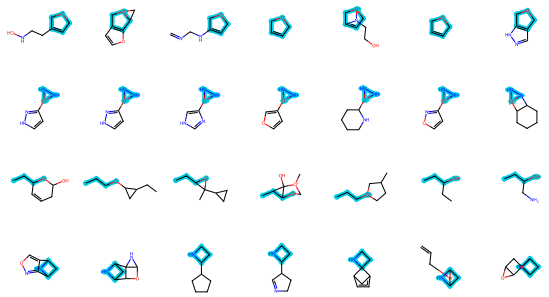


Figure 2: *Conditional molecule generation on the QM9 dataset.* The blue area highlights the known part of the molecule. There is one such known part per row. Each column corresponds to a new molecule that is generated conditionally on the known part.

Molecule generation. Deep generative models for molecular graphs have recently gained significant attention. They hold great potential in important applications, such as discovering drugs and materials with desired chemical properties. Given a dataset of molecular graphs, the task is to learn a probability distribution of chemically valid molecules, $p(G)$. This is an intricate combinatorial problem, where not all combinations of atoms and bonds can be connected, but the connections must satisfy chemical valency constraints. We want $p(G)$ to generate molecules that were not seen during the training and satisfy the chemical valency rules.

Dataset. We test GraphSPNs on QM9 [Ramakrishnan et al., 2014] dataset, a standard benchmark often used to assess deep generative models for molecular design. QM9 contains around 134k stable, small organic molecules with at most 9 atoms of 4 different types.

Metrics. We adopt the standard metrics for the molecule generation task. *Validity* (V) and *Validity w/o check* (V w/o check) are the percentages of chemically valid molecules (i.e., those not violating the chemical valency rules) in all generated molecules with and without any correction mechanisms, respectively. *Uniqueness* (U) is the percentage of all generated molecules that are valid and unique (i.e., not a duplicate of some other generated molecule). *Novelty* (N) is the percentage of valid molecules not in the training data.

GraphSPN variants. We consider the following GraphSPN variants: sort, k-ary, and rand that implement (3), (4), and (5), respectively, and the none variant with no invariance properties (g_n in (1) is identity).

Results. Table 1 shows that all the GraphSPN variants achieve 100% validity. This is because we use the posthoc validity correction mechanism from [Zang and Wang, 2020] to satisfy the valency rules. However, the effectiveness of this correction depends on the validity w/o check. Naturally, if we have to correct more (possibly large) molecules, this quickly becomes computationally expensive. From this perspective, the sort variant delivers the best results, as its validity w/o check is highest. This result is because sort imposes the canonical ordering of atoms in each molecule, which is not satisfied by the other GraphSPN variants. $p^{\text{spn}}_{m,n}$ of the none, rand, and sort variants has the same architecture. Still, it is hard for the none and rand variants to capture the underlying data distribution. This result leads us to conclude that departing from the canonical ordering yields a more complicated distribution (with more modes), which is harder to capture sufficiently well, as evidenced by the poor ability to learn the essence of this problem, i.e., the valency rules. In the top of Table 1, we show baseline models (see Section C). They are permutation-sensitive and also rely on the canonical ordering. We can see that the sort variant is competitive to—and sometimes even outperforms—these baselines.

Model	V	V w/o check	U	N
GraphVAE	55.70	n/a	76.00	61.60
GVAE	60.20	n/a	9.30	80.90
CVAE	10.20	n/a	67.50	90.00
RVAE	96.60	n/a	n/a	95.50
GraphNVP	83.10	n/a	99.20	58.20
GRF	84.50	n/a	66.00	58.60
GraphAF	**100.00**	67.00	94.20	88.80
GraphDF	**100.00**	82.70	97.60	98.10
MoFlow	**100.00**	**89.00**	98.50	96.40
ModFlow	n/a	99.10	99.30	**100.00**
GraphSPN: none	**100.00**	19.75	**100.00**	99.49
GraphSPN: rand	**100.00**	15.38	**100.00**	99.67
GraphSPN: sort	**100.00**	81.62	98.65	69.08
GraphSPN: k-ary	**100.00**	2.10	**100.00**	98.30

Table 1: *Unconditional molecule generation on the QM9 dataset.* The percentage of valid, valid w/o check, unique, and novel molecules for *intractable*, permutation-sensitive models that all rely on the canonical ordering of atoms (above) and the *tractable* GraphSPNs with various forms of permutation invariance (below).

To illustrate that GraphSPNs provide tractable probabilistic inference, Figure 2 displays conditional molecule generation for the sort variant. Here, it can be seen that each molecule's newly generated part varies in size and composition.

5 CONCLUSION

The study of permutation invariance in the SPN community has received limited attention. This paper has the ambition to make a few steps towards improving upon this state. We have investigated different ways of ensuring the permutation invariance of SPNs and proposed GraphSPNs, a novel class of tractable deep generative models for exact and efficient probabilistic inference over graphs. We show that GraphSPNs are competitive with existing deep generative models for graphs. Among the proposed GraphSPN variants, the best performance was achieved with the one based on canonical ordering. This shows that the canonical ordering simplifies the target data distribution and makes it easier for an SPN—which is less densely connected than the models based on neural networks—to be trained successfully.

Acknowledgements

The authors acknowledge the support of the GAČR grant no. GA22-32620S and the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".

References

Ole Barndorff-Nielsen. *Information and exponential families: In statistical theory*. John Wiley & Sons, 1978.

Payal Chandak, Kexin Huang, and Marinka Zitnik. Building a knowledge graph to enable precision medicine. *Scientific Data*, 10(1):67, 2023.

Y Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. *UCLA. URL: http://starai. cs. ucla. edu/papers/ProbCirc20. pdf*, 2020.

Kamal Choudhary, Brian DeCost, Chi Chen, Anubhav Jain, Francesca Tavazza, Ryan Cohn, Cheol Woo Park, Alok Choudhary, Ankit Agrawal, Simon JL Billinge, et al. Recent advances and applications of deep learning methods in materials science. *npj Computational Materials*, 8(1): 59, 2022.

Daryl J Daley and David Vere-Jones. *An introduction to the theory of point processes: volume I: elementary theory and methods*. Springer, 2003.

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

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.

Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. ETA prediction with graph neural networks in Google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3767–3776, 2021.

Federico Errica and Mathias Niepert. Tractable probabilistic graph representation learning with graph-induced sum-product networks. *arXiv preprint arXiv:2305.10544*, 2023.

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

Shion Honda, Hirotaka Akita, Katsuhiko Ishiguro, Toshiki Nakanishi, and Kenta Oono. Graph residual flow for molecular graph generation. *arXiv preprint arXiv:1909.13521*, 2019.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International conference on machine learning*, pages 2323–2332. PMLR, 2018.

Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. In *International Conference on Machine Learning*, pages 10362–10383. PMLR, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *International conference on machine learning*, pages 1945–1954. PMLR, 2017.

Greg Landrum et al. Rdkit: Open-source cheminformatics, 2006.

Lorenzo Loconte, Aleksanteri M Sladek, Stefan Mengel, Martin Trapp, Arno Solin, Nicolas Gillis, and Antonio Vergari. Subtractive mixture models via squaring: Representation and learning. In *The 12th International Conference on Learning Representations*, 2024.

Youzhi Luo, Keqiang Yan, and Shuiwang Ji. GraphDF: A discrete flow model for molecular graph generation. In *International conference on machine learning*, pages 7192–7203. PMLR, 2021.

Tengfei Ma, Jie Chen, and Cao Xiao. Constrained generation of semantically valid graphs via regularizing variational autoencoders. *Advances in Neural Information Processing Systems*, 31, 2018.

Kaushalya Madhawa, Katushiko Ishiguro, Kosuke Nakago, and Motoki Abe. Graphnvp: An invertible flow model for generating molecular graphs. *arXiv preprint arXiv:1905.11600*, 2019.

Grégoire Montavon, Katja Hansen, Siamac Fazli, Matthias Rupp, Franziska Biegler, Andreas Ziehe, Alexandre Tkatchenko, Anatole Lilienfeld, and Klaus-Robert Müller. Learning invariant representations of molecules for atomization energy prediction. *Advances in neural information processing systems*, 25, 2012.

Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pages 4663–4673. PMLR, 2019a.

Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *The 7th International Conference on Learning Representations*, 2019b.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023. PMLR, 2016.

Peter Orbanz and Daniel M Roy. Bayesian models of graphs, arrays and other exchangeable random structures. *IEEE transactions on pattern analysis and machine intelligence*, 37(2):437–461, 2014.

Milan Papež, Martin Rektoris, Václav Šmídl, and Tomáš Pevný. Sum-product-set networks: Deep tractable models for tree-structured graphs. In *The 12th International Conference on Learning Representations*, 2024.

Robert Peharz, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van den Broeck, Kristian Kersting, and Zoubin Ghahramani. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference on Machine Learning*, pages 7563–7574. PMLR, 2020a.

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pages 334–344. PMLR, 2020b.

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011.

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.

Jean-Louis Reymond, Lars Ruddigkeit, Lorenz Blum, and Ruud Van Deursen. The enumeration of chemical space. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 2(5):717–733, 2012.

Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. GraphAF: A flow-based autoregressive model for molecular graph generation. *arXiv preprint arXiv:2001.09382*, 2020.

Andy Shih, Guy Van den Broeck, Paul Beame, and Antoine Amarilli. Smoothing structured decomposable circuits. *Advances in Neural Information Processing Systems*, 32, 2019.

Martin Simonovsky and Nikos Komodakis. GraphVAE: Towards generation of small graphs using variational autoencoders. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*, pages 412–422. Springer, 2018.

Victor Veitch and Daniel M Roy. The class of random graphs arising from exchangeable random measures. *arXiv preprint arXiv:1512.03099*, 2015.

Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Visualizing and understanding sum-product networks. *Machine Learning*, 108(4):551–573, 2019.

Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations: From simple transformations to complex information-theoretic queries. *arXiv preprint arXiv:2102.06137*, 2021.

Yogesh Verma, Samuel Kaski, Markus Heinonen, and Vikas Garg. Modular flows: Differential molecular generation. *Advances in neural information processing systems*, 35: 12409–12421, 2022.

Edward Wagstaff, Fabian B Fuchs, Martin Engelcke, Michael A Osborne, and Ingmar Posner. Universal approximation of functions on sets. *Journal of Machine Learning Research*, 23(151):1–56, 2022.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

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*, pages 5708–5717. PMLR, 2018.

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.

Chengxi Zang and Fei Wang. Moflow: An invertible flow model for generating molecular graphs. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 617–626, 2020.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):249–270, 2020.

Kaiyu Zheng, Andrzej Pronobis, and Rajesh Rao. Learning graph-structured sum-product networks for probabilistic semantic maps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

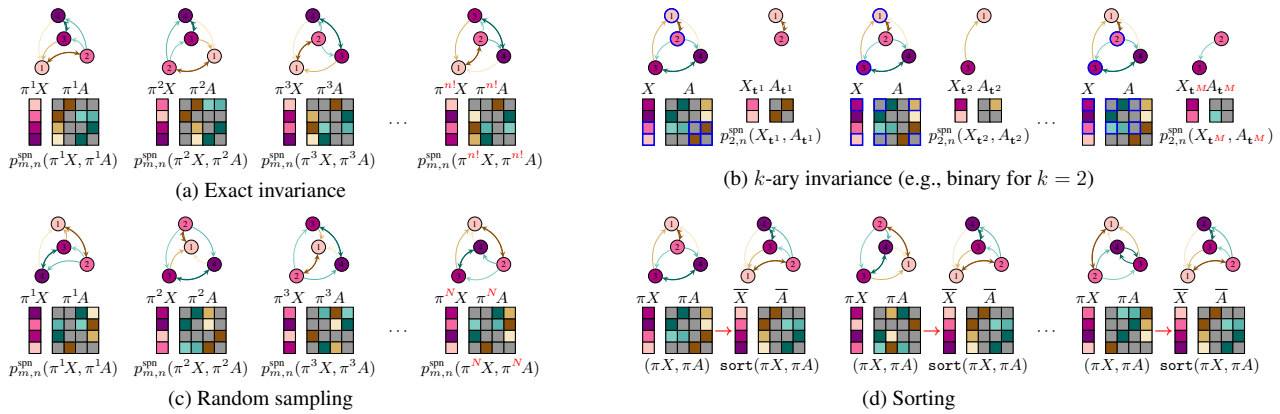


Figure 3: *An illustration of the key principles behind various GraphSPNs.* (a) The exact permutation invariance first computes $p^{\text{spn}}_{m,n}$ for all $n!$ permutations and then averages the results in (2). (b) The k-ary permutation invariance approximates the exact invariance (a) by computing $p^{\text{spn}}_{k,n}$ (with the root scope of size $k(k+1)$) for all ways to choose k-node sub-graphs from the n-node original graph, G, without repetition and with the order, and then averaging the $M = n!/(n-k)! < n!$ results in (4). (c) The random sampling approximates the exact invariance (a) by computing the average only for $N < n!$ permutations of G in (5). (d) The sorting approach is also exact, but it simplifies the target data distribution by first imposing the same canonical ordering of G and then computing $p^{\text{spn}}_{m,n}$, as displayed in (3).

A MORE NOTES ON PERMUTATION INVARIANCE

Figure 3 illustrates the fundamental principles underlying the GraphSPN variants discussed in Section 3.

B TRACTABILITY

Tractability. An SPN is tractable if it answers probabilistic queries *exactly* and *efficiently*. Here, exact means that the answers do not involve any approximation or heuristic, and efficient means that the answers can be obtained (computed) in polynomial time [Choi et al., 2020]. We are interested in probabilistic queries that can collectively be expressed in terms of the following expectation:

$$\nu(f) := \int f(G)p(G)dG, \tag{6}$$

where $f(G)$ is a function that allows us to formulate a desired query over (a part of) G. We defer concrete examples of $f(G)$ to Section B.1. The expectation (6) admits a closed-form solution only if both f and p satisfy certain structural constraints.

Assumption 1. *(Constraints on p). We consider the following constraints. (i)* ***Smoothness****: each $u \in \mathsf{L_S}$ satisfies $\forall a, b \in \mathbf{in}(u) : \psi_a = \psi_b$, where $\mathbf{in}(u)$ is the set of inputs of a u-unit. (ii)* ***Decomposability****: each $u \in \mathsf{L_P}$ satisfies $\forall a, b \in \mathbf{in}(u) : \psi_a \cap \psi_b = \varnothing$. (iii)* ***Tractable input layers****: $\int f_u(\psi_u)_u p(\psi_u) d\psi_u$ is tractable for each $u \in \mathsf{L_I}$.*

Assumption 1(iii) requires that each $p_i(\psi)$ is from a tractable family of probability distributions [Barndorff-Nielsen, 1978] and $f_i(\psi)$ is such that the integral admits an algebraically closed-form solution.

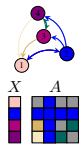
Assumption 2. *(Constraints on f). The function $f(G) := f(X_1, A_1 \ldots, X_n, A_n)$ is* omni-compatible *[Vergari et al., 2021] with respect to $p^{spn}_{m,n}$.*

Proposition 1. *(Tractability of GraphSPNs.) Let $p(G)$ be a GraphSPN* (1) *satisfying Assumptions 1, and let $f(G)$ be a function satisfying Assumption 2. Then, the integral* (6) *is tractable.*

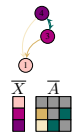
Proposition 1 covers all the GraphSPN variants (2), (3), and (5), except the k-ary sub-graphs (4). Indeed, the average in (2) and (5) can be seen as a sum unit with fixed uniform weights, which satisfies the smoothness assumption. The sorting operation in (3) does not affect the tractability. However, regarding the k-ary GraphSPNs, the average in (4) does not satisfy the smoothness assumption (i.e., it is not a smooth sum unit), as each of its children has a different scope. To make the k-ary variant tractable, we must perform the smoothing [Shih et al., 2019, Choi et al., 2020].

$$f(X,A) = h(X_2)h(A_{22}) \prod_{k \notin \mathbf{a}} h(A_{2k}) \prod_{t \notin \mathbf{a}} h(X_t) \Big(\prod_{u \notin \mathbf{a}} h(A_{tu}) h(A_{t2}) \Big)$$
$$= \mathbb{1}_{\mathcal{X}}(X_2)\mathbb{1}_{\mathcal{A}}(A_{22}) \prod_{k \notin \mathbf{a}} \mathbb{1}_{\mathcal{A}}(A_{2k}) \prod_{t \notin \mathbf{a}} \mathbb{1}_{\overline{\mathcal{X}}_t}(X_t) \Big(\prod_{u \notin \mathbf{a}} \mathbb{1}_{\overline{\mathcal{A}}_{tu}}(A_{tu}) \mathbb{1}_{\mathcal{A}}(A_{t2}) \Big)$$

(a) An example of (7) for $\mathbf{a} := \{2\}$



(b) The corresponding node of G



(c) G after the marginal query

Figure 4: *An example of a tractable inference query over a graph.* (a) An instantiation of the omni-compatible function (7) over a graph G for $\mathbf{a} := \{2\}$. The blue color highlights the targeted node and its associated edges. (b) A visual and matrix representation of G, where the targeted node and its associated edges, which correspond to (a), are highlighted in blue. (c) After targeting the node and its connected edges, we can perform, e.g., the marginal query and obtain a marginal graph $\overline{G}$.

B.1 TRACTABLE INFERENCE QUERIES OVER GRAPHS

Querying r-node sub-graphs. An omni-compatible function f (Assumption 2) that targets an r-node sub-graph of G can be expressed as follows:

$$f(X,A) := \prod_{i \in \mathbf{a}} h(X_i) \left(\prod_{j \in \mathbf{a}} h(A_{ij}) \prod_{k \notin \mathbf{a}} h(A_{ik}) \right) \prod_{t \notin \mathbf{a}} h(X_t) \left(\prod_{u \notin \mathbf{a}} h(A_{tu}) \prod_{v \in \mathbf{a}} h(A_{tv}) \right), \tag{7}$$

where $\mathbf{a} := \{a_1, \ldots, a_r\} \subseteq \{1, \ldots, n\}$ specifies indices of r nodes presented in a queried sub-graph.

Marginalizing selected nodes and (or) edges. The marginal query is realized by an appropriate choice of h for X_i and A_{ij}. The choice $h(X_i) := \mathbb{1}_{\mathcal{B}}(X_i)$, where $\mathbb{1}_{\mathcal{B}}$ is the indicator function and $\mathcal{B} := \mathcal{X}$, corresponds to marginalizing node feature of i-th node in the graph G. The evidence query is obtained for $\mathcal{B} := X_i$. Similarly, choosing $h(A_{ij}) := \mathbb{1}_{\mathcal{B}}(A_{ij})$ and $\mathcal{B} := \mathcal{A}$ corresponds to marginalizing edge between nodes i and j in the graph G. Again, the evidence query is given by setting $\mathcal{B} := A_{ij}$. We demonstrate an example of the marginal query for $\mathbf{a} := \{2\}$ in Figure 3.

C EXPERIMENTAL DETAILS

Preprocessing. We use the RDKit library [Landrum et al., 2006] to first kekulize the molecules and then remove the hydrogen atoms. The final molecules contain only the single, double, and triple bonds. Since we aim to test the permutation invariance of various GraphSPNs, we randomly permute the atoms in each molecule during the preprocessing.

Baselines. We compare GraphSPNs with various molecular deep generative models: grammar variational autoencoder (GVAE) [Kusner et al., 2017], character VAE (CVAE) [Gómez-Bombarelli et al., 2018], regularizing VAE (RVAE) [Ma et al., 2018], junction tree VAE (JT-VAE) [Jin et al., 2018], graph VAE (GraphVAE) [Simonovsky and Komodakis, 2018], graph real-valued non-volume preserving (GraphNVP) flow [Madhawa et al., 2019], graph autoregressive flow (GraphAF) [Shi et al., 2020], graph discrete flow (GraphDF) [Luo et al., 2021], molecular flow (MoFlow) [Zang and Wang, 2020], modular flow (ModFlow) [Verma et al., 2022], and graph residual flow (GRF) [Honda et al., 2019].

Architecture. To implement the permutation-sensitive part of GraphSPNs, we adopt the EinSum networks [Peharz et al., 2020a]. The expressive power of this monolithic and tensorized variant of SPNs is driven by four hyper-parameters: the number of layers $n_l \in \{1, 2, 3\}$, the number of sum units $n_S \in \{10, 40, 80\}$, the number of input units $n_I \in \{10, 40\}$, and the number of repetitions $n_R \in \{10, 40, 80\}$. The input units are categorical distributions. Molecules are undirected acyclic graphs. To satisfy this constraint, we use only the lower triangular part of A in the sampling procedure.

For the rand variant of GraphSPNs (5), we use $\mathbb{S}_m^{20}$, i.e., 20 permutations. For the k-ary variant (4), we use $k = 2$ to obtain the average with only 72 elements. To impose the canonical ordering with the sort variant (3), we use the RDKit library.

Learning. The results for the baseline methods in the top part of Table 1 are obtained from their respective papers. For all the GraphSPN variants in the bottom part of Table 1, we minimize negative log-likelihood for 40 epochs using the ADAM optimizer [Kingma and Ba, 2014] with 256 samples in the minibatch, step-size $\alpha = 0.05$ and decay rates $\beta_1 = 0.9$ and $\beta_2 = 0.82$. All experiments are repeated 5 times with different initialization of the model's parameters. We sample 4000 molecules to compute the metrics introduced in Section 4.

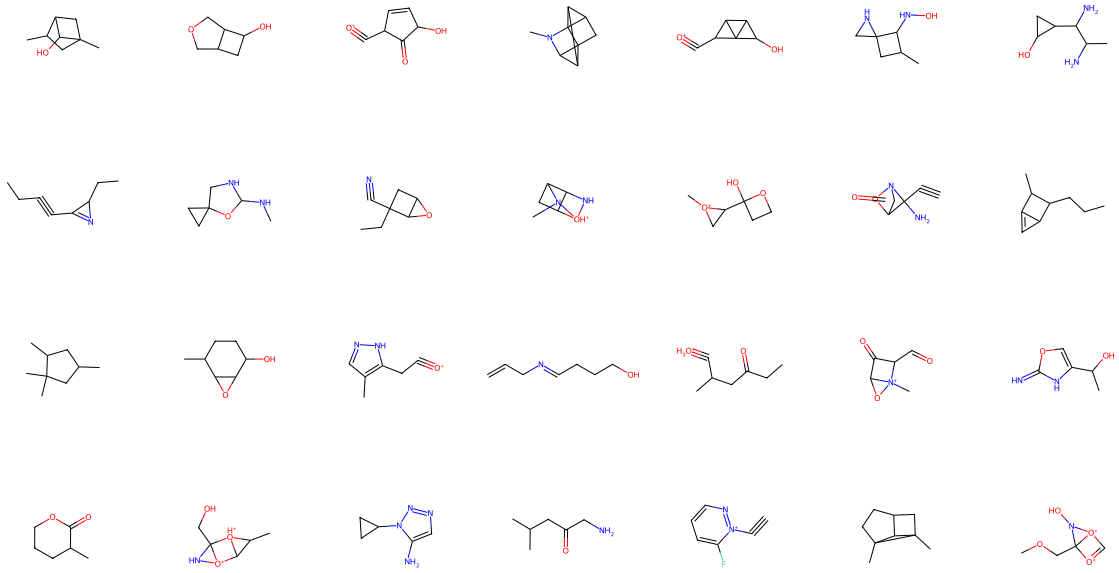


Figure 5: *Unconditional molecule generation on the QM9 dataset.*

D ADDITIONAL RESULTS

Figure 5 shows unconditional samples of molecules from the sort variant (3) that was trained on the QM9 dataset. The resulting molecules resemble those from the training data, yet they are all newly discovered molecules.