Sum-Product-Set Networks

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

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

Abstract

Daily internet communication relies heavily on tree-structured graphs, embodied by popular data formats such as XML and JSON. However, many recent generative (probabilistic) models utilize neural networks to learn a probability distribution over undirected cyclic graphs. This assumption of a generic graph structure brings various computational challenges, and, more importantly, the presence of non-linearities in neural networks does not permit tractable probabilistic inference. We address these problems by proposing sum-product-set networks, an extension of probabilistic circuits from unstructured tensor data to tree-structured graph data. To this end, we use random finite sets to reflect a variable number of nodes and edges in the graph and to allow for exact and efficient inference. We demonstrate that our tractable model performs comparably to various intractable models based on neural networks.

1 INTRODUCTION

One of the essential paradigm shifts in artificial intelligence and machine learning over the last years has been the transition from probabilistic models over fixed-size unstructured data (tensors) to probabilistic models over variable-size structured data (graphs) [Bronstein et al., 2017, Wu et al., 2021]. Tree-structured data are a specific type of generic graph-structured data which describe real or abstract objects (vertices) and their hierarchical relations (edges). These data structures appear in many scientific domains, including cheminformatics [Bianucci et al., 2000], physics [Kahn et al., 2022], and natural language processing [Ma et al., 2018]. They are used by humans in data-collecting mechanisms to organize knowledge into various machine-generated formats, such as JSON [Pezoa et al., 2016], XML [Tekli et al., 2016] and YAML [Ben-Kiki et al., 2009] to mention a few.

The development of models for tree-structured data has been thoroughly, but almost exclusively, pursued in the NLP domain [Tai et al., 2015, Zhou et al., 2016, Cheng et al., 2018, Ma et al., 2018]. However, these models rely solely on variants of neural networks (NNs) and are non-generative, lacking clear probabilistic interpretation. Despite a growing interest in designing generative models for general graph-structured data [Simonovsky and Komodakis, 2018, De Cao and Kipf, 2018, You et al., 2018, Jo et al., 2022, Luo et al., 2021], there are no generative models that take advantage of the parent-child ancestry inherent in tree-structured graphs. Therefore, directly applying these generic models to the trees would incur unnecessary computational costs. Moreover, these models assume the input features with homogeneous dimensions. More importantly, they preclude tractable probabilistic inference, necessitating approximate techniques to answer even basic queries.

In sensitive applications (e.g., healthcare, finance, and cybersecurity), there is an increasing legal concern about providing non-approximate and fast decision-making. Probabilistic circuits (PCs) [Vergari et al., 2020] are *tractable* probabilistic (generative) models that guarantee to answer a large family of complex probabilistic queries [Vergari et al., 2021] exactly and efficiently.

To the best of our knowledge, no PC is designed to represent a probability distribution over a tree-structured graph. Here, it is essential to note that a PC is also a graph. To distinguish between the two graphs, we refer to the former as the *data* graph and to the latter as the *computational* graph. Similarly, to distinguish between the vertices of these two graphs, we refer to vertices of the data graph and computational graph as data *nodes* and computational *units*, respectively. We propose a new PC by seeing the data graph as a recursive hierarchy of sets, where each parent data node is a set of its child data nodes. We use the theory of random finite sets [Nguyen, 2006] to induce a probability distribution over

Accepted for the 6th Workshop on Tractable Probabilistic Modeling at UAI (TPM 2023).

repeating data subgraphs, i.e., sets with identical properties. This allows us to extend the computational graph with an original computational unit, a *set* unit. Our model provides an efficient sampling of new data graphs and exact marginalization of selected data nodes. It also permits the data nodes to have heterogeneous features, where each node can represent data of different modalities.

In summary, this paper offers the following contributions:

- We propose sum-product-set networks (SPSNs), a novel type of PCs that enables tractable and efficient inference for complex tree-structured data.
- We show that SPSNs provide similar (or better) performance to (than) models relying on NNs.

2 TREE-STRUCTURED DATA

A single instance of tree-structured, heterogeneous data is given by an attributed data graph, T. In contrast to a fixed-size, *unstructured*, random variable, $\mathbf{x} = (x_1, \ldots, x_d) \in \mathsf{X} \subset \mathbb{R}^d$, this graph forms a hierarchy of random-size sets.

Definition 1. *(Data graph). $T = (V, E, X)$ is an attributed, tree-structured graph, where V is a set of vertices, E is a set of edges, and X is a set of attributes (features). $V \coloneqq (L, H, O)$ splits into three subsets of data nodes: leaf nodes, L, heterogenenous nodes, H, and homogeneous nodes, O. Let $\mathbf{ch}(v)$ and $\mathbf{pa}(v)$ denote the set of child and parent nodes of $v \in V$, respectively. All elements of $\mathbf{ch}(v)$ are of an identical type if $v \in O$, and some or all elements of $\mathbf{ch}(v)$ are of a different type if $v \in H$. We assume that only the leaf nodes are attributed by $\mathbf{x}_v \in \mathsf{X}_v \subseteq \mathbb{R}^{d_v}$, with possibly different space and its dimension for each $v \in L$.*

Definition 2. *(Schema). Let T be an instance of tree-structured, heterogeneous data. Then, a subtree, S, which results from T by following all children of each $u \in H$ and only one child of each $u \in O$—such that it allows us to reach the deepest level of T—is referred to as the* schema.

Definition 1 implies that T is defined recursively by a subtree, $T_v \coloneqq \{T_{u_1} \ldots, T_{u_m}\}$, rooted at $v \in V$. For heterogeneous nodes, $v \in H$, each child subtree, T_u, has a different schema for all $u \in \mathbf{ch}(v)$. For homogeneous nodes, $v \in O$, each child, T_u, has the same schema for all $u \in \mathbf{ch}(v)$. The leaf nodes, $v \in L$, are terminal nodes (with no children), containing a feature vector, $T_v \coloneqq \mathbf{x}_v$. The cardinality of all homogeneous nodes, $v \in O$, is random and differs for each instance of T. This also randomizes the cardinality of heterogeneous nodes, $v \in H$, if $\mathbf{ch}(v)$ contains at least one homogeneous node, $u \in O$. We show an example of a single instance of tree-structured data in Figure 1(a).

Problem definition. Our objective is to learn a probability density over tree-structured graphs (Definition 1), $p(T)$, given a collection of observed graphs $\{T_1, \ldots, T_n\}$, where each T_i contains a different number of vertices and edges.

3 SUM-PRODUCT-SET NETWORKS

A sum-product-set network (SPSN) is a probability density over tree-structured, heterogeneous data, $p(T)$. This differs from the conventional sum-product network [Poon and Domingos, 2011], which is a probability density over the unstructured data, $p(\mathbf{x})$[1]. We define an SPSN by a parameterized computational graph, $\mathcal{G}$, and a scope function, ψ.

Definition 3. *(Computational graph). $\mathcal{G} \coloneqq (\mathcal{V}, \mathcal{E}, \theta)$ is a parameterized, directed, acyclic graph, where $\mathcal{V}$ is a set of vertices, $\mathcal{E}$ is set of edges, and $\theta \in \Theta$ are parameters. $\mathcal{V} \coloneqq (\mathsf{S}, \mathsf{P}, \mathsf{B}, \mathsf{L})$ contains four subsets computational units: sum units, S, product units, P,* set *units, B, and leaf units, L. The sum units and product units have multiple children; however, as detailed later, the set unit has only two children, $\mathbf{ch}(n) \coloneqq \{u, v\}$, $n \in \mathsf{B}$. θ contains parameters of sum units, i.e., non-negative and normalized weights, $\{w_{n,c}\}_{c \in \mathbf{ch}(n)}$, $w_{n,c} \geq 0$, $\sum_{c \in \mathbf{ch}(n)} w_{n,c} = 1$, $n \in \mathsf{S}$, and parameters of leaf units which are specific to possibly different densities.*

Definition 4. *(Scope function). The mapping $\psi : \mathcal{V} \to 2^T$—from the set of units to the power set of T—outputs a subset of T for each $n \in \mathcal{V}$ and is referred to as the* scope function. *If n is the root unit, then $\psi(n) = T$. If n is a sum unit, product unit, or set unit, then $\psi(n) = \bigcup_{c \in \mathbf{ch}(n)} \psi(c)$.*

Each unit of the computational graph (Definition 3), $n \in \mathcal{V}$, induces a probability density over a given node of the data graph (Definition 1), $v \in V$. The functionality of this density, $p_n(T_v)$, depends on the type of the computational unit.

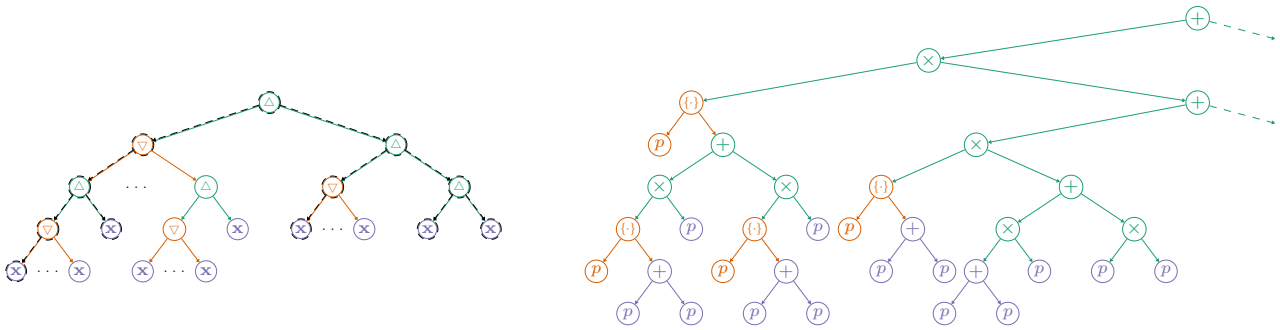
The **sum unit** computes the mixture density, $p_n(T_v) = \sum_{c \in \mathbf{ch}(n)} w_{n,c} p_c(T_v)$, $n \in \mathsf{S}$, $v \in \{L, H\}$, where $w_{n,c}$ is the weight connecting the sum unit with a child unit. The **product unit** computes the factored density, $p_n(T_v) = \prod_{c \in \mathbf{ch}(n)} p_c(\psi(c))$, $n \in \mathsf{P}$, $v \in \{L, H\}$. It introduces conditional independence among the scopes of its children, $\psi(c)$, establishing unweighted connections between this unit and its child units. The **leaf unit** computes a user-defined probability density, $p_n(x_v)$, $n \in \mathsf{L}$, $v \in L$. It is defined over a subset x_v of $T_v \coloneqq \mathbf{x}_v$ given by the scope, $\psi(n)$, which can be univariate or multivariate [Peharz et al., 2015].

The **set unit** computes a probability density of a finite random set

$$p_n(T_v) = p(m) U^m m! p(T_{u_1}, \ldots, T_{u_m}), \qquad (1)$$

$n \in \mathsf{B}$, $v \in O$, where $p(m)$ is the cardinality distribution and $p(T_{u_1}, \ldots, T_{u_m})$ is the feature density (conditioned on m). These are the two children of the set unit (Definition 3),

[1] For a short introduction to PCs and conventional SPNs, see the supplementary material.



(a) Data graph

(b) Computational graph

Figure 1: *Sum-product-set networks.* (a) The tree-structured, heterogeneous, data graph, T, (Definition 1), and the *schema* (dashed line), S, (Definition 2). Here, $\triangle$ is the heterogeneous node, $\triangledown$ is the homogeneous node, and $\mathbf{x}$ denotes the leaf node of T. (b) The computational graph, $\mathcal{G}$, of an SPSN (Definition 3) designed based on S, where $+$, $\times$, $\{\cdot\}$ and p are the sum unit, product unit, *set* unit and leaf unit of $\mathcal{G}$, respectively. The subtrees of T in (a) are modeled by the corresponding parts of $\mathcal{G}$ in (b), as displayed in green, orange, and blue.

spanning computational subgraphs on their own (Figure 1). The random finite set, $T_v \coloneqq \{T_{u_1}, \ldots, T_{u_m}\} \in 2^{T_v}$, is a simple, finite point process [Van Lieshout, 2000, Daley et al., 2003, Nguyen, 2006, Mahler, 2007], where 2^{T_v} is the power set of T_v. T_v is an unordered set of *distinct* features or other sets, such that not only the individual elements, $T_u \in 2^{T_u}$, are random, but also the number of these elements, $m \coloneqq |T_v| \in \mathbb{N}_0$, is random. The proportionality factor, $m! = \prod_i^m i$, comes from the symmetry of $p(T_{u_1}, \ldots, T_{u_m})$. It reflects the fact that $p(T_{u_1}, \ldots, T_{u_m})$ is permutation invariant, i.e., it gives the same value to all $m!$ possible permutations of $\{T_{u_1}, \ldots, T_{u_m}\}$. U is the unit of hyper-volume in T_u, which ensures that (1) is unit-less by canceling out the units of $p(T_{u_1}, \ldots, T_{u_m})$ with U^m.

Assumption 1. *(Requirements on the set unit). The requirements for a probability density of the set unit to be properly defined are as follows: (a) each element of $T_v \coloneqq \{T_{u_1}, \ldots, T_{u_m}\}$ resides in the same space, i.e., $T_u \in 2^{T_u}$, for all $u \in \mathbf{ch}(v)$; (b) the elements $\{T_{u_1}, \ldots, T_{u_n}\}$ are independent and identically distributed; and (c) the realizations $\{T_{u_1}, \ldots, T_{u_n}\}$ are distinct.*

Given that Assumption 1 is satisfied, (1) contains the product of m densities, $p(T_{u_1}, \ldots, T_{u_m}) = \prod_{i=i}^m p(T_{u_i})$ over the identical scope (i.e., it does not have the disjoint scope and, therefore, it is not the product unit). Note that the feature density treats $\{T_{u_i}\}_{i=i}^m$ as instances, simply aggregating them by the product of distributions. It is a single density indexed by the same set of parameters for each $\{T_{u_i}\}_{i=i}^m$. For example, if we choose $p(m)$ as the Poisson distribution, then (1) is the Poisson point process [Grimmett and Stirzaker, 2001].

Building SPSNs. An SPSN is constructed from the schema of the data graph (Definition 2), i.e., an SPSN is a (generally non-isomorphic) mapping from the schema (graph) to the computational graph. The heterogeneous nodes, $v \in H$, are modeled by possibly many alternating layers of sum units, $n \in \mathsf{S}$, and product units, $n \in \mathsf{P}$. We can use the product unit in these alternations as long as $T_v \coloneqq \{T_{u_1}, \ldots, T_{u_m}\}$ is either factored into user-specified heterogeneous subsets or completely reduced to unique singletons. Every time this factorization yields a homogeneous subset, only the set unit, $n \in \mathsf{B}$, can be used to process it, as follows from Assumption 1. The leaf nodes, $v \in L$, rely on the sum units and product units to span computational subgraphs over plain, fixed-size feature vectors, $T_v = \mathbf{x}_v$, as in the conventional SPNs, i.e., they do not model random-size subgraphs but fixed-size vectors. See Figure 1, for an example.

Hyper-parameters. The key hyper-parameter of SPSNs is the depth of the computational subgraph modeling each heterogeneous node of the data graph, n_l. It says how many times we alter the sum units with the product units at each heterogeneous node. The other hyper-parameters are the number of children of all sum units, n_s, and product units, n_p. We use these hyper-parameters uniformly in the whole network ($n_l = 1$, $n_s = 2$, and $n_p = 2$ in Figure 1).

Structural constraints. The SPSNs are amenable to the standard structural constraints used in PCs [Shen et al., 2016]. We use the following two constraints on $\mathcal{G}$ to ensure algebraic tractability. *Smoothness*: children of any sum unit have the same scope, i.e., each $n \in \mathsf{S}$ satisfies $\forall u, v \in \mathbf{ch}(n) : \psi(u) = \psi(v)$. *Decomposability*: children of any product unit have a disjoint scope, i.e., each $n \in \mathsf{P}$ satisfies $\forall u, v \in \mathbf{ch}(n) : \psi(u) \cap \psi(v) = \varnothing$. The set unit does not violate these constraints. The cardinality distribution and the feature density are computational subgraphs given by the SPSN units (Definition 3), and the integration propagates through them in the usual way.

4 EXPERIMENTS

We illustrate the performance and properties of the algebraically tractable SPSN models compared to various in-

Table 1: *Graph classification.* The test accuracy (higher is better) for the MLP, GRU, LSTM, HMIL, and SPSN networks. It is displayed for the best model in the grid search, which was selected based on the validation accuracy. The results are averaged over 5 runs with different initial conditions. The accuracy is shown with its standard deviation. The average rank is computed as the standard competition ("1224") ranking [Demšar, 2006] on each dataset (lower is better).

dataset	MLP	GRU	LSTM	HMIL	SPSN
chess	**0.41±0.03**	**0.41±0.05**	0.34±0.04	0.39±0.02	0.39±0.03
citeseer	0.69±0.02	0.74±0.01	0.74±0.02	**0.75±0.01**	**0.75±0.01**
cora	0.75±0.03	**0.86±0.01**	0.84±0.01	0.85±0.00	**0.86±0.01**
craftbeer	0.25±0.03	0.25±0.03	0.25±0.03	**0.29±0.02**	0.07±0.01
genes	0.99±0.01	**1.00±0.01**	0.98±0.01	**1.00±0.01**	0.95±0.01
hepatitis	0.86±0.02	**0.88±0.01**	0.87±0.03	**0.88±0.02**	**0.88±0.02**
mutagenesis	**0.84±0.02**	0.83±0.02	0.82±0.04	0.83±0.00	**0.84±0.02**
uwcse	0.84±0.02	**0.87±0.03**	0.85±0.02	0.86±0.03	0.84±0.02
webkp	0.77±0.02	**0.82±0.01**	0.81±0.02	**0.82±0.01**	0.81±0.02
world	0.68±0.05	0.51±0.16	0.57±0.08	0.72±0.05	**0.94±0.03**
rank	3.40	1.90	3.70	**1.80**	2.50

tractable, NN-based models. In this context, we would like to investigate their performance in the discriminative learning regime and their robustness to missing values. We provide the implementation of SPSNs at `https://github.com/aicenter/SumProductSet.jl`.

Models. To establish the baseline with the intractable models, we choose variants of recurrent NNs (RNNs) for tree-structured data. Though these models are typically used in the NLP domain (Section 1), they are, too, directly applicable to the tree-structured data in Definition 1. These tree-RNNs differ in the type of cell. We consider the simple multi-layer perceptron (MLP) cell, the gated recurrent unit (GRU) cell [Zhou et al., 2016], and the long-short term memory (LSTM) cell [Tai et al., 2015]. The key assumption of these models is that they consider each leaf data node to have the same dimension. This requirement does not hold in Definition 1. Therefore, for all these NN-based models, we add a single dense layer with the linear activation function in front of each leaf node, $v \in L$, to make the input dimension the same. As another competitor, we use the hierarchical multiple-instance learning (HMIL) network [Pevný and Somol, 2016], which is also specifically tailored for the tree-structured data.

Settings. We convert ten publicly available datasets from the CTU relational repository [Motl and Schulte, 2015] into the JSON format [Pezoa et al., 2016]. The dictionary nodes, list nodes, and atomic nodes of the JSON format directly correspond to the heterogeneous nodes, homogeneous nodes, and leaf nodes of the tree-structured data, respectively (Definition 1, Figure 1). We present the rest of the experimental settings in the supplementary material, including specific examples of the JSON data. All models and experiments are implemented in Julia, using `JSONGrinder.jl` and `Mill.jl` [Mandlík et al., 2022].

Graph classification. Table 1 shows the test accuracy of classifying the tree-structured graphs. The HMIL and GRU networks deliver the best performance, with the SPSN network falling slightly behind. If we look closely at the individual lines, we can see that the SPSN network is often very similar to (or the same as) the HMIL and GRU networks, e.g., it is substantially better on the `world` dataset. We consider these results unexpectedly good, given that the (NN-based) MLP, GRU, LSTM, and HMIL architectures are denser than the sparse SPSN architecture.

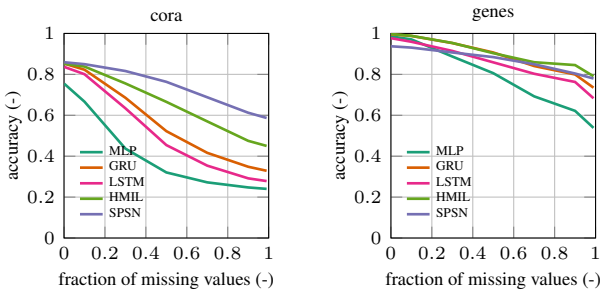


Figure 2: *Missing values.* The test accuracy (higher is better) versus the fraction of missing values for the MLP, GRU, LSTM, HMIL, and SPSN networks. It is displayed for the best model in the grid search, which was selected based on the validation accuracy. The results are averaged over five runs with different initial conditions.

Missing values. We consider an experiment where we select the best model in the grid search based on the validation data (as in Table 1) and evaluate its accuracy on the test data containing a fraction of randomly-placed missing values. Figure 2 demonstrates that the SPSN either outperforms or is similar to the NNs. We show the results only for the `cora` and `genes` datasets, but similar observations hold for the remaining datasets (see the supplementary material).

5 CONCLUSION

We have utilized the theory of finite random sets to synthesize a new class of deep learning models—sum-product-set networks (SPSNs)— representing a probability density over tree-structured graphs. Algebraic tractability is the essential benefit of SPSNs, yet it is redeemed by making them sparser than the algebraically intractable NNs. Notwithstanding this, SPSNs deliver a very competitive performance to the NNs in the graph classification task. We have demonstrated that the tractable and simple inference of SPSNs has also allowed us to achieve comparable results to the NNs regarding the robustness to missing values.

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

Oren Ben-Kiki, Clark Evans, and Brian Ingerson. YAML ain't markup language (YAML™) version 1.1. *Working Draft 2008*, 5:11, 2009.

Anna Maria Bianucci, Alessio Micheli, Alessandro Sperduti, and Antonina Starita. Application of cascade correlation networks for structures to chemistry. *Applied Intelligence*, 12:117–147, 2000.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Zhou Cheng, Chun Yuan, Jiancheng Li, and Haiqin Yang. TreeNet: Learning sentence representations with unconstrained tree structure. In *IJCAI*, pages 4005–4011, 2018.

Daryl J Daley, David Vere-Jones, et al. *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.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

Geoffrey Grimmett and David Stirzaker. *Probability and random processes*. Oxford university press, 2001.

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.

James Kahn, Ilias Tsaklidis, Oskar Taubert, Lea Reuter, Giulio Dujany, Tobias Boeckh, Arthur Thaller, Pablo Goldenzweig, Florian Bernlochner, Achim Streit, et al. Learning tree structures from leaves for particle decay reconstruction. *Machine Learning: Science and Technology*, 3(3):035012, 2022.

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.

Jing Ma, Wei Gao, and Kam-Fai Wong. Rumor detection on twitter with tree-structured recursive neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1980–1989, 2018.

Ronald PS Mahler. *Statistical multisource-multitarget information fusion*, volume 685. Artech House Norwood, MA, USA, 2007.

Šimon Mandlík, Matěj Račinský, Viliam Lisý, and Tomáš Pevný. Jsongrinder.jl: automated differentiable neural architecture for embedding arbitrary JSON data. *Journal of Machine Learning Research*, 23(298):1–5, 2022.

Jan Motl and Oliver Schulte. The CTU Prague relational learning repository. *arXiv preprint arXiv:1511.03086*, 2015.

Hung T Nguyen. *An introduction to random sets*. Chapman and Hall/CRC, 2006.

Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On theoretical properties of sum-product networks. In *18th International Conference on Artificial Intelligence and Statistics*, pages 744–752. PMLR, 2015.

Tomáš Pevný and Petr Somol. Discriminative models for multi-instance problems with tree structure. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, pages 83–91, 2016.

Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of JSON schema. In *Proceedings of the 25th international conference on World Wide Web*, pages 263–273, 2016.

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.

Yujia Shen, Arthur Choi, and Adnan Darwiche. Tractable operations for arithmetic circuits of probabilistic models. *Advances in Neural Information Processing Systems*, 29, 2016.

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.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, 2015.

Joe Tekli, Nathalie Charbel, and Richard Chbeir. Building semantic trees from XML documents. *Journal of Web Semantics*, 37:1–24, 2016.

MNM Van Lieshout. *Markov point processes and their applications*. World Scientific, 2000.

Antonio Vergari, YooJung Choi, Robert Peharz, and Guy Van den Broeck. Probabilistic circuits: Representations, inference, learning and applications. In *Tutorial at the The 34th AAAI Conference on Artificial Intelligence*, 2020.

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.

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, 2021.

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.

Yao Zhou, Cong Liu, and Yan Pan. Modelling sentence pairs with tree-structured attentive encoder. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2912–2922, 2016.

Sum-Product-Set Networks (Supplementary Material)

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

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

A PROBABILISTIC CIRCUITS

A probabilistic circuit (PC) is a deep learning model representing a joint probability density, $p(\mathbf{x})$, over a fixed-size, *unstructured*, random variable, $\mathbf{x} = (x_1, \dots, x_d) \in \mathsf{X} \subset \mathbb{R}^d$. The key feature of a PC is that—under certain regularity assumptions—it permits exact and efficient inference scenarios. We define a PC by a parameterized computational graph, $\mathcal{G}$, and a scope function, ψ.

Definition 5. *(Computational graph). $\mathcal{G} \coloneqq (\mathcal{V}, \mathcal{E}, \theta)$ is a parameterized, directed, acyclic graph, where $\mathcal{V}$ is a set of vertices, $\mathcal{E}$ is set of edges, and $\theta \in \Theta$ are parameters. $\mathcal{V} \coloneqq (\mathsf{S}, \mathsf{P}, \mathsf{L})$ contains three different subsets of computational units: sum units, S, product units, P, and leaf units, L. Let $\mathbf{ch}(n)$ and $\mathbf{pa}(n)$ denote the set of child and parent units of $n \in \mathcal{V}$, respectively. If $\mathbf{pa}(n) = \varnothing$, then n is the root unit. If $\mathbf{ch}(n) = \varnothing$, then n is a leaf unit. We consider that $\mathcal{V}$ contains only a single root unit, and each product unit has only a single parent. The parameters $\theta \coloneqq \{\theta_s, \theta_l\}$ are divided into (i) parameters of all sum units, $\theta_n = \{w_{n,c}\}_{c \in \mathbf{ch}(n)}$, which contain non-negative and* locally *normalized weights [Peharz et al., 2015], $w_{n,c} \geq 0$, $\sum_{c \in \mathbf{ch}(n)} w_{n,c} = 1$; and (ii) parameters of all leaf units, θ_l, which are specific to a given family of densities, with possibly a different density for each $n \in \mathsf{L}$.*

Definition 6. *(Scope function). The mapping $\psi : \mathcal{V} \to 2^{\mathbf{x}}$—from the set of units to the power set of $\mathbf{x}$—outputs a subset of $\mathbf{x} \in \mathsf{X}$ for each $n \in \mathcal{V}$ and is referred to as the* scope function. *If n is the root unit, then $\psi(n) = \mathbf{x}$. If n is a sum unit or a product unit, then $\psi(n) = \bigcup_{c \in \mathbf{ch}(n)} \psi(c)$.*

PCs are an instance of neural networks [Vergari et al., 2019, Peharz et al., 2020], where each computational unit is a probability density characterized by certain functionality. Leaf units are the input of a PC. For each $n \in \mathsf{L}$, they compute a (user-specified) probability density, $p_n(\cdot)$, over a subset of $\mathbf{x}$ given by the scope, $\psi(n)$, which can be univariate or multivariate [Peharz et al., 2015]. Sum units are mixture densities that compute the weighted sum over its children, $p_n(\cdot) = \sum_{c \in \mathbf{ch}(n)} w_{n,c} p_c(\cdot)$, where $w_{n,c}$ (Definition 5) weights the connection between the sum unit and a child unit. Product units are factored densities that compute the product of its children, $p_n(\psi(n)) = \prod_{c \in \mathbf{ch}(n)} p_c(\psi(c))$, establishing an unweighted connection between n and c and introducing the conditional independence among the scopes of its children, $\psi(c)$. It is commonly the case that (layers of) sum units interleave (layers of) product units. The computations then proceed recursively through $\mathcal{G}$ until reaching the root unit—the output of a PC.

PCs are generally intractable. They instantiate themselves into specific circuits—and thus permit tractability of specific inference scenarios—by imposing various constraints on $\mathcal{G}$, examples include smoothness, decomposability, structured decomposability, determinism, consistency [Chan and Darwiche, 2006, Poon and Domingos, 2011, Shen et al., 2016]. In this work, we use only the first two of these constraints.

Definition 7. *(Structural constraints). We restrict ourselves to PCs that respect the following constraints on $\mathcal{G}$. **Smoothness**: children of any sum unit have the same scope, i.e., each $n \in \mathsf{S}$ satisfies $\forall u, v \in \mathbf{ch}(n) : \psi(u) = \psi(v)$. **Decomposability**: children of any product unit have a disjoint scope, i.e., each $n \in \mathsf{P}$ satisfies $\forall u, v \in \mathbf{ch}(n) : \psi(u) \cap \psi(v) = \varnothing$.*

A PC satisfying Definition 7 can be seen as a polynomial composed of leaf units [Darwiche, 2003]. This construction guarantees that any single-dimensional integral interchanges with a sum unit and impacts only a single child of a product

Accepted for the 6th Workshop on Tractable Probabilistic Modeling at UAI (TPM 2023).

unit [Peharz et al., 2015]. The integration is then propagated down to the leaf units, where it can be computed under a closed-form solution (for a tractable form of $n \in \mathsf{L}$). The key practical consequence lies in that various inference tasks—such as integrals of $p(\mathbf{x})$ over $\{x_a, \ldots, x_b\} \subset \mathbf{x}$—are *tractable* and can be computed in time which is linear in the circuit size (i.e., the cardinality of $\mathcal{V}$). $p(\mathbf{x})$ is guaranteed to be normalized if Definition 7 holds and leaf units are from the exponential family [Barndorff-Nielsen, 1978]. PCs that fulfill Definition 7 are commonly referred to as sum-product networks.

B RELATED WORK

Non-probabilistic models (NPMs). Graph neural networks (GNNs) have become a flexible and powerful approach for (non-probabilistic) representation learning on graph-structured data. Variants of GNNs range from the original formulation [Gori et al., 2005, Scarselli et al., 2008] to GCN [Kipf and Welling, 2017], MPNN [Gilmer et al., 2017], GAT [Veličković et al., 2018] and GraphSAGE [Hamilton et al., 2017], among others. They assume input data as an undirected cyclic graph, which they encode into a low-dimensional representation by aggregating and sharing features from neighboring nodes. However, without necessary adaptations, their structure-agnostic character incurs unnecessary computational costs when applied to graphs with structural constraints. This fact has led to the design of GNNs that respect the constraints in the form of directed acyclic graphs (DAGs) [Thost and Chen, 2021]. GNNs for trees (i.e., a specific case of DAGs) create the encoding by traversing nodes of the graph bottom-up (or up-bottom) and updating the state representation of a given node based only on its children. Examples of this approach include RNN [Socher et al., 2011, Shuai et al., 2016], Tree-LSTM [Tai et al., 2015] and TreeNet [Cheng et al., 2018].

Intractable probabilistic models (IPMs). Extending deep generative models from unstructured to graph-structured domains has recently gained significant attention. Variational autoencoders learn a probability distribution over graphs, $p(G)$, by training an encoder and a decoder to map between space of graphs and continuous latent space, minimizing the evidence lower bound on the marginal log-likelihood in the process [Kipf and Welling, 2016, Simonovsky and Komodakis, 2018, Grover et al., 2019]. Generative adversarial networks learn $p(G)$ by training (i) a generator to map from latent space to space of graphs and (ii) a discriminator to distinguish whether the graphs are synthetic or real, relying on the two-player, minimax objective [De Cao and Kipf, 2018, Bojchevski et al., 2018]. Flow models use the change of variables formula to transform a base distribution on latent space to a distribution on space of graphs, $p(G)$, via an invertible mapping, using direct optimization of the marginal log-likelihood [Liu et al., 2019, Luo et al., 2021]. Autoregressive models learn $p(G)$ by relying on the chain rule of probability to decompose a graph, G, into a sequence of subgraphs and constructing G node by node [You et al., 2018, Liao et al., 2019]. Diffusion models learn $p(G)$ by noising and denoising trajectories of graphs based on forward and backward diffusion processes, respectively, optimizing the score matching objective or evidence lower bound on marginal log-likelihood [Jo et al., 2022, Huang et al., 2022, Vignac et al., 2022]. GNNs mentioned in the previous paragraph (NMPs) have been used as the building blocks of all these generative models, which is the main reason their marginal probability density is intractable.

Tractable probabilistic models (TPMs). There has not been a substantial interest in probabilistic models facilitating tractable inference for graph-structured data. Graph-structured SPNs [Zheng et al., 2018] heuristically decompose generic cyclic graphs into components that are isomorphic to a pre-specified set of sub-graph templates (of an arbitrary acyclic structure) and then design the conventional SPNs for each of the templates. The roots of these SPNs are aggregated by the sum unit and a layer of the product units. Graph-induced SPNs [Errica and Niepert, 2023] are similar to this approach. They also decompose generic cyclic graphs, but in a more principled manner, building a set of trees based on a user-specified neighborhood. The SPNs are not designed for the whole trees (templates) but for their nodes, only to model the feature vectors. The aggregation is performed by using the posterior probabilities of the root sum units at lower depths of the tree to condition the sum units at upper depths. Relational SPNs (RSPNs) [Nath and Domingos, 2015] are tractable probabilistic models for relational data (a particular form of graph-structured data). The semantics of the relational data differs from our graph-structured data. However, the set unit of the SPSNs is similar to the exchangeable distribution template of the RSPNs. The main difference to the RSPNs lies in that SPSNs model cardinality. The mixture of probability densities over finite random sets is perhaps the most related approach to SPSNs [Phung and Vo, 2014, Tran et al., 2016, Vo et al., 2018]. In our context, it can be seen as the sum unit with its children given by the set units. However, these shallow probabilistic models are designed for plain point pattern data. SPSNs generalize them to deep probabilistic models for graph-structured data, achieving far higher expressivity by stacking the sum units with the product units and reflecting the variable size nature of the graph by incorporating a hierarchy of the set units. Another approach relies on SPNs to introduce correlations into graph variational autoencoder [Xia et al., 2023], which does not allow for tractable probabilistic inference.

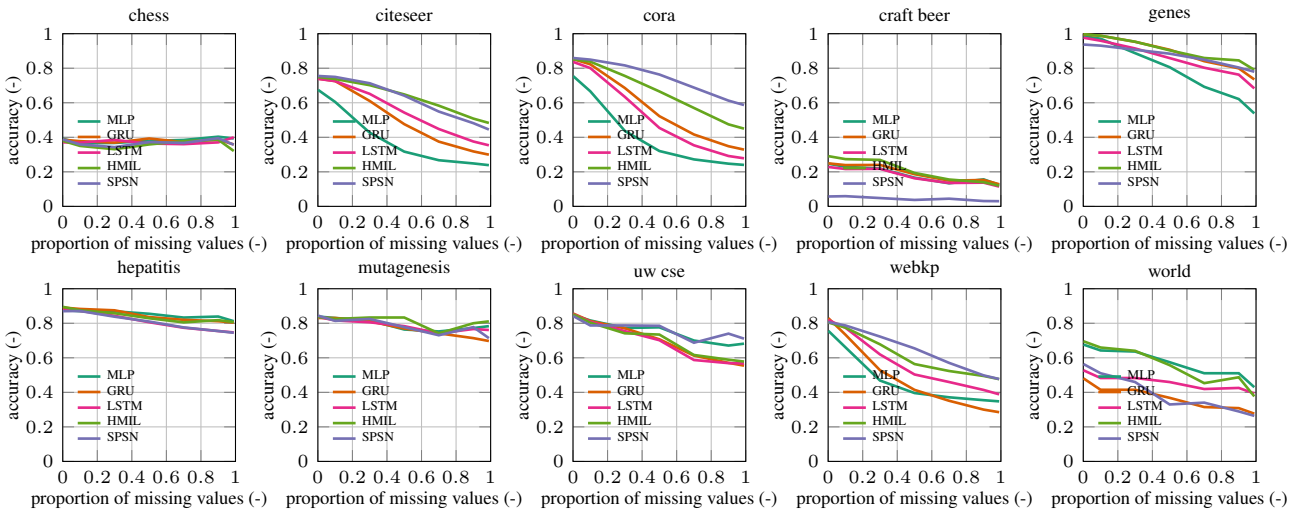


Figure 3: *Missing values.* The test accuracy (higher is better) versus the fraction of missing values for the MLP, GRU, LSTM, HMIL, and SPSN networks. It is displayed for the best model in the grid search, which was selected based on the validation accuracy. The results are averaged over five runs with different initial conditions.

C EXPERIMENTAL SETTINGS

The leaf nodes, $v \in L$, contain different data types, including reals, integers, and strings. We use the default feature extractor from `JSONGrinder.jl` (v2.3.2) to pre-process these data. We perform the grid search over the hyper-parameters of the models mentioned in Section 4. For the MLP, GRU, and LSTM networks, we set the dimension of the hidden state(s) and the output in $\{10, 20, 30, 40\}$. For the HMIL network, we use the default settings of the model builder from `Mill.jl` (v2.8.1), only changing the number of hidden units of all the inner layers in $\{10, 20, 30, 40\}$. We add a single dense layer with the linear activation function to adapt the outputs of these networks to the number of classes in the datasets. For the SPSN networks, we choose the Poisson distribution as the cardinality distribution and the following hyper-parameters: $n_l \in \{1, 2, 3\}$, $n_s \in \{2, 3, \ldots, 10\}$, and $n_p \coloneqq 2$. We use the ADAM optimizer [Kingma and Ba, 2014] with fixing 10 samples in the minibatch and varying the step-size in $\{0.1, 0.01, 0.001\}$. The datasets are randomly split into 64%, 16%, and 20% for training, validation, and testing, respectively.

D ADDITIONAL RESULTS

Figure 3 complements Figure 2 of the main paper. It contains the remaining experiments with the missing values, as mentioned in Section 4. We can observe that the SPSNs provide slightly better or very similar results to the NNs, except for the `craftbeer` and `world` datasets.

E DATASETS

Table 2 shows the two main attributes of the datasets under study [Motl and Schulte, 2015]: the number of instances (i.e., the number of tree-structured graphs) and the number of different classes of these instances. A detailed description of these datasets, additional attributes, and accompanying references are accessible at `https://relational.fit.cvut.cz/`.

Figure 4 shows a single instance of the tree-structured graph data in the JSON format [Pezoa et al., 2016], and Figure 5 illustrates the corresponding schema (Definition 2 of the main paper). As can be seen (and as also mentioned in the main paper), the leaf nodes contain different data types: integers, floats, and strings. In Figures 6-14, we provide the schemata of the remaining datasets in Table 2.

Table 2: *Datasets.* The number of instances (trees) and the number of classes of the datasets under study.

dataset	# of instances (trees)	# of classes
mutagenesis	188	2
genes	862	15
cora	2708	7
citeseer	3312	6
webkp	877	5
world	239	7
craftbeer	558	51
chess	295	3
uwcse	278	4
hepatitis	500	2

References

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

Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. NetGAN: Generating graphs via random walks. In *International conference on machine learning*, pages 610–619. PMLR, 2018.

Hei Chan and Adnan Darwiche. On the robustness of most probable explanations. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*, pages 63–71, 2006.

Zhou Cheng, Chun Yuan, Jiancheng Li, and Haiqin Yang. TreeNet: Learning sentence representations with unconstrained tree structure. In *IJCAI*, pages 4005–4011, 2018.

Adnan Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM (JACM)*, 50(3):280–305, 2003.

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

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

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

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs. In *International conference on machine learning*, pages 2434–2444. PMLR, 2019.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. 2017.

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*, 2022.

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.

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

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations (ICLR)*, 2017.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, William L Hamilton, David Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 4255–4265, 2019.

Jenny Liu, Aviral Kumar, Jimmy Ba, Jamie Kiros, and Kevin Swersky. Graph normalizing flows. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 13578–13588, 2019.

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.

Jan Motl and Oliver Schulte. The CTU Prague relational learning repository. *arXiv preprint arXiv:1511.03086*, 2015.

Aniruddh Nath and Pedro Domingos. Learning relational sum-product networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On theoretical properties of sum-product networks. In *18th International Conference on Artificial Intelligence and Statistics*, pages 744–752. PMLR, 2015.

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, 2020.

Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of JSON schema. In *Proceedings of the 25th international conference on World Wide Web*, pages 263–273, 2016.

Dinh Phung and Ba-Ngu Vo. A random finite set model for data clustering. In *17th International Conference on Information Fusion (FUSION)*, pages 1–8. IEEE, 2014.

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.

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.

Yujia Shen, Arthur Choi, and Adnan Darwiche. Tractable operations for arithmetic circuits of probabilistic models. *Advances in Neural Information Processing Systems*, 29, 2016.

Bing Shuai, Zhen Zuo, Bing Wang, and Gang Wang. DAG-recurrent neural networks for scene labeling. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3620–3629, 2016.

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.

Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, 2015.

Veronika Thost and Jie Chen. Directed acyclic graph neural networks. In *9th International Conference on Learning Representations (ICLR)*, 2021.

Nhat-Quang Tran, Ba-Ngu Vo, Dinh Phung, and Ba-Tuong Vo. Clustering for point pattern data. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 3174–3179. IEEE, 2016.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.

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

Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. DiGress: Discrete denoising diffusion for graph generation. *arXiv preprint arXiv:2209.14734*, 2022.

Ba-Ngu Vo, Nhan Dam, Dinh Phung, Quang N Tran, and Ba-Tuong Vo. Model-based learning for point pattern data. *Pattern Recognition*, 84:136–151, 2018.

Riting Xia, Yan Zhang, Chunxu Zhang, Xueyan Liu, and Bo Yang. Multi-head variational graph autoencoder constrained by sum-product networks. In *Proceedings of the ACM Web Conference 2023*, pages 641–650, 2023.

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.

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.


```
{
    "ind1": 1,
    "lumo": -1.246,
    "inda": 0,
    "logp": 4.23,
   "atoms": [
        {
          "element": "c",
            "bonds": [
                { "element": "c", "charge": -0.117, "type_bond": 7, "type_atom": 22 },
                { "element": "h", "charge":  0.142, "type_bond": 1, "type_atom":  3 },
                { "element": "c", "charge": -0.117, "type_bond": 7, "type_atom": 22 }
            ],
           "charge": -0.117,
        "type_atom": 22
        },
        :
        {
          "element": "h",
            "bonds": [
                { "element": "c", "charge": -0.117, "type_bond": 1, "type_atom": 22 }
            ],
           "charge": 0.142,
        "type_atom": 3
        }
    ]
}
```


Figure 4: *Instance.* A single instance of the tree-structured graph in the JSON format, taken from the `mutagenesis` dataset.

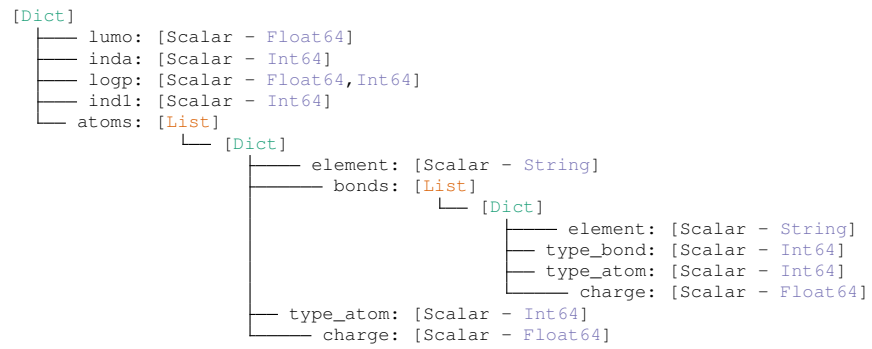


Figure 5: *Schema.* The schema (Definition 2 of the main paper) of the tree-structured graph in Figure 4 (the `mutagenesis` dataset).

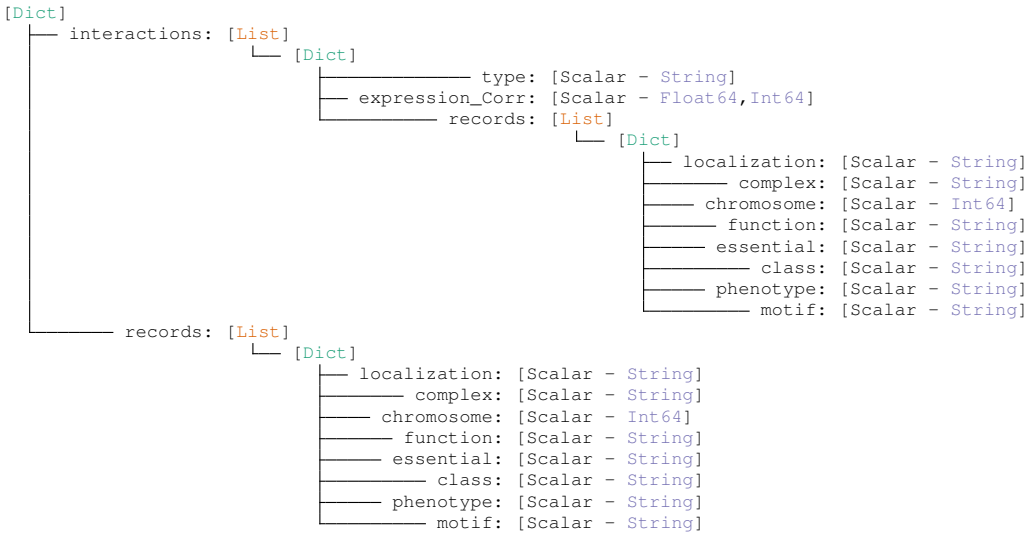


Figure 6: The schema of the `genes` dataset.

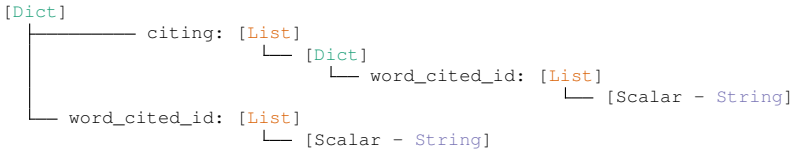


Figure 7: The schema of the `cora` dataset.

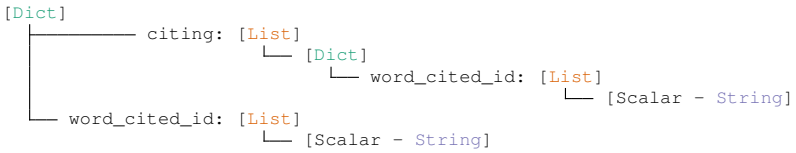


Figure 8: The schema of the `citeseer` dataset.

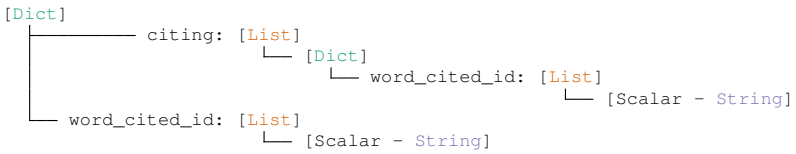


Figure 9: The schema of the `webkp` dataset.

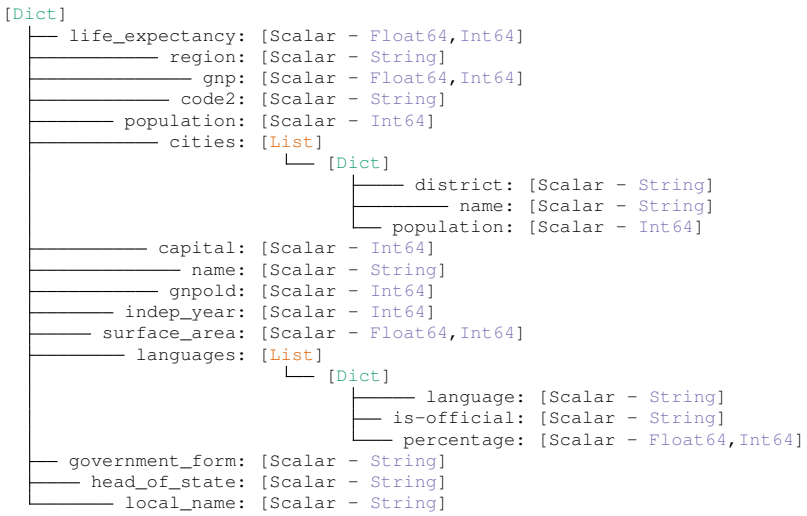


Figure 10: The schema of the `world` dataset.

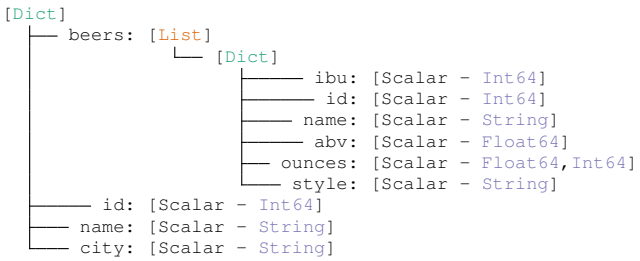


Figure 11: The schema of the `craft_beer` dataset.

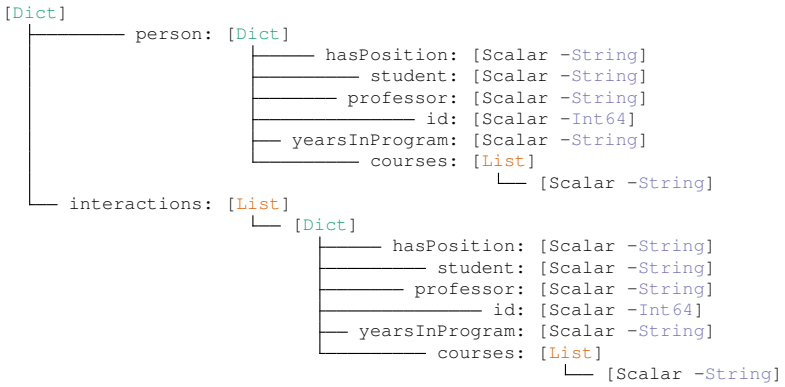


Figure 12: The schema of the the `uw_cse` dataset.

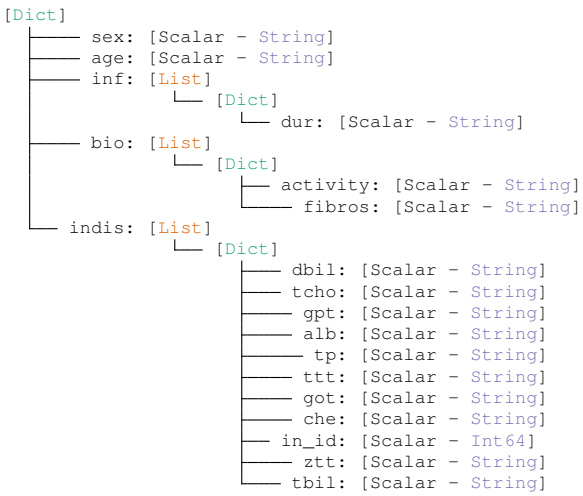


Figure 13: The schema of the `hepatitis` dataset.

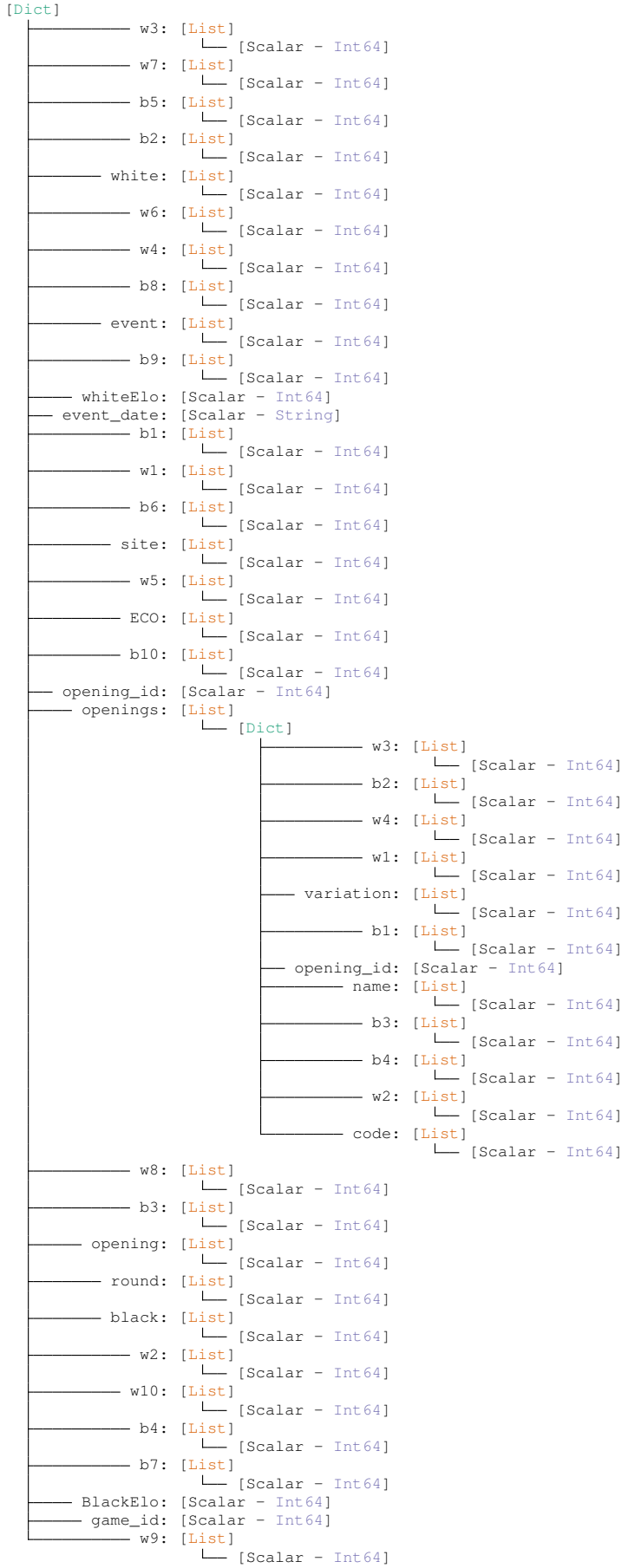


Figure 14: The schema of the `chess` dataset.