Towards Coreset Learning in Probabilistic Circuits

Martin Trapp[1] **Steven Lang**[2] **Aastha Shah**[*3] **Martin Mundt**[2,4] **Kristian Kersting**[2,4] **Arno Solin**[1]

[1]Aalto University, Espoo, Finland
[2]TU Darmstadt, Darmstadt, Germany
[3]University of Pennsylvania, Philadelphia, USA
[4] Hessian Center for AI (hessian.AI), Darmstadt, Germany

Abstract

Probabilistic circuits (PCs) are a powerful family of tractable probabilistic models, guaranteeing efficient and exact computation of many probabilistic inference queries. However, their sparsely structured nature makes computations on large data sets challenging to perform. Recent works have focused on tensorized representations of PCs to speed up computations on large data sets. In this work, we present an orthogonal approach by sparsifying the set of n observations and show that finding a coreset of $k \ll n$ data points can be phrased as a monotone submodular optimisation problem which can be solved greedily for a deterministic PCs of $|\mathcal{G}|$ nodes in $\mathcal{O}(k\,n\,|\mathcal{G}|)$. Finally, we verify on a series of data sets that our greedy algorithm outperforms random selection.

1 INTRODUCTION & RELATED WORK

Over the years, probabilistic circuits (PCs) (*e.g.*, Darwiche [2003], Poon and Domingos [2011], Peharz [2015], Trapp [2020], Choi et al. [2020], Vergari et al. [2021]) have been established as a powerful family of deep probabilistic models rendering many probabilistic inferences tractable. However, scaling computations in PCs to large amounts of data is a challenging and unsolved task. To overcome the computational challenges arising in PCs, prior works have primarily focused on restricting the model architecture in order to obtain tensorized representations of PCs (*e.g.*, Peharz et al. [2020b] and Peharz et al. [2020a]). In this work, we present an orthogonal approach by studying coreset selection in PCs, *i.e.*, summarizing the data set with a coreset.

Coresets have been widely studied in machine learning, with applications to summarization or active learning. In particular, there exists a plethora of approaches to coreset selection for classical modelling families (*e.g.*, Tsang et al. [2005], Har-Peled and Kushal [2007], Rosman et al. [2014], Wei et al. [2015], Tremblay et al. [2019]) in the context of supervised and unsupervised learning problems. Recently, coreset selection has also been explored in the context of Bayesian learning to accelerate approximate inference (*e.g.*, Campbell and Broderick [2019], Zhang et al. [2021]) and in the context of deep architectures such as neural networks (*e.g.*, Borsos et al. [2020], Killamsetty et al. [2021]). We refer to Munteanu and Schwiegelshohn [2018] for a more detailed review of coreset methods. However, to the best of our knowledge, coreset selection has not been considered in the context of probabilistic circuits.

In this work, we study coreset selection in PCs, *i.e.*, summarising a data set of size n with a coreset of size $k \ll n$, and show that deterministic PCs with indicator leaves admit efficient greedy coreset selection. In particular, we show that the optimisation problem for deterministic PCs can be phrased as a cardinality constrained monotone submodular maximisation problem [Fujishige, 2005], which can be solved for a circuit of $|\mathcal{G}|$ nodes in $\mathcal{O}(k\,n\,|\mathcal{G}|)$ or in $\mathcal{O}(k\,\log(n)\,\log(|\mathcal{G}|))$ for a parallelised implementation.

The contributions of this work are summarised as follows:

- We discuss coreset selection in deterministic PCs with indicators and show that the task can be phrased as cardinality constrained monotone submodular problem.
- Moreover, we introduce an efficient greedy algorithm for approximate coreset selection.
- Lastly, we assess the efficiency of our greedy algorithm on a series of standard benchmark data sets.

2 PRELIMINARIES

We briefly review background information on coresets and probabilistic circuits, and introduce the notation used in this work. See appendix for a notation summary.

[*]Work done as an intern at Aalto University.

Accepted for the 5th Workshop on Tractable Probabilistic Modeling at UAI (TPM 2022).

2.1 CORESETS

Given a set $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^n$ of n data points, we aim to summarize $\mathcal{X}$ through a sparsely weighted set $\mathcal{C} = \boldsymbol{\gamma}\mathcal{X}$ called a coreset. More formally, let $\boldsymbol{\gamma} \in \mathbb{R}_+^n$ be a sparse vector of positive weights such that $k \ll n$ and $k \geq ||\boldsymbol{\gamma}||_0$, where $||\cdot||_0$ denotes the ℓ_0 pseudo-norm. Moreover, let $\theta \in \Theta$ denote a solution to a cost function, which we assume to additively decompose into non-negative functions. An example for such a cost function is the negative log-likelihood (NLL) function defined as:

$$\mathcal{L}_\theta(\mathcal{X}) = \sum_{i=1}^{n} -f(\boldsymbol{x}_i \mid \theta), \tag{1}$$

where $f(\boldsymbol{x} \mid \theta) \leq 0$ is a log-probability function. Further, let the coreset weighted NLL function be given by:

$$\mathcal{L}_\theta(\mathcal{C}) = \sum_{i=1}^{n} -\boldsymbol{\gamma}_i f(\boldsymbol{x}_i \mid \theta), \tag{2}$$

we aim to find a coreset $\mathcal{C}$ of size k such that the difference between the costs is minimal.

Definition 1 (Coreset)**.** *Let $k \geq 1, \varepsilon \in (0,1)$. The $\boldsymbol{\gamma}$-weighted set $\mathcal{C}$ is a ε-coreset for $\mathcal{L}$, if for every $\theta \in \Theta$:*

$$|\mathcal{L}_\theta(\mathcal{X}) - \mathcal{L}_\theta(\mathcal{C})| \leq \varepsilon \mathcal{L}_\theta(\mathcal{X}) \tag{3}$$

$\mathcal{C}$ is a *strong* coreset if Eq. (3) holds uniformly for all $\theta \in \Theta$, and a *weak* coreset if Eq. (3) only holds for the optimal solution, *i.e.* the maximum likelihood estimate (MLE). We refer to Bachem et al. [2017] for further details.

2.2 PROBABILISTIC CIRCUITS

There are multiple ways to define probabilistic circuits (PCs), for consistency with recent works, we will use the formalism in Trapp et al. [2019] and Choi et al. [2020]. A PC on a set of RVs $\boldsymbol{X} = \{X_j\}_{j=1}^d$ is defined as tuple $(\mathcal{G}, \psi)$ consisting of a directed acyclic graph $\mathcal{G}$ and a scope function ψ. Nodes in $\mathcal{G}$ are either sum nodes $(\mathrm{S}(\boldsymbol{x}) = \sum_{\mathrm{N}\in\mathrm{ch}(\mathrm{S})} \theta_{\mathrm{S,N}}\, \mathrm{N}(\boldsymbol{x}))$, product nodes $(\mathrm{P}(\boldsymbol{x}) = \prod_{\mathrm{N}\in\mathrm{ch}(\mathrm{P})} \mathrm{N}(\boldsymbol{x}))$ or leaf nodes $(\mathrm{L}(\boldsymbol{x}) = p(\boldsymbol{x} \mid \theta_{\mathrm{L}}))$ nodes. We use N to denote a generic node and use boldface to indicate sets of nodes. The scope function $\psi\colon \mathbf{N} \to \mathscr{P}(\boldsymbol{X})$ assigning each node $\mathrm{N} \in \mathbf{N}$ in $\mathcal{G}$ a scope, *i.e.* a subset $\boldsymbol{Y} \subseteq \boldsymbol{X}$, where $\mathscr{P}(\boldsymbol{X})$ is the power set including $\emptyset$ and $\boldsymbol{X}$.

Depending on the structural properties of the circuit, specific probabilistic inference queries can be answered tractably. We will assume throughout the paper that the circuit is *smooth* and *decomposable* (see Choi et al. [2020] for details). Moreover, we will focus most of our analysis on circuits that are *deterministic*.

Definition 2 (Determinism)**.** *A sum node is* deterministic *if, for any fully-instantiated input, the output of at most one of its children is nonzero. A PC is* deterministic *if all of its sum nodes are* deterministic*.*

Even though *determinism* is a strong structural constraint, it has been shown by prior work that ensembles thereof can obtain competitive results in density estimation tasks (*e.g.*, Peharz et al. [2014], Liang et al. [2017]) and can be effective surrogate models for Bayesian inference (*e.g.*, Belle et al. [2015], Shih and Ermon [2020]).

3 MAIN RESULTS

The following section will study coreset estimation under the modelling assumption of deterministic PCs. For simplicity, we will use DC to denote a deterministic PC.

Given a DC $(\mathcal{G}, \psi)$ with indicator leaves and a data set $\mathcal{X}$, the NLL function admits the following simple form:

$$\mathcal{L}_{\boldsymbol{\theta}}(\mathcal{X}) = \sum_{\mathrm{S}\in\mathcal{G}} \sum_{\mathrm{N}\in\mathrm{ch}(\mathrm{S})} -\log \theta_{\mathrm{S,N}}^{g(\mathrm{S,N})}, \tag{4}$$

where $g(\mathrm{S,N}) = \sum_{i=1}^n \mathbb{1}[(\mathrm{S,N}) \in \mathcal{T}_{\boldsymbol{x}_i}]$ and $\mathcal{T}_{\boldsymbol{x}_i}$ denotes the *induced tree* [Zhao et al., 2016, Trapp et al., 2019] associated with the i^{th} datum.

We aim to summarize $\mathcal{X}$ through a sparsely weighted set $\mathcal{C}$ that minimizes the difference between costs:

$$\begin{aligned} \underset{\boldsymbol{\gamma}\,\in\,\mathbb{R}_+^n}{\arg\min} \quad & \left| \sum_{\mathrm{S}\in\mathcal{G}} \sum_{\mathrm{N}\in\mathrm{ch}(\mathrm{S})} -\Big[\underbrace{\log \theta_{\mathrm{S,N}}^{g(\mathrm{S,N})} - \log \theta_{\mathrm{S,N}}^{g(\mathrm{S,N},\boldsymbol{\gamma})}}_{=\log \theta_{\mathrm{S,N}}^{g(\mathrm{S,N},\mathbf{1}-\boldsymbol{\gamma})}} \Big] \right| \\ \text{s.t.} \quad & ||\boldsymbol{\gamma}||_0 \leq k \end{aligned} \tag{5}$$

where k denotes the coreset size and $g(\mathrm{S,N},\boldsymbol{\gamma}) = \sum_{i=1}^n \gamma_i \mathbb{1}[(\mathrm{S,N}) \in \mathcal{T}_{\boldsymbol{x}_i}]$. Note that each term in Eq. (5) will be positive or zero if $g(\mathrm{S,N},\mathbf{1}-\boldsymbol{\gamma}) \geq 0$ for all $(\mathrm{S,N}) \in \mathcal{G}$. Therefore, we obtain the following alternative constrained program:

$$\begin{aligned} \underset{\boldsymbol{\gamma}\,\in\,\mathbb{R}_+^n}{\arg\min} \quad & J(\boldsymbol{\gamma}) = \sum_{\mathrm{S}\in\mathcal{G}} \sum_{\mathrm{N}\in\mathrm{ch}(\mathrm{S})} -\log \theta_{\mathrm{S,N}}^{g(\mathrm{S,N},\mathbf{1}-\boldsymbol{\gamma})} \\ \text{s.t.} \quad & g(\mathrm{S,N},\mathbf{1}-\boldsymbol{\gamma}) \geq 0, \quad \forall (\mathrm{S,N}) \in \mathcal{G}, \\ & ||\boldsymbol{\gamma}||_0 \leq k \end{aligned} \tag{6}$$

3.1 PRODUCT-FREE DETERMINISTIC CIRCUITS

Definition 3 (Product-free)**.** *A PC is said to be product-free if it is a convex combination of leaf nodes with scope $\psi(\mathrm{L}) = \psi(\mathcal{G})$ for all $\mathrm{L} \in \mathcal{G}$, i.e., $\mathcal{G}$ only contains products with at most one child.*

Note that for product-free DCs, each observation is allocated to precisely one leaf node, *i.e.*, each inducing tree $\mathcal{T}$ is a

single path through $\mathcal{G}$. Therefore, the number of leaves is equivalent to the number of inducing trees $\mathcal{T}$ denoted as τ. We will now show that Eq. (6) simplifies for product-free DCs to optimise a monotone modular function.

Definition 4 (Submodular function [Fujishige, 2005])**.** *A function $f\colon 2^V \to \mathbb{R}$ is submodular if given some ground set N for every $A \subseteq B \subseteq N$ and $C \in N \setminus B$ we have:*

$$f(A \cup C) - f(A) \geq f(B \cup C) - f(B), \tag{7}$$

i.e. marginal gains are decreasing. A function is supermodular if its negative is submodular and modular if the inequality holds with equality.

Assume a product-free DC with indicator leaves and let L_j denote the j^{th} leaf, *i.e.*, the unique leaf of the induced tree $\mathcal{T}_j$. Moreover, let $\pi_j \coloneqq \prod_{(\mathrm{S},\mathrm{L})\in\mathcal{T}_j} \theta_{\mathrm{S},\mathrm{L}}$ denote the weight of the j^{th} induced tree. Then the NLL function is given as:

$$\mathcal{L}_\pi(\mathcal{X}) = \sum_{j=1}^{\tau} \underbrace{-g(\mathrm{L}_j)\log \pi_j}_{=w_j}, \quad g(\mathrm{L}_j) = \sum_{i=1}^{n} \mathbb{1}[\mathrm{L}_j \in \mathcal{T}_i]. \tag{8}$$

Therefore, the objective $J(\boldsymbol{\gamma})$ reduces to finding a sparse vector $\boldsymbol{\gamma} \in \mathbb{R}_+^\tau$ where $\tau \leq n$ by minimizing:

$$J(\boldsymbol{\gamma}) = \sum_{j=1}^{\tau} [\gamma_j - g(\mathrm{L}_j)] \log \pi_j. \tag{9}$$

Let $\sigma(\cdot)$ be a permutation of the induced trees such that $w_{\sigma(1)} \geq w_{\sigma(2)} \geq w_{\sigma(3)} \geq \cdots \geq w_{\sigma(\tau)}$. Then, the optimal solution for a coreset of size k is found by simply setting $\gamma_{\sigma(j)} = g(\mathrm{L}_{\sigma(j)})$ for $1 \leq j \leq k$. Alternatively, given a binary vector $\boldsymbol{b} \in \mathbb{B}^\tau$ we write Eq. (9) as $J(\boldsymbol{b}) = \sum_{j=1}^{\tau} b_j\, w_j$, which is a linear set function.

Corollary 1. *Given a product-free deterministic circuit with indicator leaves, finding an ε-coreset of size k is a linear set function, and the optimal solution can be obtained exactly and gives an approximation of: $\sum_{j=k+1}^{\tau} -g(\mathrm{L}_{\sigma(j)}) \log \pi_{\sigma(j)} \leq \varepsilon\, \mathcal{L}_\theta(\mathcal{X})$.*

3.2 DETERMINISTIC CIRCUITS

We will now discuss general DCs with indicator leaves. Again, we may reformulate the objective of finding a coreset of size k in terms of a set function. Let $\boldsymbol{b} \in \mathbb{B}^n$ denote a binary vector of size n with $b_i = 0$, then we can bound the i^{th} coreset weight to be:

$$0 \leq \gamma_i \leq \min_{(\mathrm{S},\mathrm{N})\in\mathcal{T}_{\boldsymbol{x}_i}} g(\mathrm{S}, \mathrm{N}, \mathbf{1} - \boldsymbol{\gamma} \odot \boldsymbol{b}) \tag{10}$$

where $\odot$ denotes the Hadamard (element-wise) product. Therefore, the coreset objective for DCs with indicator leaves is a cardinality constrained linear program.

Algorithm 1 Greedy Coreset Selection

```
Initialize γ_i = 0 for each i = 1, ..., n;
for j = 1, ..., k do
    u ← 0;
    Δ ← 0;
    for i = 1, ..., n do
        u_i ← compute upper bound (Eq. (10));
        Δ_i ← compute improvement (Eq. (12));
    end for
    c_j ← argmax_i Δ_i;
    γ_{c_j} ← u_{c_j};
end for
Return: c, γ
```

Let the optimal coreset be given by maximising:

$$f_{\boldsymbol{\gamma}}(\boldsymbol{b}) = \sum_{\mathrm{S}\in\mathcal{G}} \sum_{\mathrm{N}\in\mathrm{ch}(\mathrm{S})} [g(\mathrm{S},\mathrm{N}) - \boldsymbol{\gamma}^T(\boldsymbol{v}(\mathrm{N}) \odot \boldsymbol{b})] \log \theta_{\mathrm{S},\mathrm{N}} \tag{11}$$

subject to the constraint $\|\boldsymbol{b}\|_0 \leq k$ and where $\boldsymbol{v}(\mathrm{N}) = (\mathbb{1}[\mathrm{N} \in \mathcal{T}_{\boldsymbol{x}_1}], \ldots, \mathbb{1}[\mathrm{N} \in \mathcal{T}_{\boldsymbol{x}_n}])^T$.

Due to the constraints on $\boldsymbol{\gamma}$, finding the optimal solution for Eq. (11) is a cardinality constrained monotone submodular maximization problem. For the derivation, we refer to Appendix A. Therefore, a greedy algorithm provides a $1 - 1/e$ approximation to the optimal solution [Nemhauser et al., 1978, Fujishige, 2005] in the worst-case. To greedily maximize $f_{\boldsymbol{\gamma}}(\boldsymbol{b})$, we compute the improvement obtained by adding the i^{th} datum at iteration $j < k$ and select the datum which results in the largest marginal gain, *i.e.*,

$$\arg\max_i \sum_{(\mathrm{S},\mathrm{N})\in\mathcal{T}_i} -\gamma_i^* \log \theta_{\mathrm{S},\mathrm{N}} \tag{12}$$

where $\boldsymbol{e}_i$ denotes the i^{th} one-hot coded binary vector of size n and γ_i^* is the optimal coreset weight for the i^{th} datum.

Note that the objective function *w.r.t.* $\boldsymbol{\gamma}$ is a cardinality constrained linear program, *cf.* Appendix A. Therefore, the optimal solution lies on the vertices of the solution polytope [Tanahashi, 1971]. Motivated by this observation, we pick the optimal coreset weight as follows:

$$\gamma_i^* = \min_{\mathrm{L}\in\mathcal{T}_j} g(\mathrm{L}) - \boldsymbol{\gamma}^T(\boldsymbol{v}(\mathrm{L}) \odot \boldsymbol{b} \odot \neg\boldsymbol{e}_j), \tag{13}$$

which is the locally optimal solution if $\mathcal{G}$ is a tree, *cf.* Appendix A for further details. Algorithm 1 summarizes the proposed greedy coreset selection for DCs.

Note that choosing the coreset weights based on the locally optimal solution may result in degeneracy of the greedy algorithm. Therefore, one may optimize the coreset objective directly using a solver for mixed-integer programs [Wolsey, 2007] in exchange for higher computational cost. We leave a detailed discussion for future work.

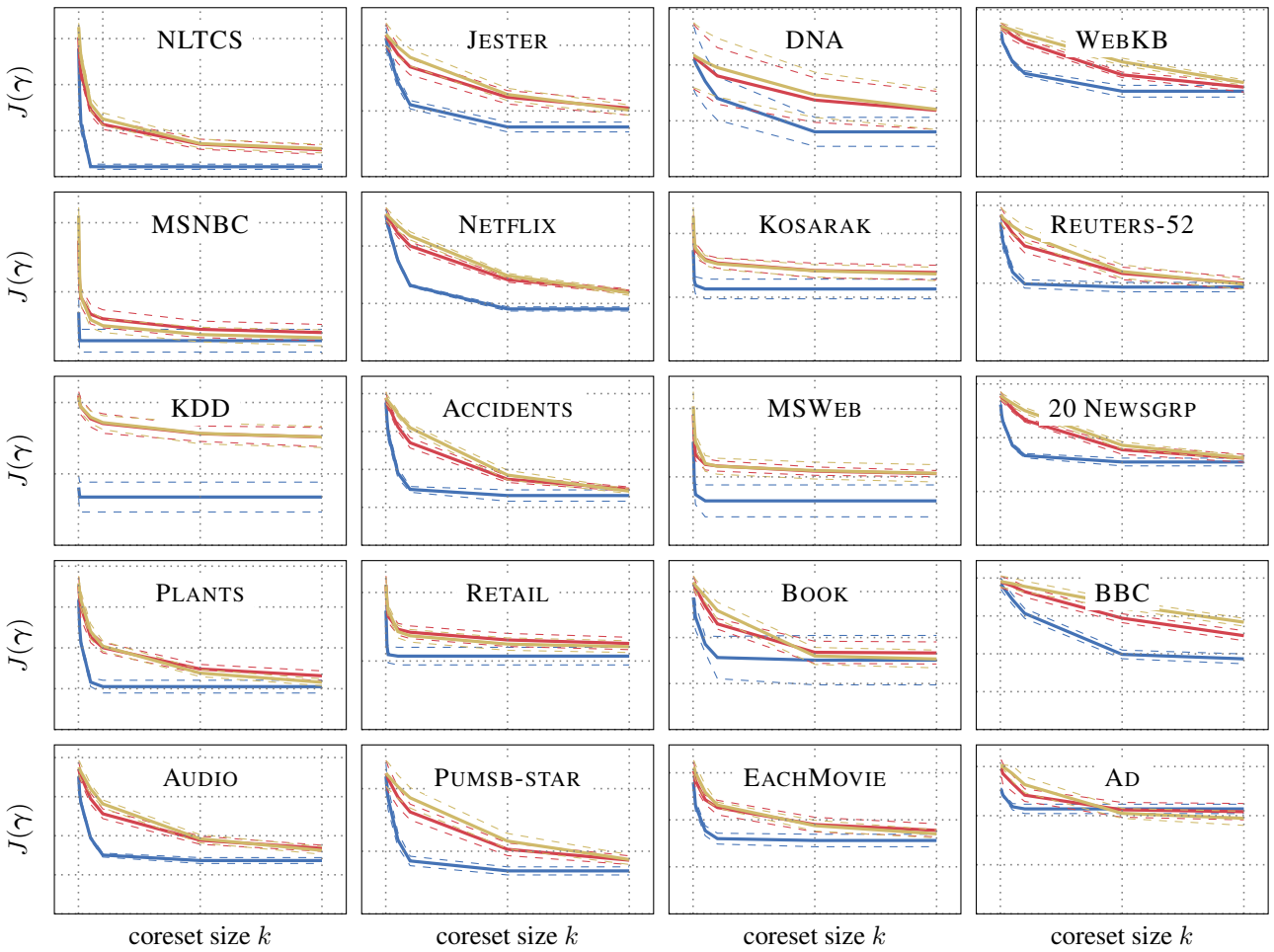

Figure 1: Results on 20 discrete benchmark data sets comparing greedy selection (), random selection with γ_i^* () as weight, and random selection with randomly sampled weights (). Each plots shows the objective function value $J(\gamma)$ (y-axis with minimum $= 0.0$) as a function of the coreset size k (x-axis) for core sets of increasing relative sizes, *i.e.*, $[1\text{e}{-3}\%, 5\text{e}{-3}\%, 0.01\%, 0.05\%, 0.1\%, 0.5\%, 10\%]$ of the training set. All results depict the mean and std compute for ten randomly generated deterministic circuits with fixed maximum depth of 15 layers. **Smaller is better.**

4 EXPERIMENTS

We analyze the performance of greedy coreset selection on 20 commonly used discrete data sets introduced by Lowd and Davis [2010] and Van Haaren and Davis [2012]. See Table 1 for details on the setup and data sets. Each experiment uses ten randomly generated DCs with a maximum of 15 consecutive sum and product nodes and a minimum number of 100 observations per leaf. For reproducibility and fair comparison, we use predefined random seeds.

Fig. 1 depicts the mean and std of the objective function values ($J(\gamma)$) for each data set as a function of the coreset size k for the greedy selection (), random selection using γ_i^* as weight (), and random selection with $\gamma_i \sim \mathcal{U}(0, \gamma_i^*)$ (). Overall, we find that greedy coreset selection performs best. In many cases, greedy coreset selection quickly converges to a local optimum.

5 CONCLUSION & DISCUSSION

In this work, we discussed coreset selection in deterministic probabilistic circuits (DCs). Our results indicate that coreset selection in DCs can be solved exactly for a restricted class of DCs and approximately for a more general class. For the latter, we utilize the fact that the problem can be phrased as a cardinality constrained submodular maximization problem and that the optimal coreset weights are an extreme point on the solution polytope of the respective linear program, allowing us to pick locally optimal coreset weights. We showed on 20 common data sets that greedy coreset selection outperforms random selection and provides a promising avenue for further research.

In the future, we plan to account for the factorization applied by the PC, discuss more general PCs, and explore applications to continual learning and probabilistic programming.

References

Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coreset constructions for machine learning. *arXiv:1703.06476*, 2017.

Vaishak Belle, Andrea Passerini, and Guy Van den Broeck. Probabilistic inference in hybrid domains by weighted model integration. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

Zalán Borsos, Mojmir Mutny, and Andreas Krause. Coresets via bilevel optimization for continual learning and streaming. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:14879–14890, 2020.

Trevor Campbell and Tamara Broderick. Automated scalable Bayesian inference via Hilbert coresets. *The Journal of Machine Learning Research*, 20(1):551–588, 2019.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. Technical report, UCLA, 2020.

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

Satoru Fujishige. *Submodular Function and Optimization*, volume 58 of *Annals of Discrete Mathematics*. Elsevier, 2 edition, 2005.

Sariel Har-Peled and Akash Kushal. Smaller coresets for k-median and k-means clustering. *Discrete & Computational Geometry*, 37(1):3–19, 2007.

Krishnateja Killamsetty, Xujiang Zhao, Feng Chen, and Rishabh Iyer. RETRIEVE: Coreset selection for efficient and robust semi-supervised learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:14488–14501, 2021.

Yitao Liang, Jessa Bekker, and Guy Van den Broeck. Learning the structure of probabilistic sentential decision diagrams. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

Daniel Lowd and Jesse Davis. Learning Markov network structure with decision trees. In *IEEE International Conference on Data Mining (ICDM)*, pages 334–343. IEEE, 2010.

Alexander Munteanu and Chris Schwiegelshohn. Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. *KI-Künstliche Intelligenz*, 32(1):37–53, 2018.

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming*, 14(1):265–294, 1978.

Robert Peharz. *Foundations of Sum-Product Networks for Probabilistic Modeling*. PhD thesis, Graz University of Technology, 2015.

Robert Peharz, Robert Gens, and Pedro Domingos. Learning selective sum-product networks. In *ICML workshop on LTPM*, volume 32, 2014.

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 (ICML)*, 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 *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 334–344, 2020b.

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 337–346, 2011.

Guy Rosman, Mikhail Volkov, Dan Feldman, John W Fisher III, and Daniela Rus. Coresets for k-segmentation of streaming data. In *Advances in Neural Information Processing Systems (NIPS)*, volume 27. Curran Associates, Inc., 2014.

Andy Shih and Stefano Ermon. Probabilistic circuits for variational inference in discrete graphical models. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:4635–4646, 2020.

Tsuguhiko Tanahashi. *Cardinality-constrained linear programming*. PhD thesis, Department of Engineering-Economic Systems, Stanford University, 1971.

Martin Trapp. *Sum-Product Networks for Complex Modelling Scenarios*. PhD thesis, Graz University of Technology, 2020.

Martin Trapp, Robert Peharz, Hong Ge, Franz Pernkopf, and Zoubin Ghahramani. Bayesian learning of sum-product networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6347–6358, 2019.

Nicolas Tremblay, Simon Barthelmé, and Pierre-Olivier Amblard. Determinantal point processes for coresets. *Journal of Machine Learning Research*, 20:168–1, 2019.

Ivor W Tsang, James T Kwok, Pak-Ming Cheung, and Nello Cristianini. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6(4), 2005.

Jan Van Haaren and Jesse Davis. Markov network structure learning: A randomized feature generation approach. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 26, pages 1148–1154, 2012.

Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations for probabilistic inference. *Advances in Neural Information Processing Systems (NeurIPS)*, 34: 13189–13201, 2021.

Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In *International Conference on Machine Learning (ICML)*, pages 1954–1963. PMLR, 2015.

Laurence A Wolsey. Mixed integer programming. *Wiley Encyclopedia of Computer Science and Engineering*, pages 1–10, 2007.

Jacky Zhang, Rajiv Khanna, Anastasios Kyrillidis, and Sanmi Koyejo. Bayesian coresets: Revisiting the nonconvex optimization perspective. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 2782–2790. PMLR, 2021.

Han Zhao, Tameem Adel, Geoff Gordon, and Brandon Amos. Collapsed variational inference for sum-product networks. In *International Conference on Machine Learning (ICML)*, pages 1310–1318. PMLR, 2016.

Supplementary Material:
Towards Coreset Learning in Probabilistic Circuits

This appendix contains further derivations in Appendix A, further details on the proposed algorithm in Appendix B, and details on the experiments in Appendix C. Moreover, we will summarize the notation used in the paper.

General Notation Scalars are written lowercase (*e.g.*, x, y) vectors are written lowercase bold (*e.g.*, $\boldsymbol{x}, \boldsymbol{y}$) and matrices are written uppercase bold (*e.g.*, $\boldsymbol{X}, \boldsymbol{Y}$). Furthermore, the following is used for general mathematical objects.

$\mathcal{X}$	Set of data points
$\lvert\mathcal{X}\rvert$	Cardinality of a set
n	Number of data points, *e.g.*, size of training set
p	Number of dimensions / features
$\boldsymbol{\gamma}$	Coreset weights
γ_i	Coreset weight of i^{th} observation
$\mathcal{C}$	Coreset weighted data set
$\boldsymbol{x}_i$	i^{th} observation
$\boldsymbol{b}$	binary vector
$\boldsymbol{e}_i$	i^{th} one-hot encoded vector
$\mathbf{1}$	vector of ones
$\mathscr{P}(\boldsymbol{X})$	Power set including empty set
$\mathbb{1}[\cdot]$	Indicator function
$\odot$	Element-wise multiplication
$\lVert\boldsymbol{\gamma}\rVert_0$	Pseudo-norm of $\boldsymbol{\gamma}$, *i.e.*, $\sum_i \mathbb{1}[\gamma_i > 0]$

Notation on Probabilistic Circuits The following notation is used for objects related to probabilistic circuits.

$\mathcal{G}$	Graph, *i.e.*, computational graph
ψ	Scope function
$\mathrm{S}, \mathrm{P}, \mathrm{L}$	Sum, Product and Leaf node (respectively)
N	Generic node, *i.e.*, a sum, product of leaf node
$\mathbf{N}$	Set of nodes
$\theta_{\mathrm{S,N}}$	Edge weight for edge from S to N
$\mathcal{L}_\theta$	Negative log-likelihood function
$f(\boldsymbol{x} \mid \theta)$	log-probability function

A DERIVATIONS

Derivation of submodularity Given a deterministic PC $(\mathcal{G}, \psi)$ with indicator leaves and let the coreset objective be defined as:

$$\begin{aligned} \underset{\boldsymbol{b} \in \mathbb{B}^n}{\arg\max} \quad & f(\boldsymbol{b}, \boldsymbol{\gamma}) \\ \text{s.t.} \quad & \lVert\boldsymbol{b}\rVert_0 \leq k \end{aligned} \tag{14}$$

with

$$f(\boldsymbol{b}, \boldsymbol{\gamma}) = \sum_{\mathrm{S} \in \mathcal{G}} \sum_{\mathrm{N} \in \mathrm{ch(S)}} [g(\mathrm{S}, \mathrm{N}) - \boldsymbol{\gamma}^T (\boldsymbol{v}(\mathrm{N}) \odot \boldsymbol{b})] \log \theta_{\mathrm{S,N}}. \tag{15}$$

Further, let $\boldsymbol{b}^{[a]} \subseteq \boldsymbol{b}^{[b]} \in \mathbb{B}^n$ and let $\boldsymbol{e}_i$ denotes the i^{th} one-hot coded binary vector of size n. Then, we have for any $\boldsymbol{e}_i$ with $b_i^{[b]} = 0$ that:

$$f(\boldsymbol{b}+\boldsymbol{e}_i, \boldsymbol{\gamma} \odot (\boldsymbol{b}+\boldsymbol{e}_i)) - f(\boldsymbol{b}, \boldsymbol{\gamma} \odot \boldsymbol{b}) = f(\boldsymbol{e}_i, \boldsymbol{\gamma} \odot (\boldsymbol{b}+\boldsymbol{e}_i)) \tag{16}$$

and

$$f(\boldsymbol{e}_i, \boldsymbol{\gamma} \odot (\boldsymbol{b}^{[a]} + \boldsymbol{e}_i)) \geq f(\boldsymbol{e}_i, \boldsymbol{\gamma} \odot (\boldsymbol{b}^{[b]} + \boldsymbol{e}_i)) \tag{17}$$

due to the fact that

$$[\boldsymbol{\gamma} \odot (\boldsymbol{b}^{[a]} + \boldsymbol{e}_i)]_i \geq [\boldsymbol{\gamma} \odot (\boldsymbol{b}^{[b]} + \boldsymbol{e}_i)]_i \tag{18}$$

by the definition of bound on γ_i given in Eq. (10) where $[\cdot]_i$ denotes the i^{th} element. We, therefore, conclude that $f(\boldsymbol{b}, \boldsymbol{\gamma})$ is submodular and monotone due to $\gamma_i \geq 0$ for all i.

Connection to linear programming Coreset selection in deterministic PC $(\mathcal{G}, \psi)$ with indicator leaves can be formulated as a cardinality-constrained linear program [Tanahashi, 1971].

In particular, let the coreset objective be defined as:

$$\begin{aligned} \underset{\boldsymbol{b}, \boldsymbol{\gamma}}{\arg\max} \quad & \sum_{\mathrm{S} \in \mathcal{G}} \sum_{\mathrm{N} \in \mathrm{ch(S)}} [g(\mathrm{S}, \mathrm{N}) - \boldsymbol{\gamma}^T (\boldsymbol{v}(\mathrm{N}) \odot \boldsymbol{b})] \log \theta_{\mathrm{S,N}} \\ \text{s.t.} \quad & \lVert\boldsymbol{b}\rVert_0 \leq k, \\ & 0 \leq \gamma_i \leq \min_{(\mathrm{S,N}) \in \mathcal{T}_{\boldsymbol{x}_i}} g(\mathrm{S}, \mathrm{N}, \mathbf{1} - \boldsymbol{\gamma} \odot \boldsymbol{b}) \end{aligned} \tag{19}$$

which is a mixed integer program with packing constraints.

Note that, if $\mathcal{G}$ is a tree, then the observation count is an additive function on $\mathcal{G}$, *i.e.*,

$$g(\mathrm{S}, \mathrm{N}) = \sum_{\mathrm{S}' \in \mathrm{ch(N)}} \sum_{\mathrm{N}' \in \mathrm{ch(S')}} g(\mathrm{S}', \mathrm{N}') \tag{20}$$

and is non-increasing for every path in $\mathcal{G}$, *i.e.*, $g(\mathrm{S}, \mathrm{N}) \geq g(\mathrm{S}', \mathrm{N}')$ for every $\mathrm{S}' \in \mathcal{G}$ that is a descendant of N.

Therefore, it is sufficient to consider coreset weight constraints only for the edges to the leaves of $\mathcal{G}$, *i.e.*,

$$0 \leq \gamma_j \leq \min_{\mathrm{L} \in \mathcal{T}_j} g(\mathrm{L}) - \boldsymbol{\gamma}^T (\boldsymbol{v}(\mathrm{L}) \odot \boldsymbol{b} \odot \neg \boldsymbol{e}_j) \qquad (21)$$

where $\boldsymbol{v}(\mathrm{N}) = (\mathbb{1}[\mathrm{N} \in \mathcal{T}_{\boldsymbol{x}_1}], \ldots, \mathbb{1}[\mathrm{N} \in \mathcal{T}_{\boldsymbol{x}_n}])^T$ and $\boldsymbol{e}_j$ denotes the j^{th} one-hot coded binary vector.

B ALGORITHM

This section presents further details on the proposed greedy selection algorithm.

In praxis, computing LINE 6 has computational cost of $\mathcal{O}(|\mathbf{L}|)$ and LINE 7 a computational cost of $\mathcal{O}(|\mathcal{G}|)$. Both computations can be parallelized across nodes while the inner for loop over the observations is parallelisable over observations, resulting in an overall computational cost of $\mathcal{O}(k \log(n) \log(|\mathcal{G}|))$ for a parallelised implementation.

C EXPERIMENTS

This section contains details on the experiments conducted. Appendix C.1 contains detail on the experimental setup, and Appendix C.2 contains a listing of the data sets used in the experiments depicted in the main text.

C.1 EXPERIMENTAL SETUP

For each data set, we randomly generated ten DCs with a maximum of 15 consecutive sum and product nodes and a minimum of 100 observations per leaf. Each random DC structure is generated based on a simple recursive algorithm that produces randomly defined deterministic sum nodes and product nodes until the maximum depth or the minimum number of observations is reached. In particular, for each sum node, we choose to split the data into two disjoint subsets. The splitting is chosen randomly, and if no separation into two sub-sets can be found, we construct a leaf node instead. Note that this is a pretty naïve approach, and a more elaborate strategy will likely result in a more accurate model. We then recurse to generate a product node. We randomly separate the input dimensions into two disjoint sub-sets for each product node. If the number of dimensions is one, we construct a product node with a single child. Afterwards, we recurse to generate a new sum node. This procedure continues until no partition in two sub-sets of the data can be found, a depth of 15 consecutive sum and product nodes has reached, or the number of observations in a sub-set is below 100. We used the following random seeds for reproducibility: $[1, \ldots, 10]$.

Random selection with γ^* Random selection with γ^* relies on the computation of Eq. (10). In particular, we first draw an observation i at random without replacement. Then we obtained γ_i^* for the respective observation. After this, we draw another observation at random without replacement and re-evaluate Eq. (10) to set the coreset weight accordingly. This procedure continues until we have selected a pre-specified number of coreset members.

Random selection with random weights Random selection with random weights picks a weight in the range defined by Eq. (10) at random. In particular, we first draw an observation i at random without replacement. Then we obtained $\gamma_i \sim \mathcal{U}(0, \gamma_i^*)$ for the respective observation. After this, we draw another observation at random without replacement, re-evaluate Eq. (10) and draw the coreset weight from the updated uniform distribution. This procedure continues until we have selected a pre-specified number of coreset members.

C.2 BINARY BENCHMARK DATA SETS

Table 1 lists the binary benchmark data sets used in the empirical evaluation of the greedy coreset selection algorithm.

Table 1: Commonly used discrete benchmark data sets.

Data set	p	n_{train}	n_{valid}	n_{test}
NLTCS	16	16181	2157	3236
MSNBC	17	291326	38843	58265
KDD	65	180092	19907	34955
PLANTS	69	17412	2321	3482
AUDIO	100	15000	2000	3000
JESTER	100	9000	1000	4116
NETFLIX	100	15000	2000	3000
ACCIDENTS	111	12758	1700	2551
RETAIL	135	22041	2938	4408
PUMSB-STAR	163	12262	1635	2452
DNA	180	1600	400	1186
KOSAREK	190	33375	4450	6675
MSWEB	294	29441	3270	5000
BOOK	500	8700	1159	1739
EACHMOVIE	500	4524	1002	591
WEBKB	839	2803	558	838
REUTERS-52	889	6532	1028	1540
20 NEWSGRP.	910	11293	3764	3764
BBC	1058	1670	225	330
AD	1556	2461	327	491