BAYESCARD: A Unified Bayesian Framework for Cardinality Estimation

Ziniu Wu
Alibaba Group
ziniu.wzn@alibaba-inc.com

Amir Shaikhha
University of Edinburgh
amir.shaikhha@ed.ac.uk

ABSTRACT
Cardinality estimation is one of the fundamental problems in database management systems and it is an essential component in query optimizers. Traditional machine-learning-based approaches use probabilistic models such as Bayesian Networks (BNs) to learn joint distributions on data. Recent research advocates for using deep unsupervised learning and achieves state-of-the-art performance in estimating the cardinality of selection and join queries. Yet the lack of scalability, stability and interpretability of such deep learning models, makes them unsuitable for real-world databases.

Recent advances in probabilistic programming languages (PPLs) allow for a declarative and efficient specification of probabilistic models such as BNs, and achieve state-of-the-art accuracy in various machine learning tasks. In this paper, we present BayesCard, the first framework incorporating the techniques behind PPLs for building BNs along with relational extensions that can accurately estimate the cardinality of selection and join queries in database systems with model sizes that are up to three orders of magnitude smaller than deep models. Furthermore, the more stable performance and better interpretation of BNs make them viable options for practical query optimizers. Our experimental results on several single-relation and multi-relation databases indicate that BayesCard with a reasonable estimation time has a better estimation accuracy than deep learning models, and has from one to two orders of magnitude less training cost nevertheless.

1 INTRODUCTION
Cardinality estimation is one of the fundamental problems in database management systems and it is an essential component in query optimizers [31, 34, 52]. The cardinality of a query \( Q \) refers to the total number of tuples in its result. The search for good execution plan of a given query essentially relies on good cardinality estimation on its subqueries. Specifically, estimating the cardinality of multi-relation join queries is the key to find the best join order, which is arguably the most important step during query optimization [31]. After several decades of research on cardinality estimation, this problem remains notoriously challenging [31, 33].

Recently, cardinality estimators based on deep probabilistic models, such as sum-product-networks (SPNs) [24], FSPN [65], and deep auto-regressive (DAR) models [22, 61, 62], have achieved the state-of-the-art performance. Despite their high estimation accuracy and relatively fast inference speed, none of the commercial query optimizers use these methods in their system. In fact, most commercial optimizers tend to use much simpler cardinality estimators such as histogram methods with strong assumptions such as attribute independence and join uniformity to simplify the problem. Bayesian networks (BNs) are well-known graphical models employed in various machine learning tasks [27]. Since they can compactly estimate the joint data distribution, BNs are naturally suitable for cardinality estimation [17, 21, 57]. However, the structure learning and inferences of BNs can be generally inefficient.

In this paper, we employ the recent advances in the programming languages and machine learning communities on probabilistic programming languages (PPLs) [2, 4, 40, 51, 56] for cardinality estimation. PPLs allow for a declarative specification of probabilistic models and have been successfully used for several machine learning tasks [29]. In this work, we incorporate the techniques implemented in various PPLs to efficiently generate small and accurate BNs that can be used to estimate the cardinalities of selection and join queries. Furthermore, BNs have apparent superiority over the “deep” models from the following aspects, which are essential for query optimizers:

Scalability: Real-world databases (DBs) may contain many relations [31]. Each relation requires an individual model to represent its data distribution. Thus, the model size and training time are of high importance to a database [13, 14]. Using the attribute conditional independence property, BNs have much more compact structures than the deep probabilistic models. Although the exact structure learning in BNs is an NP-hard problem [8], there exist approximate variants that are very efficient by constraining the search space without sacrificing accuracy in practice. As we show in the experimental results, the BNs learnt by these algorithms have estimation accuracy comparable or even better than the deep probabilistic models, in addition to up to two orders of magnitude improvement in model size and training time.

Stability: As mentioned, real-world databases may contain many relations with possibly very different structures, requiring the cardinality estimation models to have a consistent estimation performance for all relations. Unfortunately, the current state-of-the-art cardinality estimation methods, DeepDB [24] based on SPN [47], FLAT [65] based on FSPN [60], as well as Naru [62] based on DAR [16, 41, 59], do not have such stability. 1) The SPNs used in DeepDB have limited expressive efficiency in the case of highly correlated attributes [38]. For example, in the case of two functionally dependent attributes, DeepDB requires very deep SPN structures to represent their distributions, as illustrated in Section 2.3. In general, real-world DBs would contain sets of strongly correlated attributes. Thus, when modeling these DBs, SPN model size and training time will be prohibitively large and the estimation time and accuracy will be poor at the same time. 2) The DAR models used in Naru learn an exact representation of the joint data distribution without using the attribute independence or conditional independence properties. Thus, the learning space of these models grows exponentially with the number of attributes in the relations. Hence, with an increase in the number of attributes, their model size will increase significantly.
On the contrary, a BN maintains a compact structure using the conditional independence property. Therefore, for a relation with a large amount of strongly correlated attributes, BNs can capture their dependence unlike SPNs. Furthermore, for a relation with any independent or conditionally independent attributes, BNs can detect them and constrain the model size, as opposed to DAR models.

**Interpretability:** For commercial query optimizers, model interpretability is also an important factor of consideration because DB experts who investigate and tune query plans would like to understand why the system uses a particular plan. In addition, interpretable models are easier to maintain, validate and improve with expert knowledge. For example, one can always validate or improve a BN based on prior knowledge about the dataset such as known conditional independence within the data. On the contrary, the deep probabilistic models serve as black-box approximators of joint distribution and thus do not give users much information to reason with, as illustrated by Chakraborty et al. [6].

**Contributions:** The contributions of this paper are as follows:
- We present BayesCard, a framework that employs the recent advances in probabilistic programming for the problem of cardinality estimation (cf. Section 3). This framework unifies all the existing work on using BNs for cardinality estimation [17, 21, 57] by incorporating all the structure learning, parameter learning and inference algorithms for BNs employed in existing mainstream probabilistic programming packages and languages (cf. Table 1).1
- We extend BayesCard for estimating the cardinality of join queries (cf. Section 4). More specifically, we adapt an accurate and efficient estimation method based on fanout attributes [65] into our framework. Based on this method, we design a novel model construction method, which balances the model training cost and estimation accuracy.
- We experimentally compare different BN-based cardinality estimation approaches realisable by BayesCard to the current state-of-the-art deep probabilistic models on various benchmarks (cf. Section 5). Our results suggest that BN’s performance accuracy is comparable or even better than the deep probabilistic models but the inference time is slightly slower. However, the model size and training time are up to two orders of magnitude better.

2 BACKGROUND AND PROBLEM FORMULATION

In this section, we first formally define the cardinality estimation problem from both database and statistical perspectives, and then briefly explain the machine learning methods for cardinality estimation. The following notations will be used throughout the paper.
- A single capital letter (eg. T, R, A, B, C) denotes a relation in the DB. Alternatively, the capitalized specific name (eg. CENSUS, DMV) also represents the specific relation.
- A single capital letter with an index subscript (eg. Ti, Tj) or a dot followed by another capital letter (eg. T.X or T.Y) denotes the column attributes in relation T. Alternatively, the specific word (eg. Age, Sex) also represents the specific attribute.
- A single lower case letter, t denotes a tuple in relation T. t.X denotes the value of attribute T.X in this particular tuple t.
- A single lower case letter with an index subscript, ti denotes the specific values from the domain D(Ti) of Tj.
- A single lower case letter with an index subscript, ti denotes the specific values from the domain D(Ti) of Tj.

From statistical perspective, we have the following definition:
- \( p_T(T_1) \) denotes probability density function (PDF) of relation T with n attributes.
- \( p_T(T_1) \) denotes probability density function (PDF) of relation T with n attributes.
- \( P_T(T_1, T_2, \ldots, T_n) \) denotes probability density function (PDF) of relation T with n attributes.

2.1 Estimation for single relation query

Single relation query refers to the query requesting information only from one relation in the DB. The following single relation query written in SQL represents selecting total number of people under 28 with at least two children.

\[
\text{SELECT COUNT(*) FROM CENSUS WHERE Age < 28 AND Child \geq 2} \quad (1)
\]

Without loss of generality, every query Q on a single relation T with n attributes \( T_1, \ldots, T_n \) can be represented as \( Q = \{ T_{i1} \in d(T_{i1}) \land T_{i2} \in d(T_{i2}) \land \ldots \land T_{ik} \in d(T_{ik}) \} \), where each \( T_{ij} \in d(T_{i}) \) is called a filter predicate. Q is called a point query if \( |d(T_{ij})| = 1 \) for every filter predicates in Q. Thus in this case, we can write Q as \( T_{i1} \land T_{i2} \land \ldots \land T_{ik} = t_{ij} \). Otherwise, Q is called a range query. For simplicity of illustration, our examples throughout this paper will be point queries.

Now consider query \( Q = T_i \land T_j = t_j \). Denote the cardinality of Q as \( C(Q) \), then we can formulate C as

\[
C(T_i \land T_j = t_j) = P_T(T_i = t_i, T_j = t_j) * |T| \quad (2)
\]

Thus the query cardinality estimation can essentially be reduced to estimating the joint distribution \( P_T(T_1, T_2, \ldots, T_n) \) from the data or data samples in T. The storage of exact representation of high dimensional joint PDF grows exponentially with the number of attributes and thus is almost always intractable. Fortunately, many compact probabilistic models have been proposed for this task with relatively accurate representation of the joint distribution, such as BNs [45], SPNs [47] and DAR models [16].

2.2 Estimation for multi-relation join query

A practical DB may contain many relations and the attributes being queried are very likely to be distributed across several relations. This requires different types of join operations to pair up tuples from different relations. A natural join between two relations T and R is an operation which produces a new relation \( \Omega \), such that \( \Omega \) contains attributes from both T and R. The join predicate \( T.K = R.K \) specifies that for every pair of tuples t and r, if \( t.K = r.K \), t and r will be combined as a single tuple in \( \Omega \). In this case, K is called the join key. There are four types of natural join, namely inner join \( \Join \), left outer join \( \LeftJoin \), right outer join \( \RightJoin \) and full outer join \( \LeftJoin \). For simplicity of illustration, we will treat the full outer join symbol, \( T \Join R \) as a general join between T and R throughout the paper.

Consider a join query Q involving T and R with filter predicates \( T_i = t_i \land R_j = r_j \). We represent the cardinality \( C(Q) \) as \( C(T_i = t_i \land R_j = r_j) \). Similar to the single relation case as

1The features of these languages that are related to graphical models other than BNs is beyond the scope of this framework and not as important as BNs in cardinality estimation.
Equation 2, we can write as Equation 3, where $\Omega = T \bowtie R$ denotes join of $T$ and $R$.

$$C(Q) = P_{\Omega}(T_i = t_i, R_j = r_j) \cdot |\Omega|$$  \hspace{1cm} (3)

Representing the joint distribution on the join of multiple relations can be particularly hard since theoretically we can have exponential number of possible joins between relations in a DB. Thus it is prohibitive to learn a distribution over every join of relations. In addition, the size $|R \bowtie S|$ can get up to $|R| \cdot |S|$, making it unaffordable to join several relations together to learn its probability distribution. Therefore, many approaches have been proposed to estimate the cardinality of multi-relation join queries.

The current state-of-the-art deep probabilistic approaches [24, 61] all use the unsupervised model to achieve an accurate representation of single relation joint distribution; then use some join estimation approaches to ensemble the results from single relations. The details of these approaches will be illustrated in Section 4.

### 2.3 Machine Learning for Cardinality Estimation

In general, there are two directions machine learning (ML) techniques approach the cardinality estimation problems: supervised learning (query-driven) and unsupervised learning (data-driven).

**Supervised cardinality estimation methods** use the feedback of past queries to train ML models. The input is featurized query $Q$ and the output is the actual cardinality of $Q$, $C(Q)$. Then ML models will be trained as regression between input and output. The first approach using neural networks on cardinality estimation was published for UDF predicates [30]. Later on, a regression-based model [1] and a semi-automatic alternative [37] were presented. Then, an index based join sampling algorithm (IBJS) [32] used qualifying base relation samples to estimate the cardinality of the join of multiple relations. However, IBJS would fail when there are no qualifying samples for a particular query, causing large estimation error. Recently, a deep learning approach using multi-set convolutional network (MSCN) [26] efficiently solved this problem.

The supervised learning approaches have two major drawbacks as mentioned in [24]: 1) Their model neglects the data itself and heavily relies on query featurization. 2) Collecting the training data can be very expensive and training data has to be recollected heavily replies on query featurization. 2) Collecting the training data can be very expensive and training data has to be recollected when the workload changes. Therefore, in general, query-driven supervised ML methods on cardinality estimation are not as flexible and accurate as data-driven unsupervised ML methods.

**Unsupervised ML methods** characterize the PDF $P_T$ of relation $T$ with $n$ attributes $X = \{T.X_1, \ldots T.X_n\}$. Previously, people proposed to circumvent this issue by making convenient assumptions, such as the attribute value independence, under which $P_T$ can be estimated as the product of single-attribute distributions [39]. Later, people started to use BNs for cardinality estimation [17, 21, 57, 58] using the conditional independence property. Recently, deep probabilistic models, such as sum-product-networks (SPNs) [47] and deep auto-regressive (DAR) models [16, 41, 59] have achieved state-of-the-art results for cardinality estimation.

**SPNs** used by DeepDB [24], approximate $P_T$ by recursively decomposing the more complex distributions into local and simpler distributions. Specifically an SPN defines a tree-structured graphical model, with each node representing a PDF $P_T(X')$ where $X' \subseteq \text{All, } T' \subseteq T$ and the root node representing $P_T(X)$. There are three node types in an SPN, namely sum, product and leaf nodes. Each sum node splits the current data tuples $T'$ into $k$ partitions $T'_i$, each with a weight $w_i$. Thus, the PDF represented by this sum node can be written as $P_T(X') = \sum_{1 \leq i \leq k} P_T(X'_i) \cdot w_i$. Each product node detects the $k$ partitions of $X'$, where attributes in each partition $X'_i$ are mutually independent. Thus, the PDF represented by this product node can be written as $P_T(X') = \prod_{1 \leq i \leq k} P_T(X'_i)$. Each leaf node models the PDF of a single attributes using a histogram or a uni-variate distribution. The probability inference in SPNs can be efficiently computed in a single traversal from leaves to root. However, SPN has limited expressiveness: there exists certain simple distributions that an SPN with an arbitrary depth cannot represent [38].

**DAR models** used in Naru [62], learn an exact representation of the joint data distribution as follows.

$$P_T(X) = P_T(T.X_1) \cdot P_T(T.X_1 | T.X_2) \cdots P_T(T.X_n | T.X_1, \ldots, T.X_{n-1})$$

Each conditional PDF in this equation is approximated by a deep neural network (DNN). The lossless factorization and rich expressiveness of DNN make DAR models very accurate in probability inference. However, these DNNs can have large model size with very slow model training speed, especially for large relations. Meanwhile, the probability inference for large DAR models is also inefficient.

### 3 BAYESCARD FRAMEWORK

This section starts with the introduction to the general Bayesian cardinality estimation framework BayesCard. Using probabilistic programming, it efficiently implements Bayesian networks’ (BNs) existing mainstream algorithms for structure learning, parameter learning and probability inference as shown in Table 1. The details of these algorithms will be explained in Section 3.3 and Section 3.4.

#### 3.1 A unified Bayesian cardinality estimation framework

We present the general single relation cardinality estimation procedure using BNs in Figure 1. Specifically, for a relation $T$, BayesCard will first construct a BN representing the PDF of $T$, $P_T$. Later, a selection query $Q$ will be given to the learned BN and the estimated cardinality of $Q$ will be computed as $P_T(\Omega) \cdot |\Omega|$. The process of retrieving the probability $P_T(\Omega)$ on a learnt BN is called probability inference in BN.

BayesCard is a unified framework for different BN’s main-stream structure learning and inference algorithms. It supports two types of exact structure learning algorithms [5, 63], one greedy heuristics structure learning algorithm [9], one Chow-liu tree based algorithm [11] and the method for converting learned BN into a junction tree for fast probability inference. The details of these structure
Table 1: Comparison on BNs’ structure learning, inference algorithms and join query estimation methods between different probabilistic programming frameworks (above the thick line) and the existing papers using BNs in cardinality estimation (Below the thick line). Note 1: [17] uses a fanout method with stronger arguments than our approach (cf. Section 4.1). Note 2: [57] uses a method based on biased and inefficient sampling algorithms whereas our fanout method uses an unbiased and efficient one.

### 3.2 Semantics of Bayesian networks

Consider learning the joint PDF $P_T(T_1, T_2, \ldots, T_n)$ of relation $T$ with $n$ attributes. We can factorize the joint probability $P_T$ into:

$$P_T(T_1) \cdot P_T(T_2|T_1) \cdot \cdots \cdot P_T(T_n|T_1, \ldots, T_{n-1})$$  \hspace{1cm} (4)

Representing these conditional distributions exactly is almost always intractable. Fortunately, in practice, not every attribute is directly dependent on every other attributes. There exist many independence and conditional independence that can be harnessed to simplify this exact factorization in Equation 4. More specifically, when modeling the conditional PDF of $P(T_4|T_1, T_2)$, if $T_4$ and $T_2$ are independent, then this conditional PDF is equivalent to $P(T_4|T_2)$.

Attribute conditional independence is slightly harder to understand. For example in CENSUS relation, a person’s working industry determines his/her occupation and the income is closely related to the occupation. Even though industry and income are dependent, given the information of one’s occupation, industry and income do not provide additional information about each other. Thus in this case, we say a person’s income and working industry are conditionally independent given his/her occupation, which means $P(\text{Income}(\text{Occupation, Industry}) = P(\text{Income} | \text{Occupation})$.

Conditional independence is widely present in every real-world data relations [17], which is the key foundation of BNs [27]. A BN defines a directed acyclic graph (DAG) as in Figure 2, where each node corresponds to an attribute and each edge defines the causal dependency between two nodes. The attributes in BN satisfy the first order Markov property, which specifies that an attribute is dependent on its parents and conditionally independent of all other attributes given its parents. Thus, Equation 4 can be compactly written as $P_T(T_1, \ldots, T_n) = \prod_{i=1}^{n} P_T(T_i | \text{Parents}(T_i))$, where $\text{Parents}(T_i)$ denotes the set of parents of $T_i$ in BN’s defined DAG.

### 3.3 Model construction

BN’s parameter estimation and structure learning are two main components of model construction. Parameter estimation consists of estimating the conditional probability distribution (CPD) $P(T_i | \text{Parents}(T_i))$ of all attributes in $T$. Representing CPD has been a well studied problem in graphical models. The most common approach is to discretize the continuous attribute domains into categorical ones and represent each CPD in a tabular form. Alternative, a tractable family of distributions can be used to represent CPDs for continuous attributes [20].

Unlike parameter estimation, structure learning is still an active area of research since it is an NP-hard combinatorial optimization problem [8]. The goal of structure learning is to find a DAG that captures the casual structure of relation $T$ based on n independent and identically distributed (i.i.d.) observations $T^n \in \mathbb{R}^{n \times d}$, where $d$ is the total number of attributes in $T$. A DAG with $d$ nodes can be represented as a $d \times d$ adjacency matrix $M$, with $M[i, j] \neq 0$ refers to an edge from attribute node i to attribute node j. Given a scoring criteria $L$, which evaluates how well $M$ fits the data, the structure learning objective function can be written as Equation 5.

$$\arg\min_{M} L(M, T^n) = \frac{1}{n} \sum_{i=1}^{d} L(T'_i, T' \cdot M'_i)$$

subject to $G(M) \in \mathbb{D}$

$$T'_i \in \mathbb{R}^n \text{ refers to ith column of } T', M'_i \in \mathbb{R}^d \text{ refers to transpose of ith column in } M \text{ and denotes the matrix multiplication. Scoring criteria } L \text{ is commonly chosen to be the least square or negative log likelihood function } [17].$$

The structure learning algorithm of BN can be roughly categorized into three categories: exact algorithms directly solve Equation 5, greedy algorithms approximate the solution of this equation and Chow-Liu tree algorithm restricts the learning space to tree structured BNs.

**Exact algorithms:** Early structure learning algorithms [5, 10, 23] use different methods such as dynamic programming to explore the search space of all DAGs and select the structure that maximizes the
These algorithms can process up to hundreds of attributes. The most commonly used approximate algorithm is Chow-Liu tree algorithm: Alternatively, the CLT [11] based algorithm can easily scale to more than thousands of attributes. The scoring function in CLT is the mutual information (MI). Initially, the algorithm calculates the MI between every pair of attribute nodes. These MI values define a fully connected graph where each value is translated to a weighed edge. Then a minimum spanning tree of this graph is retrieved as the optimal structured BN. This algorithm is efficient to use but can be inaccurate for complicated relations. For example, in Figure 2, the CLT structured BN characterizes a dependency between “Age” and “Sex” where such dependency should not exist in US demographics. Also, the number of children “Child” should also be determined by “Income” but the CLT algorithm removes this dependence in order to preserve the tree structure.

Junction tree algorithm: Apart from these three classes of BN structure learning algorithms, people sometimes transform a learned BN into a junction tree for more efficient probability inference. Junction tree [25] is an undirected tree-structured graphical model. The standard algorithm first turns a learned BN into a chordal moral graph by moralization and triangulation [3] and then constructs a junction tree from the chordal moral graph [12].

3.4 Probability Inference in Bayesian networks

Probability inference in BN refers to calculating the probability \( P_T(Q) \) of query \( Q \)’s occurrence in relation \( T \). In general, the inference algorithms can be grouped into exact or approximate algorithms. Exact algorithms includes variable elimination on any BN structures and belief propagation on tree structured graphical models. The most commonly used approximate algorithm is sampling.

Variable elimination (VE): The most commonly used exact inference algorithm is the VE algorithm [27]. We will illustrate the VE algorithm with the above example, CENSUS relation with tree-structured BN as in the left image of Figure 2. For clear notation, let us denote \( Ag = “Age”, Se = “Sex”, Re = “Relatives”, Sc = “School”, Ma = “Marital”, Mi = “Military”, Ye = “Yearwork”, Ind = “Industry”, Ch = “Child”, Oc = “Occupation” and Inc = “Income”. Let \( All \) denote the set of all these attributes. Consider a query \( Q = (Se = 1 \land Ch = 1) \), i.e. total number of female with exactly one child. By law of total probability, we can sum over all attributes other than \( Se \) and \( Ch \) as illustrated in Equation 6:

\[
P_T(Se = 1, Ch = 1) = \sum_{ag \in D(Ag)} \cdots \sum_{inc \in D(Inc)} P_T(\cdots , Se = 1, Ch = 1, \cdots)
\]

\[
P_T(Se = 1, Ch = 1) = \sum_{ag \in D(Ag)} \cdots \sum_{inc \in D(Inc)} P_T(\{ag\}) \cdot P_T(\{Se = 1|ag\}) \cdot \cdots \cdot P_T(\{inc\}|oc)
\]

(6)

The VE algorithm selects one attribute at a time from the attributes that are not in the query, which in this case are \( All/\{Se, Ch\} \). It starts by eliminating \( Inc \). After computing \( M_{T}(Oc = oc) = \sum_{inc \in D(Inc)} P_{T}(\{inc\}|oc) \cdot P_{T}(\{inc\}|oc) \) for each value \( oc \in D(Oc) \), attribute \( Inc \) is eliminated from Equation 6 and derives Equation 7.

\[
P_T(Se = 1, Ch = 1) = \sum_{ag \in D(Ag)} \cdots \sum_{oc \in D(Oc)} (P_T(\{ag\}) \cdot P_T(\{Se = 1|ag\}) \cdot \cdots \cdot M_{T}(Oc = oc))
\]

(7)

In this way, VE can eventually eliminate all attributes in \( All/\{Se, Ch\} \), resulting in \( P_T(Se = 1, Ch = 1) \). The VE can have exponential complexity in the worst case [45], but for tree-structured BNs, it is relatively efficient. Since each attribute in tree-structured BNs can have at most one parent, the time complexity is \( O(n \cdot D_{max}) \) as long as we follow the elimination order in reversed tree order, where \( n \) is the number of attribute nodes in the tree and \( D_{max} \) is the maximum value domain size of all attributes.

Belief propagation (BP): Variable elimination’s time complexity is prohibitive for large graphs with complex structures. Therefore, sometimes a tractable inference algorithm is required, such as BP algorithm [45]. BP is motivated by the observation that performing VE on a tree is equivalent to message passing along tree branches, where messages are real-valued functions conveying information about the attribute nodes. BP defines a two steps message passing algorithm. In the first step the root collects messages from all nodes. Starting at the leaves, each node passes a message to its parent until the root has obtained messages from all nodes. The second step involves the root passing the messages backward in the reverse direction. The details of this algorithm can be found at [25, 45].

BP is an exact inference algorithm for tree structured graphical models, such as tree-structured BNs or junction trees and one iteration of these two steps of message passing is guaranteed to find the exact solution for probability inference. As for general DAG-structured BNs, the result of BP will converge to the exact probability after many iterations of message passing [45], which might be inefficient. Thus, reasonable BP algorithms will terminate after a few iterations so it is an approximate probability inference algorithm for general DAG-structured BNs.

Alternatively, one can convert a learnt BN into a tree-structured undirected graphical model, junction tree and then perform efficient and exact BP probability inference on the junction tree. This approach is used by [57, 58] to construct a junction from BN for cardinality estimation.
Sampling: Sampling algorithm approximates the query’s probability by generating samples from BN [27]. A BN is an generative model that can generate data samples from its learned structure. Ideally, these samples should follow the similar distribution as the training data. Thus, in order to estimate \( P_T(Se = 1 \land Ch = 1) \), one can count the number of tuples satisfying \( Se = 1, Ch = 1 \) in the generated samples and divide it by the total number of sampled tuples. However, to ensure estimation accuracy, BNs need to generate large amount of samples, which can be inefficient. Furthermore, this algorithm is not very useful in DB settings, since one can directly fetch samples from the original data to perform cardinality estimation, which can be more efficient and accurate. Thus we do not report the performance of BNs using sampling inference algorithm in our experimental results as it is not very informative.

Graph Reduction: In fact, VE and BP involve a large amount of redundant computation because not all attributes are relevant to compute the probability of query \( Q \). Let us reconsider the previous example \( Q = (Se = 1 \land Ch = 1) \) in a tree-structured BN (left image in Figure 3). We can compute the probability \( P_T(Q) \) as in Equation 8, where the first equality is achieved by law of total probability, second equality is derived by chain rule factorization and the third equality is achieved by BN’s conditional independent assumption. However, in BN parameter learning, each node attributes will only store the CPD table as discussed in previous section. Thus, we need to eliminate node “Age” in order to get the marginal probability \( P_T(Se = 1) \), i.e. \( P_T(Se = 1) = \sum_{Ag \in Ag} P_T(Se = 1 | Ag = ag) \). Therefore, we can deduce that in this example running VE or BP on the full graph is equivalent to running on the reduced graph, which only contains four nodes “Age”, “Sex”, “Marital” and “Child” and the three edges between them.

\[
P_T(Se = 1, Ch = 1) = \sum_{ma \in D(Ma)} P_T(Se = 1, Ma = ma, Ch = 1)
\]

\[
= \sum_{ma \in D(Ma)} \left[ P_T(Se = 1) \cdot P_T(Ma = ma | Se = 1) \right.
\]

\[
\cdot P(Ch = 1 | Ma = ma, Se = 1)
\]

\[
= \sum_{ma \in D(Ma)} \left[ P_T(Se = 1) \cdot P_T(Ma = ma | Se = 1) \cdot P(Ch = 1 | ma) \right]
\]

Theorem 1. Given a BN and its defined DAG \( G = (V, E) \) representing a relation \( T \) with attributes \( V = \{T_1, \ldots, T_n\} \) and a query \( Q = (T^*_1 = t'_1 \land \cdots \land T^*_k = t'_k) \) where \( T^*_i \in \{T_1, \ldots, T_n\} \). Let \( G' = (V', E') \) be a subgraph of \( G \) where \( V' = \bigcup_{i \in S} \text{Ancestor}(T^*_i) \) and \( E' \) is all edges in \( E \) with both endpoints in \( V' \). \text{Ancestor}(T^*_i) \) includes all parent nodes of \( T^*_i \) and all parents of parent node recursively. Then performing VE or BP on full graph \( G \) is equivalent to running VE or BP on reduced graph \( G' \).

The proof of Theorem 1 is given in the Appendix of the accompanied technical report. With this algorithm, the probability inference of BN can be made much faster especially for relations with a large amount of attributes.

4 MULTI-RELATIONAL EXTENSION OF BAYESCARD

In this section, we explain how to extend our BayesCard framework to estimate multi-relation join queries. In the first subsection, we first introduce the background of probabilistic relational models (PRMs) which extend BNs to relational DB settings. Then we briefly discuss the join uniformity assumption and two previous approaches [17, 57] to learn PRM but they are flawed and impractical for real-world DBs. In the rest of this section, we will thoroughly explain our improved method for estimating join queries using fanout attribute correction.

4.1 Existing methods for PRMs

Probabilistic relational models (PRMs) [28] extend BNs to the relational settings. They support modeling attributes not only within the single relation but across relations in a DB as well.

Join uniformity method: The join uniformity assumption is the most common approach used by mainstream query optimizers, which states that a tuple from one relation can be joined with any tuple from the second relation with equal probability. Normally, under this assumptions, people also assume that the attributes between two relations are independent. Thus, the cardinality of query \( Q = (T_i = t_i \land R_j = r_j) \) can be rewritten as:

\[
C(Q) = P_T(T_i = t_i) \cdot P_R(R_j = r_j) \cdot |T \bowtie R|
\]

Therefore, under this assumption one can construct a PRM, which is equivalent to a BN in this case, for each relation in the DB. This approach is easy to maintain and does not require additional storage in practice but join uniformity assumption can be easily violated. For example, large movie companies tend to produce a lot more movies than small companies so in IMDB database, each tuple in relation “movie_companies” is not equally likely to join with every tuple in relation “title”.

Referential integrity method: Another approach to learn PRMs proposed by Getoor et al. [17] is restricted to DBs satisfying referential integrity (RI) assumptions: Let \( T \) be a relation, and let \( FK \) be a foreign key in \( T \) that refers to some relation \( R \) with primary key \( PK \); for every tuple \( t \in T \) there must be exactly one tuple \( r \in R \) such that \( t.FK = r.PK \). In this case, a binary indicator variable \( J_{t,r} \) can be added and the dependence between attributes across \( T \) and \( R \) is defined only for \( J_{t,r} = 1 \) as shown in top image of Figure 4. Then one can directly construct a BN based on the full outer join \( T \bowtie R \) as a PRM. The RI assumption is a very strong constraint, that normally does not hold in real-world DBs. Inspired by [24], we generalize this PRM learning algorithm using fanout attributes. We will discuss
side-by-side our generalized approach and the approach under RI assumption [17] in details in Section 4.2.

Sampling Cartesian product of relations: The last existing approach [57] to learn PRMs is to create a Cartesian product, \( C \), on all relations that can potentially be joined together and add a binary-valued join indicator \( J_{fr} \) for each join operation between two relations \( T, R \). The value of \( J_{fr} \) for a particular tuple in \( C \) is 1 if it corresponds to a tuple in the join \( T \bowtie R \). Let us reconsider the previous example in Equation 3, where \( Q = (T_j = t_j \land R_j = r_j) \). Thus, in this PRM, we can use the following equation to estimate the query’s cardinality.

\[
C(Q) = |T| \cdot |R| \cdot P_C(T_j = t_j, R_j = r_j, J_{fr} = 1)
\]

Apparently, building such \( C \) on all relations is prohibitive for any DB with reasonable size. Previous method [57] proposes to construct a PRM to estimate \( P_C \) on the Cartesian product \( C' \) created from i.i.d. samples from each relation. However, \( |C'| \) is still exponential with the number of relations in the database schema, requiring a very small sample size for each relation. Thus this method suffers from low predicating accuracy. Inspired by [61, 64], we build an efficient and unbiased version of sampling to assist our fanout attribute based PRM learning method, which will be explained in next subsection.

4.2 Fanout attributes method

Previous method [17] proposes a learning algorithm for PRM by adding binary join indicators but it is restricted to databases satisfying RI. Motivated by DeepDB [24] and FLAT [65], we replace the binary join indicator with a discrete fanout attribute to handle more general DBs. DeepDB’s method is theoretically sound only for primary-key→foreign-key joins and do not support left-join or right-join but later, extended by FLAT to support more general joins. We adapt the FLAT’s method in our BayesCard framework. Suppose that we would like to learn PRMs over a DB with two relations \( T \) and \( R \). A DB satisfying RI is shown in the top image in Figure 4. A more general DB of two relations is demonstrated in the bottom image, where the values of two join keys \( T.K \) and \( R.K \) are different. \( F_{T \rightarrow \Omega} \) indicates how many tuples in the join relation \( \Omega \) does a particular tuple in \( T \) fanouts to. Thus, we call it fanout attribute. We will use these two examples throughout this subsection.

To estimate the cardinality of a multi-relationship join query using a PRM or an ensemble of PRMs, the following three cases will be encountered: (1) there exists a PRM representing relations that exactly match the relations being queried, (2) several PRMs with each covering a subset of relations being queried exist and (3) a larger PRM that contains the relations being queried exists. The case (1) is trivial since we can directly calculate the cardinality using the learnt PRM.

Case (2). Combining smaller PRMs: Suppose that we have two PRMs for relations \( T \) and \( R \) respectively. In this case, we are automatically assuming that the attributes in \( T \) are independent with attributes in \( R \) because there is no way to capture the dependence across two PRMs learned separately. The modeling error could be introduced here since this assumption might not hold. Now we wish to answer the following query, that requests information from both \( T \) and \( R \).

**SELECT COUNT(*) FROM T, R**

**WHERE T.K = R.K AND T1 ≤ 20 AND R1 = 15**

**Referential integrity (RI) case:** First, let us consider two relations \( T, R \) satisfying RI (top images of Figure 4) and denote \( T \bowtie R \) as \( Q \). Then the probability of \( Q \) can be computed as:

\[
P_Q(T1 ≤ 20, R1 = 15) ≈ P_Q(T1 ≤ 20) \cdot P_Q(R1 = 15)
\]

\[
= P_Q(T1 ≤ 20, Jk = 1) \cdot P_Q(R1 = 15)
\]

\[
= P_T(T1 ≤ 20) \cdot P_R(Jk = 1) \cdot P_R(R1 = 15)
\]

(11)

The first row is derived based on the underlying independent assumption. The last row is evidenced by the RI assumption that every tuple in \( R \) must have exactly one match in \( S \).

**General case:** The idea is similar for general DBs without RI assumption. The only problem is that each tuple in \( T \) satisfying \( T1 ≤ 20 \) may appear several times in the outer join \( \Omega \) as in bottom image of Figure 4. Thus, we need to reweight the probability of \( P_T(T1 ≤ 20) \) in order to account for this effect. It is not hard to observe that for each tuple in \( T \) that satisfies \( T1 ≤ 20 \land F_{T \rightarrow \Omega} = k \), there will be \( k \) tuples in \( \Omega \) satisfying \( T1 ≤ 20 \land F_{T \rightarrow \Omega} = k \). Thus by law of total probability, we have the following equation:

\[
|\Omega| \cdot P_Q(T1 ≤ 20) = |\Omega| \cdot \sum_{k \in D(F_{T \rightarrow \Omega})} P_Q(T1 ≤ 20, F_{T \rightarrow \Omega} = k)
\]

\[
= |T| \cdot \sum_{k \in D(F_{T \rightarrow \Omega})} P_T(T1 ≤ 20, F_{T \rightarrow \Omega} = k) \cdot k
\]

\[
= |T| \cdot \mathbb{E}_{F_T} [P_T(T1 ≤ 20) \cdot F_{T \rightarrow \Omega}] (12)
\]

Therefore, we can derive the following equation for calculating probability of Query 10.

\[
P_Q(T1 ≤ 20, R1 = 15) ≈ P_Q(T1 ≤ 20) \cdot P_Q(R1 = 15)
\]

\[
= (\mathbb{E}_{F_T} [P_T(T1 ≤ 20) \cdot F_{T \rightarrow \Omega}] / |\Omega|) \cdot (\mathbb{E}_{F_R} [P_R(R1 = 15) \cdot F_{R \rightarrow \Omega}] / |\Omega|)
\]

(13)

We can pre-store the fanout attributes in relation \( T \) and \( R \) and the expected value can be calculated as efficient as the simple probability inference in our BayesCard implementations. Note that, sometimes we need to change \( F \) with value 0 to 1 during probability inference depending on the join type (i.e. inner join, left outer join, right outer join or full outer join). At last, we only need to calculate the size of the join \( \Omega \) in order to output the cardinality of join queries. For DBs satisfying referential integrity assumption, \( |\Omega| = |S| \) and in general DBs, \( |\Omega| = \sum_i^n F_{T \rightarrow \Omega}(i) = \sum_i^n F_{R \rightarrow \Omega}(i) \), i.e. summing the fanout variables over all tuples in \( T \) or \( R \).
In the case where we need to combine more than two smaller PRMs, we can combine two PRMs together at a time and apply Equation 13 recursively.

**Case (3): Extracting from a larger PRM.** Suppose that we have a single PRM learned on the full outer join \( \Omega \) of \( T \) and \( R \) but we wish to answer the following query on \( T \) only:

\[
\text{SELECT COUNT (*) FROM } T \text{ WHERE } T_1 \leq 20
\]

For \( T \) and \( R \) satisfying RI assumptions, estimation of this query can be trivially done since \( P_T(T_1 \leq 20) = P_\Omega(T_1 \leq 20) \) for \( k = 1 \).

For general DBs without RI assumption, tuples satisfying \( T_1 \leq 20 \) would appear several times in the join \( \Omega \), thus over-counted. Similar to case (2), we need to reweigh the probability \( P_\Omega(T_1 \leq 20) \) based on the fanout attributes, as in Equation 14:

\[
|T| \cdot P_T(T_1 \leq 20) = |T| \cdot \sum_{k \in \mathcal{D}(F_{T \rightarrow \Omega})} P_T(T_1 \leq 20, F_{T \rightarrow \Omega} = k) = |\Omega| \cdot \mathbb{E}_{F_T} P_\Omega(T_1 \leq 20 | F_{T \rightarrow \Omega}) \cdot \frac{1}{\max\{k, 1\}} \tag{14}
\]

In the case where the PRM is constructed on the full outer join of more than two relations, Equation 14 can be generalized according to \([24]\) and \([65]\).

**Efficient expectation computation:** With the benefits of PPL, BayesCard can compute the expectation \( E_F[P_T(Q|F) * F] \) as efficiently as computing the probability \( P_T(Q) \) for any query \( Q \). BayesCard can specify a particular elimination order during variable elimination (VE) inference algorithm as described in previous section, and choose the fanout variable \( F \) as the last variable to eliminate. Using PPL, the intermediate result after each elimination step is materialized as a distribution. Therefore, before the last elimination step of VE algorithm for computing \( P_T(Q) \), BayesCard can store the intermediate result, which represents the conditional distribution \( P_T(F|Q) \). Then, \( P_T(Q, F) \) can be derived from the Bayesian rule, \( P_T(Q, F) = P_T(Q|F) * P_T(Q) \). At last, the expectation \( E_F[P_T(Q|F) * F] = P_T(Q, F) \cdot \text{Dom}(F) \), where \( \cdot \) denotes the vector inner product and \( \text{Dom}(F) \) denotes the vector of all distinct values of \( F \).

### 4.3 Model construction for fanout attributes method

Given a DB schema containing two relations \( T \) and \( R \) as in the previous example, one can learn two PRMs for \( T \) and \( R \) separately. Thus attributes from different relations are automatically assumed to be independent and potential modeling errors might appear. Alternatively, one can construct a larger PRM over the full outer join \( \Omega = T \bowtie R \), which is more accurate but less efficient in both model construction and probability inference. Thus, deciding which ensembles of PRMs to learn is also a non-trivial task.

Previous approach \([24]\) suggests to first create every possible two relation join results and then compute the level of dependence between attributes across the two. A large PRM will be learnt directly on the outer join if there is a relatively high correlation between attributes of the two relations. Otherwise, one small PRM will be learnt for each relation respectively. This process will carry on recursively if it considers learning a PRM for more than two relations. However, generating the full outer join of multiple relations can require exponential memory in the worst case. Thus, this approach \([24]\) normally cannot explore the possibility of creating a PRM on the join of more than three relations.

**Sampling outer join results:** Computing the exact join of multiple relations is sometimes infeasible so if we want to explore larger and more accurate PRMs, generating unbiased samples of the join results is crucial. A naive approach would be taking random samples from each relation and calculate the join on the sampled relations. Previous work on PRMs \([57]\) used a modified version of this method but sampled join result using this approach is biased \([7]\). Recent work \([64]\) provides an efficient sampling algorithm that produces unbiased samples of the join result. We adopt this unbiased sampling algorithm to generate samples on the full outer join of all relations in the schema and use these samples to guide our PRM ensemble construction. We denote these samples as \( S \).

One approach suggested by NeuroCard method \([61]\) is to learn a single large model on \( S \), which contains all attributes and the added fanout attributes of all relations. However, this approach is not optimal for two reasons. 1) The resulting single PRM contains extensive amount of attributes so the model construction can be costly or sometimes intractable and the probability inference in the learnt model is redundant and very inefficient. 2) For DBs with a large number of relations, the size of \( S \) is relatively small with respect to the full outer join size. Thus, there is a large amount of information loss so the learnt model might not accurately represent the data distribution. However, the attributes dependence level can be easily calculated on relative small samples \([35]\), so this sampled join \( S \) only serves this purpose in our PRM ensemble construction.

**PRMs ensemble construction with budget:** In order to balance between probability inference efficiency and accuracy, we want to explore the full possibility of learning PRM ensembles but at the same time constrain the maximum number of joined relations to construct a single PRM upon. Therefore, the resulting ensemble should capture as much dependence between relations as possible and simultaneously keep each PRM in this ensemble as small as possible. Here we define the budget \( k \) such that a single PRM can be constructed on the full outer join of no more than \( k \) relations. Note that budget \( k \) is a hyper-parameter that varies with specific DBs and the computing resources.

Here we consider a DB schema containing \( 11 \) relations \( \{A, \cdots, K\} \), as shown in Image (a) of Figure 5, where each node represents a relation and each edge represents a possible join between two relations. Here we restrict the schema graph to a tree since a cycle will result in an infinite loop of joins \([64]\). In our Algorithm 1, given a

![Figure 5: PRM Ensemble learning algorithm demonstration](image)
We compare our BayesCard framework with the following set of cardinality estimation methods. The first two methods are widely used in practical database management systems (DBMS) and the last three are the current state-of-the-art methods using deep learning.

1) Histogram adopts the attributes independence and the join uniformity assumptions. It is one of the simplest cardinality estimation method with small model size and fast query inference speed. This method is widely used in DBMS such as PostgreSQL [19].

2) Sampling uniformly samples tuples during execution and uses the calculated cardinality on the samples as an estimation for the true cardinality. We set the sampling rate to 0.1% and 1% of all datasets in our experiments. This method also has extensive applications in DBMS such as MySQL [44] and MariaDB [53].

3) Naru is a framework of deep auto-regressive models for cardinality estimation [62]. For our single relation experiment, we adopt the source code and the DNN structure proposed in their original paper. Naru does not support cardinality estimation of multi-relations join queries. Later, the authors of Naru proposes a new framework [61] by extending Naru to multi-relational settings. This framework is published very recently and has not open source yet, so we can not compare it with our BayesCard.

4) DeepDB is fundamentally based on sum-product networks [24]. It achieves the state-of-the-art performance on the join-order benchmark of IMDB dataset [31]. We adopt the authors’ source code and apply the same hyper-parameters as used in the original paper.

5) MSCN is the state-of-the-art supervised method for cardinality estimation for multi-relation join queries. Its performance on single relations is not as good; hence, we only compare with this method on IMDB join order benchmark. We use the authors’ source code and apply the exact same settings to reproduce the result on IMDB join order benchmark.

In addition, we reproduce the previous methods on BNs/PRMs using our BayesCard. [17] uses a greedy algorithm for structure learning and variable elimination (VE) inference algorithms. [57] converts a learned BN into a junction tree (JT) structure and used the belief propagation (BP) inference algorithms which is an exact inference for junction tree. [21] learns a chow-liu tree (CLT) BN and uses the VE inference algorithms.

### 5.1 Experimental setups

**Methods to compare with:** We compare our BayesCard framework with the following set of cardinality estimation methods. The first two methods are widely used in practical database management systems (DBMS) and the last three are the current state-of-the-art methods using deep learning.

1) **Histogram** adopts the attributes independence and the join uniformity assumptions. It is one of the simplest cardinality estimation method with small model size and fast query inference speed. This method is widely used in DBMS such as PostgreSQL [19].

2) **Sampling** uniformly samples tuples during execution and uses the calculated cardinality on the samples as an estimation for the true cardinality. We set the sampling rate to 0.1% and 1% of all datasets in our experiments. This method also has extensive applications in DBMS such as MySQL [44] and MariaDB [53].

3) **Naru** is a framework of deep auto-regressive models for cardinality estimation [62]. For our single relation experiment, we adopt the source code and the DNN structure proposed in their original paper. Naru does not support cardinality estimation of multi-relations join queries. Later, the authors of Naru proposes a new framework [61] by extending Naru to multi-relational settings. This framework is published very recently and has not open source yet, so we can not compare it with our BayesCard.

4) **DeepDB** is fundamentally based on sum-product networks [24]. It achieves the state-of-the-art performance on the join-order benchmark of IMDB dataset [31]. We adopt the authors’ source code and apply the same hyper-parameters as used in the original paper.

5) **MSCN** is the state-of-the-art supervised method for cardinality estimation for multi-relation join queries. Its performance on single relations is not as good; hence, we only compare with this method on IMDB join order benchmark. We use the authors’ source code and apply the exact same settings to reproduce the result on IMDB join order benchmark.

In addition, we reproduce the previous methods on BNs/PRMs using our BayesCard. [17] uses a greedy algorithm for structure learning and variable elimination (VE) inference algorithms. [57] converts a learned BN into a junction tree (JT) structure and used the belief propagation (BP) inference algorithms which is an exact inference for junction tree. [21] learns a chow-liu tree (CLT) BN and uses the VE inference algorithms.

### 5.2 Experimental datasets and query workloads

Our single relation experiments are performed on two datasets: DMV [43] and CENSUS [18], summarized in top two rows of Table 2.

1) **DMV** dataset is a real-world dataset consisting of vehicle registration information in New York. We use the same attributes with widely differing data types and domain sizes as in [62]. Our snapshot contains 11,575,483 tuples. This dataset has been used by [62] to demonstrate their state-of-the-art results.

2) **CENSUS** dataset contains population survey by U.S. Census Bureau conducted in 1990. This dataset has 2,458,285 tuples and 68 attributes, containing highly correlated attributes. Based on RDC test [35], we find that more half of the attributes are highly

| Dataset  | #Relations | #Attributes | #Tuples in full join | Size |
|----------|------------|-------------|----------------------|------|
| DMV      | 1          | 11          | $10^9$               | 1GB  |
| CENSUS   | 1          | 68          | $2 \times 10^8$      | 361MB|
| IMDB JOB | 6          | 8           | $6 \times 10^4$      | 2GB  |
|          |            |             |                      | 1926MB|

**Table 2: Datasets summary.**
the variance in model prediction. Thus, we synthesize 1500 JOB-comp queries based on the schema of JOB-light with more number of filter predicates per query. Each JOB-comp query involves 4-6 relations with 2-7 filter predicates. The queries are uniformly distributed to each join of 4-6 relations. After determining the join graph, the filter predicates selection process is similar as in single relation cases.

**Experimental environment:** All models are trained on Nvidia V100 GPU and all query probability inference of the learned models take place on Intel(R) Xeon(R) Platinum 8163 CPU with 64 cores for fair comparison. Apart from the DAR model (Naru), the rest methods’ inference algorithms are not optimized for GPU.

**Evaluation metric:** We use the Q-error as our evaluation metrics, which is defined as follow:

\[ \text{Q-error} = \max(\frac{\text{Estimated Cardinality}}{\text{True Cardinality}}, \frac{\text{True Cardinality}}{\text{Estimated Cardinality}}) \]

This evaluation metric is well recognized in DBMS community and widely used in recent papers on cardinality estimation [17, 24, 57, 61, 62]. We report the 50% (median), 90%, 99% and 100% (worst) Q-error quantiles as evaluation of estimation accuracy.

### 5.2 Estimation accuracy and inference latency

The estimation performance with respect to single relations is reported in Table 3 and the performance of multi-relational IMDB with two query workloads is reported in Table 4. We have the following observations with respect to Table 3 and Table 4.

1. **Comparing different BNs/PRMs within BayesCard:**

   **Single relation (Table 3):** We do not report the result of BNs’ sampling inference algorithm because it is slow and inaccurate simultaneously. More importantly, it is not very useful in DB settings, since one can directly fetch samples from the original data to performance cardinality estimation, which can be more efficient and accurate. For DMV dataset, all BN structure learning methods produce similar performance in accuracy because this dataset contains limited number of attributes and the data structure is not very complicated. Whereas for CENSUS dataset, the accuracy difference between different BNs is significant but the total number of attributes are so large that exact learning algorithm in general would not work for this dataset.

   From the latency results of first two rows for each datasets, we can see that our graph reduction algorithm roughly increases the inference speed by 2 and 4 times for DMV and CENSUS dataset, respectively. Overall, we can see that CLT structured BN with VE inference algorithm achieves the best latency on both datasets.

   **Multi-relation IMDB:** We conduct the multi-relational experiment on international movie database (IMDB) benchmark, illustrated in the last row of Table 2. Prior work [31] claims that this DB contains complicated data structure and establishes it to be a good test benchmark for cardinality estimators. We use JOB-light benchmark query workload with 70 queries proposed in the original paper [31] and create another workload of 1500 JOB-comp with more comprehensive and complicated queries.

   JOB-light’s IMDB schema contains six relations (title, cast_info, movie_info, movie_companies, movie_keyword, movie_info_idx) and five join operations in total where every other relation can only join with the primary relation “title”. Each JOB-light query involves 3-6 relations with 1-4 filter predicates. The filter variety is not very diverse with equality filters on all attributes but the “title.production_year” attribute only. In addition, JOB-light’s workload only contains 70 queries, which is not enough to account for

---

Table 3: Performance of cardinality estimation algorithms on single relations. Please note that we do not show exact structure learning for CENSUS data as it contains too many attributes and infeasible for exact structure learning, explained in Section 3.2. Note 1. OG stands for original graph, meaning that the VE is performed on the original graph instead of the reduced graph.

| Dataset | Method | Algorithm | 50% | 90% | 95% | 100% | Latency (ms) |
|---------|--------|-----------|-----|-----|-----|------|-------------|
| DMV     | BayesCard | CLT-VE-OG | 1.002 1.115 1.216 23.31 | 182 | 87 | 105 | 328 |
|         |        | CLT-VE   | 1.002 1.115 1.216 23.31 | 182 | 87 | 105 | 328 |
|         |        | CLT-BP   | 1.002 1.115 1.216 23.31 | 182 | 87 | 105 | 328 |
|         |        | Greedy-VE | 1.002 1.115 1.349 36.25 | 328 | 86 | 416 | 146 |
|         |        | Greedy-BP | 1.003 1.213 1.411 35.00 | 86 | 86 | 86 | 86 |
|         |        | Exact-VE  | 1.002 1.116 1.229 24.00 | 416 | 416 | 416 | 416 |
|         |        | Exact-BP  | 1.181 1.242 2.550 55.70 | 146 | 146 | 146 | 146 |
|         |        | JT-VE     | 1.003 1.163 1.310 25.00 | 456 | 456 | 456 | 456 |
|         |        | JT-BP     | 1.003 1.163 1.310 25.00 | 456 | 456 | 456 | 456 |

| CENSUS  | Deep models | DeepDB | 1.002 1.034 1.097 7.950 | 56 | 56 |
|---------|-------------|--------|-----------------------|----|----|
|         |             |        | Sampling 0.1%          | 198 | 5  |
|         |             |        | Sampling 1%            | 140 | 5  |

| Dataset | Method | Algorithm | 50% | 90% | 95% | 100% | Latency (ms) |
|---------|--------|-----------|-----|-----|-----|------|-------------|
| DMV     | BayesCard | CLT-VE-OG | 1.063 1.367 2.002 235.0 | 780 | 780 | 780 | 328 |
|         |        | CLT-VE   | 1.063 1.367 2.002 235.0 | 780 | 780 | 780 | 328 |
|         |        | CLT-BP   | 1.063 1.367 2.002 235.0 | 780 | 780 | 780 | 328 |
|         |        | Greedy-VE | 1.082 1.293 1.872 175.1 | 1259 | 1259 | 1259 | 1259 |
|         |        | Greedy-BP | 1.105 1.645 3.132 573.8 | 372 | 372 | 372 | 372 |
|         |        | JT-VE    | 1.091 1.261 2.552 325.3 | 1556 | 1556 | 1556 | 1556 |
|         |        | JT-BP    | 1.091 1.261 2.552 325.3 | 1556 | 1556 | 1556 | 1556 |

| CENSUS  | Deep models | NARU | 1.129 1.830 5.156 795.0 | 58 | 58 |
|---------|-------------|------|-----------------------|----|----|
|         |             |      | Sampling 1%            | 193 | 5  |
|         |             |      | Sampling 1%            | 1703 | 5  |

---

Ziniu Wu and Amir Shaikhha
queries, the estimation error gap has very little influence on generating high-quality query plans [31]. Admittedly, BayesCard models is much slower in inference latency than DeepDB and some BNs’ inference speeds in BayesCard are comparable to Naru. As for multi-relational IMDB, BayesCard’s performance in accuracy on JOB-light is comparable with the current state-of-the-art model DeepDB. The results on the harder and more comprehensive queries JOB-comp demonstrate BayesCard’s better estimation accuracy than DeepDB. Overall, the inference speed of the deep models are 2-4 times faster than BayesCard for multi-relational cases.

One of the key advantages of PPLs is the usage of compiler optimizations [40]. Inspired by the impressive results of compilation for query processing [42, 54, 55], we plan to investigate the usage of just-in-time compilation (JIT) and compiler optimizations to improve inference latency.

5.3 Scalability, stability and interpretability

From the Section 5.2, we can see that BayesCard models have comparable performance with the current state-of-the-art deep probabilistic cardinality estimation methods. In this section, we will show BayesCard’s significant advantages over these methods in terms of model scalability, performance stability and learned structure interpretability.

Model scalability: Model storage size and training time are two key components of model scalability and their importance to DBMS is illustrated in previous researches [13, 14]. We report the model size and training time in Figure 6. Overall, we can see that CLT structured BN/PRM has the smallest model size and fastest training time among other models in BayesCard. The two most commonly used methods in commercial DBMS, Histogram and Sampling methods, have the best scalability of all. Sampling is a model free method and thus does not need any training or extra storage. Histogram uses the strongest attributes independence assumptions to minimize the model size and training time.

When compared with the deep probabilistic models, BayesCard models have much faster training speed than SPN and Naru. The BayesCard model with best scalability, CLT, is roughly 100 times faster to train than SPN and 1000 times faster than Naru and MSCN on single relations and multi-relational DB, respectively. In addition, BNs have significantly smaller model size than all deep probabilistic methods. The average model sizes for BNs is 10 times smaller than all deep probabilistic models as shown in Table 4. BayesCard models and the blue points are deep probabilistic models and Histogram.

Thus, there are more computations needed than the uniformity assumptions. The exact structure learning algorithm has some gain in accuracy compared with CLT algorithm. However, the probability inference time of exact structure learning algorithm is much slower.

2). Comparing BayesCard with traditional cardinality estimation methods: For all three datasets, the Histogram method is significantly faster than all BayesCard models but with much worse estimation accuracies. On single relations (Table 3), BayesCard models can achieve slightly better accuracy than Sampling method with 1% samples but in general faster in inference. Besides, BayesCard models’ estimation accuracies are much higher than Sampling with 0.1% samples with comparable inference speed. On multi-relational DBs (Table 4), BayesCard models even with uniformity assumptions significantly outperform Sampling methods in terms of both accuracy and latency. BayesCard models with fanout join estimation method can outperform Sampling method by an order of 1-3 magnitude in accuracy with comparable inference latency.

3). Comparing BayesCard with deep probabilistic models:

On a single relation, the detailed comparison of estimation accuracy grouped by the query selectivity intervals between an exemplary BN (CLT) and deep models can be found in Figure 7. We can see that BN is significantly better than DeepDB in all intervals for both datasets. For queries with high and medium selectivities, BN has comparable estimation accuracy as Naru on DMV dataset but significantly better than it on CENSUS. Naru has a better accuracy on queries with low selectivity on DMV dataset. However, for these
For commercial query optimizers, interpretability is also an important factor because interpretable models are easier to maintain, validate and improve with some heuristics or expert knowledge.

Stability of traditional methods: Histogram and Sampling, do not produce consistent results. As shown in Table 3, the predicting accuracy for Histogram drops tremendously from DMV to CENSUS because CENSUS dataset contains a large amount of highly correlated attributes so the independence assumption in Histogram is significantly violated. For more complicated dataset, Sampling methods would require a larger sampling rate resulting in slower probability inference speeds and lower accuracy in CENSUS dataset.

Comparing BayesCard methods with deep models: DMV dataset only contains a set of three high correlated attributes (Body type, registration class and fuel type) and yet DeepDB, which is based on SPN, cannot model them as accurate as BayesCard. CENSUS dataset contains a large amount of highly correlated attributes as explained earlier. Thus during training, we find that the SPN in DeepDB cannot make most of these pairs of attributes independent even with very deep structure. Therefore, its estimation accuracy q-error is extremely high when compared to BayesCard.

Naru is more stable than DeepDB in single relation experiments but we can see a significant drop in accuracy from DMV to CENSUS. Auto-regressive models do not use any attributes independence or conditional independence properties. Thus the learning space for Naru grows exponentially with the number of attributes in the dataset so Naru’s predicting accuracy on a larger dataset CENSUS is much worse than on DMV.

Furthermore, a detailed comparison between BayesCard and the current state-of-the-art model DeepDB is shown in Figure 8. Specifically, we group the queries in JOB-comp by the number of join tables and number of filter predicates and compare the estimation accuracy in each group between DeepDB and BayesCard’s models (as in Table 4 JOB-comp). Note that we do not compare with other methods in Table 4 because there is a significant accuracy difference. In addition, DeepDB has demonstrated their superiority over MSCN on a similar experiment [24].

We can see that BayesCard’s prediction accuracy is slightly better than DeepDB on queries with small amount of join tables and filter predicates. The accuracy gap gradually becomes significant as the number of joined tables and filter predicates increases. On queries covering all tables with more than five filter predicates, BayesCard’s accuracy is 2-4 times higher than DeepDB, suggesting that BayesCard’s performance is more stable for more comprehensive and complicated queries.

Interpretability of learned model structure: For commercial query optimizers, interpretability is also an important factor because interpretable models are easier to validate and improve with some heuristics or expert knowledge.

Sampling is a model-free method so there is no such concept of a learned model structure for Sampling method. Another standard cardinality estimation method, Histogram, assumes that all attributes are independent with each other. Thus the learned model structure of Histogram method do not convey any information about the attributes correlation and provide very little space for users to validate or improve the existing model structure. Interpretation in deep neural networks, for example Naru and MSCN, is in general a very hard question as they serve as black-box approximators for the joint data distribution [6]. The learned SPN model itself in DeepDB method is very hard to interpret as it involves splitting original dataset into smaller sub-datasets in order to achieve attribute independence locally, as stated by Peharz et al. [46].

Since a BN defines a directed acyclic graph to represent the attributes dependence in the data distribution, users can easily validate or improve the learned BN model structure. The previous example in Figure 2 of Section 3.3, demonstrates the learned BN structure on a subset of CENSUS data. Based on prior information, one can easily suspect the dependence between “Age” and “Sex” in the tree-structured BN, which should not exist in US population demographics. In addition, the number of children raised by someone should be dependent on his/her income but this dependence is not present in the tree-structured BN. With this information, users can train a more complicated DAG-structured BN. Alternatively, BayesCard supports adding or deleting edges from an existing BN graph structure so users can directly remove the edge connecting “Sex” and “Age”, add the edge from “Income” to “Child” and retrain the parameters of the new BN. Moreover, users of BayesCard can specify certain edges before learning BN based on some known dependence. In this case, the search space of BN is restricted and the learning algorithm is more efficient and accurate simultaneously. Thanks to the interpretability of BN’s model structure, such heuristics based on prior information and expert knowledge can be easily used on BNs but not on any other models.

5.4 Take-away messages for using BayesCard

Based on cardinality estimation performance, model scalability, estimation stability and structure interpretability, we summarize the following take-away messages for using BayesCard.

- CLT algorithm is the fastest algorithm for model construction, producing the smallest model with efficient probability inference.
- The exact structure learning algorithm can take exponential time and space. Hence, it is not optimized for datasets with a large amount of attributes. The greedy structure is slightly more accurate than CLT, but it is much slower in probability inference.
- Since a learned BN structure provides interpretation about the data distribution, a good initial step for a new dataset would be to construct a tree-structured BN using the CLT algorithm. Then users can examine the learned BN’s structure against their prior knowledge of the dataset. In the end, users can decide to learn more complicated structured BNs from scratch or improve the existing BN using the aforementioned approach.
- For multi-relational settings, there is a trade-off between accuracy and estimation time. Our extended fanout method has better accuracy in the expense of slower estimation, whereas the uniformity method sacrifices accuracy for faster estimation.
6 CONCLUSION

This paper makes the first systematic comparison between BNs and deep probabilistic models in the context of cardinality estimation. We design BayesCard, which is the first framework that unifies the existing efforts on using BNs for cardinality estimation. This framework incorporates various techniques inside PPLs for building compact and accurate BNs. BayesCard is extended to the multi-relational setting and can accurately estimate the cardinality of join queries. Users of BayesCard can easily build an ensemble of compact BNs optimized for their target database with the flexibility of making a trade-off between accuracy and estimation time. Admittedly, the deep probabilistic models have slightly faster probability inference speed; hence, we would like to further improve the inference speed of BNs/PRMs as a future research.

REFERENCES

[1] Mert Aldere and Ugur Cetintemel. 2012. Learning-based query performance modeling and prediction. ICDE (2012).
[2] Ankur Ankan and Abinash Panda. 2015. gppmp: Probabilistic graphical models using python. In Proceedings of the 14th Python in Science Conference (SCIYP 2015). Citeseer.
[3] Anne Berry, Jean R. S. Blair, and Pinar Heggernes. 2004. Maximum cardinality search for computing minimal triangulations of graphs. Algorithmica (2004).
[4] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neel Pradhan, Theofanis Karakatsoulis, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. 2019. Pyro: deep universal probabilistic programming. Journal of Machine learning research (2019).
[5] Remco R. Bouckaert. 1993. Probabilistic network construction using the minimum description length principle. ECQLU (1993).
[6] Supriyo Chakraborty, Richard Tomsett, Ramya Raghavendra, Daniel Harborne, Mostafa Alzantot, Federico Cerutti, Sriramvastavam Mani, Alun Pricee, Simon Julier, Raghuvire M. Rao, Troy D. Kelley, Dave Braines, Murat Sensoy, Christofer J. Willis, and Prudhvi Gurrum. 2017. Interpretability of deep learning models: A survey of results. In 2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIS/ATC/CBDCom/IOPSCIC), 1–6.
[7] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 1999. On Random Sampling over Joins. ACM SIGMOD International Conference on Management of Data (1999).
[8] David Maxwell Chickering. 1996. Learning Bayesian networks is NP-complete. In Learning from Data: Artificial Intelligence and Statistics (1996).
[9] David Maxwell Chickering. 2002. Optimal structure identification with greedy search. JMLR (2002).
[10] David Maxwell Chickering and David Heckerman. 1997. Efficient approximations for the marginal likelihood of Bayesian networks with hidden variables. ML 29 (1997).
[11] C. K. Chow and C. N. Liu. 1968. Approximating discrete probability distributions with dependence trees. IEEE transactions on Information Theory (1968).
[12] Philip Dawid, Robert G. Cowell, and Steffen Lauritzen. 1999. Probabilistic Networks and Expert Systems: Exact Computational Methods for Bayesian Networks. Springer.
[13] Anshuman Dutt, Chi Wang, Vivek Narasayya, and Surajit Chaudhuri. 2020. Efficiently Approximating Selectivity Functions using Low Overhead Regression Models. VLDB (2020).
[14] Anshuman Dutt, Chi Wang, Azade Naz, Srikanth Kandula, Vivek Narasayya, and Surajit Chaudhuri. 2019. Selectivity Estimation for Range Predicates using Lightweight Models. VLDB (2019).
[15] Fei Fu and Qing Zhou. 2013. Learning sparse causal Gaussian networks with experimental intervention: regularization and coordinate descent. JASA (2013).
[16] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. 2015. MADE: Masked autoregressive density estimation. International Conference on Machine Learning (2015), 881–889.
[17] Lise Getoor and Daphne Taskar, Ben and Koller. 2001. Selectivity estimation using probabilistic models. SIGMOD (2001).
[18] US Government. 1990. Current Population Survey of U.S. Census Bureau using their Data Extraction System. (1990).
[19] The PostgreSQL Global Development Group. 2018. Documentation PostgreSQL 10.3.
[20] Marco Grzegorczyk. 2010. An introduction to Gaussian Bayesian networks. Systems Biology in Drug Discovery and Development. (2010), 121–147.
[21] Max Halford, Philippe Saint-Pierre, and Franck Morvan. 2019. An approach based on bayesian networks for query selectivity estimation. DaSFAA 2 (2019).
[22] Shohei Kato, Fumihiko Tsuru, and Masaru Yoshitomi. 2016. Deep Learning, Models for Selectivity Estimation of Multi-Attribute Queries. SIGMOD (2020).
[23] David Heckerman, David Geiger, and David Maxwell Chickering. 1995. Learning Bayesian networks: The combination of knowledge and statistical data. ML 20 (1995).
[24] Ben Benjamin, Andreas Schmidt, Moritz Kulesa, Alessandro Molina, Kersting Kristian, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries’ VLDB (2020).
[25] Finn V. Jensen and Frank Jensen. 1994. Optimal junction trees. UAI (1994).
[26] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfonso Kemper. 2019. Learned Cardinalities: Estimating correlated joins with deep learning. CIDR (2019).
[27] Dophan Koller and Nir Friedman. 2009. Probabilistic Graphical Models Principles and Techniques. MIT Press.
[28] Daphne Koller and Ari Pfeffer. 1998. Probabilistic framework-based systems. AAAI (1998).
[29] Tejas D. Kulkarni, Pushmeet Kohli, Joshua B Tenenbaum, and Vikash Mansinghka. 2015. Picture: A probabilistic programming language for scene perception. In Proceedings of the ieee conference on computer vision and pattern recognition. 4390–4399.
[30] Seetha Lakshmi and Shaoyu Zhou. 1998. Selectivity estimation in extensible databases – A neural network. VLDB (1998).
[31] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfonso Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? Proc. VLDB Endow. 9 (2015), 204–215.
[32] Viktor Leis, Bernhard Radke, Andrey Gubichev, Alfonso Kemper, and Thomas Neumann. 2017. Cardinality estimation done right: Index-based join sampling. CIDR (2017).
[33] Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfonso Kemper, and Thomas Neumann. 2018. Query optimization through the looking glass, and what we found running the join benchmark. The VLDB Journal (2018).
[34] Guy Lohman. 2014. Is query optimization a solved problem? SIGMOD (2014).
[35] David Lopez-Paz, Philipp Hennig, and Bernhard Schölkopf. 2013. The randomized dependence coefficient. In NIPS 4–9.
[36] Jaakko Ountonen. 2015. BayesPy: Variational Bayesian Inference in Python. arXiv:1410.0870 (2015).
[37] Tanu Malik, Randal Burns, and Nitesh Chawla. 2007. A black-box approach to query cardinality estimation. CIDR (2007).
[38] James Martens and Venkatesh Medabalimi. 2014. On the expressive efficiency of sum product networks. arXiv preprint arXiv:1411.7717 (2014).
[39] Yoss Matias, Jeffery Scott Vitter, and Min Wang. 1998. Wavelet-based histograms for selectivity estimation. SIGMOD (1998).
[40] T. Minka, J.M. Winn, J.P. Guiver, Y. Zaykov, D. Fabian, and J. Bronskill. 2018. /infer.NET 0.3. Microsoft Research Cambridge. http://dotnet.github.io/infer/.
[41] Charlie Nash and Connor Durkan. 2019. Autogressive energy machines. ICML (2019).
[42] Thomas Neumann. 2011. Efficiently Compiling Efficient Query Plans for Modern Hardware. PVLDB 4, 9 (2011), 539–550.
[43] State of New York. 2020. Vehicle, snowmobile, and boat registrations. https://catalog.data.gov/dataset/vehicle-snowmobile-and-boat-registrations (2020).
[44] Corporation Oracle. 2020. MySQL 8.0 Reference Manual.
[45] Judea Pearl. 1988. Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann (1988).
[46] Robert Peharz, Robert Genz, Franz Pernkopf, and Pedro Domingos. 2017. On the Latent Variable Interpretation in Sum-Product Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence 39, 10 (2017), 2030–2044.
[47] Hofiung Poon and Pedro Domingos. 2011. Sum-product networks: A New Deep Architecture. IEEE International Conference on Computer Vision Workshops (2011).
[48] Robert W. Robinson. 1977. Counting unlabeled acyclic digraphs. In Combinatorial Mathematics (1977).
[49] John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. 2016. Probabilistic programming in Python using PyMC3. PeerJ Computer Science (2016). https://doi.org/10.7717/peerj-cs.55.
[50] Mauro Scagnagatta, Giorgio Corani, Cassio P De Campos, and Marco Zaffalon. 2016. Optimal structure identification with greedy search. NeurIPS (2016).
[51] Jacob Schreiber. 2018. Pomegranate: fast and flexible probabilistic modeling in python. Journal of Machine Learning Research 19, 164 (2018), 1–6.
[52] P.G. Selinger, M. M. Atrashan, D. D. Chamberlin, R. A. Lorie, and T.G. Price. 1979. Access path selection in a relational database management system. ACM SIGMOD International Conference on Management of Data (1979).
[53] MariaDB Server Documentation. 2020. Statistics for optimizing queries: InnoDB persistent statistics. https://mariaodb.com/kb/en/innodb-persistent-statistics/ (2020).
[54] Amir Shakhkha, Yannis Klonatos, and Christoph Koch. 2018. Building Efficient Query Engines in a High-Level Language. ACM Transactions on Database Systems.
Alternatively, we can derive the following Equation 16 by law of total probability and conditional independence assumption.

\[
P_T(T'_1 = t'_1, \ldots, T'_K = t'_K) = \sum_{T^*_1 \in \text{Parents}(T'_1)_{1 \leq j \leq k}} \ldots \sum_{T^*_n \in \text{Parents}(T'_n)_{1 \leq j \leq k}} \prod_{T'_j \in V/QV} P_T(T'_j = t'_j \mid \text{Parents}(T'_j))^* \\
= \sum_{T^*_1 \in \text{Parents}(T'_1)_{1 \leq j \leq k}} \ldots \sum_{T^*_n \in \text{Parents}(T'_n)_{1 \leq j \leq k}} \prod_{T'_j \in V/QV} P_T(T'_j = t'_j \mid \text{Parents}(T'_j))^* \\
\prod_{T^*_j \in V/QV} P_T(T'_j = t'_j \mid \text{Parents}(T'_j))^*
\]

where \(\text{Parents}(T''_i)\) denotes the parents of node \(T''_i\) in graph \(G\), which is the same as parents of node \(T''_i\) in graph \(G'\). By definition of reduced graph \(G'\) where \(V' = \bigcup_{1 \leq j \leq k} \text{Ancestor}(T'_j)\). \(\text{Ancestor}(T'_j)\) includes all parent nodes of \(T'_j\) and all parents of parent node recursively. Let \(|V'| = n'\) and \(V' \cap QV = T'_1, \ldots, T'_{n''-k}\). We can recursively write out \(P_T(\bigcup \text{Parents}(T'_{1 \leq j \leq k}))\) using Equation 16 and result in Equation 17.

\[
P_T(T'_1 = t'_1, \ldots, T'_K = t'_K) = \sum_{T^*_1 \in \text{Parents}(T'_1)} \ldots \sum_{T^*_n \in \text{Parents}(T'_n)} \prod_{T'_j \in QV} P_T(T'_j = t'_j \mid \text{Parents}(T'_j))^* \\
\prod_{T^*_j \in V/QV} P_T(T'_j = t'_j \mid \text{Parents}(T'_j))^*
\]

Equation 17 has the same form as Equation 15 with less attributes in the summation. Thus the VE algorithm [27] can compute Equation 17 by eliminating one attribute from \(V' \cap QV\) at a time. Thus running VE on \(G\) is equivalent to running VE on \(G'\).

B COMPUTING THE DEPENDENCE LEVEL BETWEEN RELATIONS

We use the randomized dependence coefficient (RDC) [35] as a measure of dependence level between two attributes. RDC is invariant with respect to marginal distribution transformations and has low computational cost and it is widely used in many statistical methods [24, 47]. The complexity of RDC is roughly \(O(n + \log(n))\) where \(n\) is the sample size for the two attributes.

B.1 Calculating the pairwise RDC score between two relations

Recall Figure 9, we have a DB schema with 11 relations \(A, \ldots, K\) and their join relations are defined as a tree \(T\) on the left image. In addition, we have unbiased samples \(S\) of the full outer join of all relations in \(T\) using the previously mentioned approach [64]. Now
BayesCard: A Unified Bayesian Framework for Cardinality Estimation

Figure 9: PRM Ensemble learning algorithm demonstration

Consider $T, R \in A, \ldots, K$ as two random relations in this schema with attributes $T_1, \ldots, T_n$ and $R_1, \ldots, R_m$ respectively. We can compute the pairwise RDC score between attributes $T_i$ and $R_j$, $RDC_{ij}$ based on $S$, as described in [35]. Then we take the average as the level of dependence between $T$ and $R$ in the following Equation 18.

$$\sum_{1 \leq i \leq n} \sum_{1 \leq j \leq m} RDC_{ij} / (n \times m)$$

Thus, we can compute the dependence level matrix $M$ of size $11 \times 11$ with each entry specifying the dependence level between two relations in the schema. Then the edge weights of original $T$ on the left image can be directly taken from $M$. The complexity of calculating $M$ is thus $O(m^2 \times |S| \times \log(|S|))$ where $m$ is the total number attributes in all relations.

B.2 Calculating the pairwise RDC score between two set of relations

During the PRM ensemble construction procedure, we sometimes need to calculate the dependence level between two sets of relations, such as the dependence level of $A, D$ and $H, K$ as in the right image of Figure 9. Similarly to the previous cases in Section B.1, this value can be directly computed from $M$.

Take $Attr(T)$ denotes the set of attributes in relation $T$. Same as Equation 18, the level of dependence between $A, D$ and $H, K$ is defined as Equation 19.

$$\sum_{a \in Attr(A)} \sum_{h \in Attr(H)} RDC_{ah} / \left( |Attr(A) + Attr(D)| \times |Attr(H) + Attr(K)| \right)$$

$$\sum_{a \in Attr(A)} \sum_{k \in Attr(K)} RDC_{ak} + \sum_{d \in Attr(D)} \sum_{h \in Attr(H)} RDC_{dh} + \sum_{d \in Attr(D)} \sum_{k \in Attr(K)} RDC_{dk}$$

Thus the weight of the edge can be updated quickly knowing the pre-computed $M$ and the number of attributes in each relation.