A EQUATIONS AND PROOFS

A.1 Equations for decomposing joins into single-tables

In this section, we provide the case-by-case equations on how to accurately formulate join estimation problem into single-table CardEst problem without any simplified assumptions.

Two-table join case:

Assume that we have two tables A and B with join keys $A.id$ and $B.Aid$ and a query $Q = Q(A) \bowtie Q(B)$ where $Q(A)$ is the filter on table A and same for $Q(B)$. The join condition is $A.id = B.Aid$. Then, the join cardinality of Q can be expressed as follows where $D(A.id)$ represents the domain of A.

$$|Q| = \sum_{v \in D(A.id)} P_A(A.id = v|Q(A)) * |Q(A)| * P_B(B.Aid = v|Q(B)) * |Q(B)| \tag{8}$$

Chain join cases:

Case1 A.id = B.Aid = C.Aid: Assume we have three tables A, B and C with join keys $A.id$, $B.Aid$, and $C.Aid$. There exists query $Q = Q(A) \bowtie Q(B) \bowtie Q(C)$ on join condition $A.id = B.Aid$ AND $B.Aid = C.Aid$. Then, the join cardinality can be expressed as follows:

$$|Q| = \sum_{v \in D(A.id)} P_A(A.id = v|Q(A)) * |Q(A)| * P_B(B.Aid = v|Q(B)) * |Q(B)| * P_C(C.Aid = v|Q(C)) * |Q(C)| \tag{9}$$

Case2 A.id = B.Aid, B.id = C.Bid: Assume we have three tables A, B and C with join keys $A.id$, $B.Aid$, $B.id$, and $C.Bid$. There exists query $Q = Q(A) \bowtie Q(B) \bowtie Q(C)$ on join condition $A.id = B.Aid$ AND $B.id = C.Bid$. Then, the join cardinality can be expressed as follows:

$$\begin{aligned}|Q| = \sum_{v_1 \in D(A.id)} \sum_{v_2 \in D(B.id)} & P_A(A.id = v_1|Q(A)) * |Q(A)| * \\ & P_B(B.Aid = v_1, B.id = v_2|Q(B)) * |Q(B)| * \\ & P_C(C.Bid = v_2|Q(C)) * |Q(C)|\end{aligned} \tag{10}$$

Self join case:

Assume we have one table A with join keys $A.id, A.id_2$. There exists a query $Q = Q(A) \bowtie Q(A')$ on join condition $A.id = A.id_2$. Then, the join cardinality can be expressed as follows:

$$|Q| = \sum_{v \in D(A.id)} P_A(A.id = v|Q(A)) * |Q(A)| * P_A(A.id_2 = v|Q(A')) * |Q(A')| \tag{11}$$

Cyclic join case:

Assume we have two tables A and B with join keys $A.id, A.id_2$ and $B.Aid, B.Aid_2$. There exists a query $Q = Q(A) \bowtie Q(B)$ on join condition $A.id = B.Aid$ AND $A.id_2 = B.Aid_2$. Then, the join cardinality can be expressed as follows:

$$\begin{aligned}|Q| = \sum_{v_1 \in D(A.id)} \sum_{v_2 \in D(A.id_2)} & P_A(A.id = v_1|Q(A)) * |Q(A)| * \\ & P_B(B.Aid = v_1, B.Aid_2 = v_2|Q(B)) * |Q(B)| * \\ & P_A(A.id_2 = v_2|Q(A')) * |Q(A')| \\ = \sum_{v_1 \in D(A.id)} \sum_{v_2 \in D(A.id_2)} & P_B(B.Aid = v_1|Q(B)) * |Q(B)| \\ & P_A(A.id = v_1, A.id_2 = v_2|Q(A)) * |Q(A)| * \\ & P_B(B.Aid_2 = v_2|Q(B')) * |Q(B')|\end{aligned} \tag{12}$$

A.2 Proof of lamma 1

Lemma 2. *Given a join graph $\mathcal{G}$ representing a query Q, there exists a factor graph $\mathcal{F}$ such that the variable nodes in $\mathcal{F}$ are the equivalent key group variables of $\mathcal{G}$ and each factor node represents a table I touched by Q. A factor node is connected to a variable node if and only if this variable represents a join key in table I. The potential function of a factor node is defined as table I's probability distribution of the connected variables (join keys) conditioned on the filter predicates $Q(I)$. Then, calculating the cardinality of Q is equivalent to computing the partition function of $\mathcal{F}$.*

Proof: Assume that we have a query Q joining m tables $A, B, \dots, M$. We represent Q as a join graph G, where each node represents a join key and each edge represents a join relation between two keys connected by it. We further define hyper-nodes in G as a set of nodes whose corresponding join key lies in the same table. Thus, each hyper-nodes naturally represents a table in Q. A visualization of such G can be found in Figure 3. Each connected components of G connects a group of join keys with equal join relation, suggesting that they share the same semantics. Therefore, we consider all join keys in a connected component as a equivalent key group variable, denoted as V_i. Therefore, G defines n equivalent variables $V_1, \dots, V_n$.

Then, assume that we have a set of unnormalized single table distributions $P_A(V_A|Q(A)) * |Q(A)|, \dots, P_M(V_M|Q(M)) * |Q(M)|$ and $P_A(V_{A'}|Q(A')) * |Q(A')|, \dots, P_M(V_{M'}|Q(M')) * |Q(M')|$. Each V_I presents a set of equivalent variables that represent a join key in table I. The I' is the duplicated table introduced by query Q only if there exists a cyclic join situation. Therefore, the cardinality of Q can be writen as:

$$\begin{aligned}|Q| = \sum_{v_1 \in D(V_1)} \sum_{v_2 \in D(V_2)} \cdots \sum_{v_n \in D(V_n)} & (P_A(V_A|Q(A)) * |Q(A)|) * \\ & (P_B(V_B|Q(B)) * |Q(B)|) * \cdots (P_M(V_M|Q(M)) * |Q(M)|) * \\ & (P_A(V_{A'}|Q(A')) * |Q(A')|) * \cdots (P_{M'}(V_{M'}|Q(M')) * |Q(M')|)\end{aligned} \tag{13}$$

Next, let's construct a factor graph $\mathcal{F}$, such that the variable nodes in $\mathcal{F}$ are the equivalent key group variables V_i of $\mathcal{G}$ and the factor nodes in $\mathcal{F}$ represent the tables $A, \dots, M$ touched by Q and the duplicated tables if exist $A', \dots, M'$. A factor node representing table I is connected to a variable node if and only if this variable represents a join key in table I. The potential function of a factor node is defined as the unnormalized probability distribution $P_I(V_I|Q(I)) * |Q(I)|$. The partition function of $\mathcal{F}$ is defined exactly the same as Equation 13. Therefore, instead of summing a nested loop of n variables V_i by brute-force, the partition function (cardinality) can be computed using the well-established variable elimination and belief propogation techniques in PGM domain.

A.3 Probabilistic bound

FactorJoin approximate the exact cardinality computation with a probabilistic bound based inference algorithm. An example on two table join queries can found below. In this section we discuss how to derive an upper bound ($Probabilistic\_bound(A, B, bin_i)$ in this equation) for different case of join.

$$|Q| = \sum_{i=1}^{k} \sum_{v \in bin_i} P_A(A.Id = v|Q(A)) * |Q(A)| * P_B(B.Aid = v|Q(B)) * |Q(B)| \lesssim \sum_{i=1}^{k} Probabilistic\_bound(A, B, bin_i) \tag{14}$$

Case1: Joining two tables $A.id = B.Aid$**:**

Let $V_i^*(A.id) = MAX_{v \in bin_i} |A.id = v|$ be the most frequent value (MFV) count of $A.id$ in a bin bin_i, and same for $V_i^*(B.Aid)$. We have the following bound:

$$\begin{aligned} |Q| = &\sum_{v = D(A.id)} P_A(A.Id = v|Q(A)) * |Q(A)| * \\ &P_B(B.Aid = v|Q(B)) * |Q(B)| \\ \leq &\sum_{i=1}^{k} min(\frac{P_A(A.id \in bin_i|Q(A)) * |Q(A)|}{V_i^*(A.id)}, \\ &\frac{P_B(B.Aid \in bin_i|Q(B)) * |Q(B)|}{V_i^*(B.Aid)}) * \\ &V_i^*(A.id) * V_i^*(B.Aid) \end{aligned} \tag{15}$$

Case2: Joining three tables $A.id = B.Aid = C.Aid$**:**

Let $V_i^*(A.id) = MAX_{v \in bin_i} |A.id = v|$ be the most frequent value (MFV) count of $A.id$ in a bin bin_i, and same for $V_i^*(B.Aid)$, $V_i^*(C.Aid)$. We have the following bound:

$$\begin{aligned} |Q| = &\sum_{v \in D(A.id)} P_A(A.id = v|Q(A)) * |Q(A)| * \\ &P_B(B.Aid = v|Q(B)) * |Q(B)| * P_C(C.Aid = v|Q(C)) * |Q(C)| \\ \leq &\sum_{i=1}^{k} min\{\frac{P_A(A.id \in bin_i|Q(A)) * |Q(A)|}{V_i^*(A.id)}, \\ &\frac{P_B(B.Aid \in bin_i|Q(B)) * |Q(B)|}{V_i^*(B.Aid)} \\ &\frac{P_C(C.Aid \in bin_i|Q(C)) * |Q(C)|}{V_i^*(C.Aid)}\} * \\ &V_i^*(A.id) * V_i^*(B.Aid) * V_i^*(C.Aid) \end{aligned} \tag{16}$$

Case3: Joining three tables $A.id = B.Aid$**,** $B.id = C.Bid$**:**

Let $V_i^*(A.id) = MAX_{v \in bin_i} |A.id = v|$ be the most frequent value (MFV) count of $A.id$ in a bin bin_i, and same for $V_i^*(B.Aid)$, $V_i^*(B.id)$, and $V_i^*(C.Bid)$. We have the following bound, where $Upper(A \bowtie B)$ is derived from Equation 15.

$$\begin{aligned} |Q| = &\sum_{v_1 \in D(A.id)} \sum_{v_2 \in D(B.id)} P_A(A.id = v_1|Q(A)) * |Q(A)| * \\ &P_B(B.Aid = v_1, B.id = v_2|Q(B)) * |Q(B)| * \\ &P_C(C.Bid = v_2|Q(C)) * |Q(C)| \\ \leq &\sum_{i=1}^{k} min\{\frac{Upper(Q(A) \bowtie Q(B))}{V_i^*(A.id) * V_i^*(B.id)}, \\ &\frac{P_C(C.Aid \in bin_i|Q(C)) * |Q(C)|}{V_i^*(C.Bid)}\} * \\ &* V_i^*(A.id) * V_i^*(B.id) * V_i^*(C.Bid) \end{aligned} \tag{17}$$

Case4: Self join of one table $A.id = A.id2$**:**

Let $V_i^*(A.id) = MAX_{v \in bin_i} |A.id = v|$ be the most frequent value (MFV) count of $A.id$ in a bin bin_i, and same for $V_i^*(A.id_2)$. We have the following bound:

$$\begin{aligned} |Q| = &\sum_{v = D(A.id)} P_A(A.Id = v|Q(A)) * |Q(A)| * \\ &P_A(A.id_2 = v|Q(A')) * |Q(A')| \\ \leq &\sum_{i=1}^{k} min(\frac{P_A(A.id \in bin_i|Q(A)) * |Q(A)|}{V_i^*(A.id)}, \\ &\frac{P_A(A.id_2 \in bin_i|Q(A')) * |Q(A')|}{V_i^*(A.id_2)}) * \\ &V_i^*(A.id) * V_i^*(A.id_2) \end{aligned} \tag{18}$$

Case5: Cyclic join of two tables $A.id = B.Aid$**,** $A.id_2 = B.Aid_2$**:**

Let $V_i^*(A.id) = MAX_{v \in bin_i} |A.id = v|$ be the most frequent value (MFV) count of $A.id$ in a bin bin_i, and same for $V_i^*(B.Aid)$, $V_i^*(A.id_2)$, and $V_i^*(B.Aid_2)$. We have the following bound:

$$\begin{aligned} |Q| = &\sum_{v_1 \in D(A.id)} \sum_{v_2 \in D(A.id_2)} P_A(A.id = v_1, A.id_2 = v_2|Q(A)) * \\ &|Q(A)| * P_B(B.Aid = v_1, B.Aid_2 = v_2|Q(B)) * |Q(B)| * \\ &P_A(A.id_2 = v_2|Q(A')) * |Q(A')| \\ \leq &\sum_{i=1}^{k} min\{\frac{Upper(Q(A) \bowtie Q(B)}{V_i^*(A.id_2) * V_i^*(B.Aid)}, \frac{Upper(Q(A') \bowtie Q(B))}{V_i^*(A.id) * V_i^*(B.Aid_2)}\} * \\ &V_i^*(A.id_2) * V_i^*(B.Aid) * V_i^*(A.id) * V_i^*(B.Aid_2) \end{aligned} \tag{19}$$

B GREEDY BIN SELECTION ALGORITHM DETAILS

We observe that the bound on a particular bin bin_i can be very loose if the MFV count V_i^* is a large outlier in bin_i. Taking the two table join query as an example, if bin_i contains only one value that appears 100 times in $A.id$ but 10,000 values that only appear once in $B.Aid$, then the bound could be 100 times larger than the actual cardinality. The existing equal-width or equal-depth binning strategy can generate very large estimation errors, so we design

Algorithm 1 Greedy Bin Selection Algorithm (GBSA)

Input: Equivalent key groups $Gr_1, \ldots, Gr_m$, where $Gr_i = \{Id_i^1, \ldots, Id_i^{|Gr_i|}\}$;
Column data $\mathcal{D}(Id_i^j)$ of all join keys in the DB instance $\mathcal{D}$;
Number of bins k_i for each group Gr_i.

1: **for** $Gr_i \in \{Gr_1, \ldots, Gr_m\}$ **do**
2: $\quad Bin(Gr_i) \leftarrow []$
3: $\quad Gr_i' \leftarrow sort\_key\_based\_on\_domain\_size(\mathcal{D}, Gr_i)$
4: $\quad Bin(Gr_i) \leftarrow get\_min\_variance\_bins(\mathcal{D}(Gr_i'[1]), k_i/2)$
5: $\quad$ remain_bins $\leftarrow k_i/2$
6: $\quad$ **for** $j \in \{2, \ldots, |Gr_i'|\}$ **do**
7: $\quad\quad$ binned_data $\leftarrow apply\_bin\_to\_data(\mathcal{D}(Gr_i'[j]), Bin(Gr_i))$
8: $\quad\quad$ bin_variance $\leftarrow$ $calculate\_variance$(binned_data)
9: $\quad\quad$ arg_sort_idx $\leftarrow arg\_sort\_decreasing(bin\_variance)$
10: $\quad\quad$ **for** $p \in arg\_sort\_idx[1 : remain\_bins/2]$ **do**
11: $\quad\quad\quad Bin(Gr_i) \leftarrow min\_variance\_dichotomy(Bin(Gr_i)[p]$, binned_data[p])
12: $\quad\quad$ **end for**
13: $\quad\quad$ remain_bins $\leftarrow$ remain_bins/2
14: $\quad$ **end for**
15: **end for**
16: **return** $\{Bin(Gr_i) | i, \ldots, m\}$

a new binning strategy called the greedy bin selection algorithm (GBSA) to optimize our bound tightness.

The objective of GBSA is to minimize the variance of the value counts within bin_i. In the extreme case, if the value counts have zero variance for all join keys in one equivalent key group, then our bound can output the exact cardinality (if with perfect single table CardEst models). However, as the same bin partition will be applied for all equivalent join keys, minimizing the value counts variance of bin_i on the domain of one key may result in a bad bin for other keys. Jointly minimizing the variance of one bin for all join keys has exponential complexity. There, GBSA uses a greedy algorithm to iteratively minimize the bin variance for all join keys. At a high level, GBSA first optimizes the minimal variance bins with half number of binning budget $k/2$ on the domain of one join key. Then, it recursively updates these bins by minimizing the variance of other join keys using the rest half of the budget. The details of GBSA are provided in Algorithm 1.

FactorJoin analyzes the schema of DB instance $\mathcal{D}$ to derive m equivalent key groups $Gr_1, \ldots, Gr_m$, each of which contains $|Gr_i|$ join keys $Id_i^1, \ldots, Id_i^{|Gr_i|}$. Let $\mathcal{D}(Id_i^j)$ represents the column data (domain) of join key Id_i^j. GBSA will derive the sub-optimal binning with k_i bins $Bin(Gr_i) = \{bin_1, \ldots, bin_{k_i}\}$ for each the equivalent key group Gr_i (line 1). We explain the procedure of this algorithm for binning one group Gr_i (line 2-14).

First, GBSA sorts the join keys $\{d_i^1, \ldots, Id_i^{|Gr_i|}\}$ in decreasing order based on their domain size (line 3) to get $Gr_i' = \{Id_i^{1'}, \ldots, Id_i^{|Gr_i|'}\}$. We apply half of the binning budget to generate $k_i/2$ minimal variance bins on the domain of $d_i^{1'}$ (line 4). Note that minimal variance bins on a single attribute can be easily obtained by sorting the value counts of $\mathcal{D}(d_i^{1'})$ and applying equal-depth binning over the sorted values. The remaining number of bins available $remain_bins$ is $k_i/2$ (line 5). Then for each rest of the join key d_i^j, GBSA applies the current bins $Bin(Gr_i)$ to its data column $\mathcal{D}(d_i^j)$, calculates the variance for each bin, and sorts these bins in decreasing order (line 6-9). For the top $remain_bins/2$ bins with the largest variance, we dichotomize each bin into two bins to minimize the variance on join key d_i^j (lines 10 -12). We iterative the above procedure until all join keys are optimized.

According to our evaluation results, the GBSA has a significant impact on improving our probabilistic bound tightness and estimation effectiveness.

C ADDITIONAL EXPERIMENTS AND DETAILS

In this section, we provide the detailed comparision of all baselines on *STATS-CEB* and *IMDB-JOB* benchmarks.

Performance on STATS-CEB: We sort the 146 queries of *STATS-CEB* based on their Postgres end-to-end runtime and cluster them into 6 different runtime intervals. Figure 10 reports relative improvements of all competitive baselines over *Postgres* for each query and the overall performance comparision on each cluster of queries on the last figure.

For the very short-running queries (which represent an OLTP-like workload), *Postgres* is the best among all baselines. These baselines perform worse because the estimation latency plays a significant role in these queries. We observe that improving estimation accuracy has a very limited effect on the query plan quality of short-running queries. This also explains why the optimal *TrueCard* only marginally outperforms *Postgres* on queries with less than $2s$ of runtime. Overall, *FactorJoin* has the best performance among all baselines except for *Postgres*.

For the extremely long-running queries, the advantage of the learning-based methods over *Postgres* gradually appear. The reason is that estimation latency becomes increasingly insignificant for queries with a long execution time. *FactorJoin* has comparable performance as the SOTA learning-based methods on these queries.

Performance on IMDB-JOB: Since a large proportion of the queries in *IMDB-JOB* workload are short-running queries, *Postgres* has a better performance than all baselines for more than half of the queries. Similar to STATS-CEB queries, we do not observe any performance gain for using the learned CardEst method over *Postgres* on queries with less on $1s$ runtime. In fact, the optimal *TrueCard* also barely improves the *Postgres*. This suggests that the learned methods shoulf fall back to *Postgres* for OLTP workloads.

Similar to *STATS-CEB*, *FactorJoin* also has a significantly better performance over all other baselines except *Postgres*.

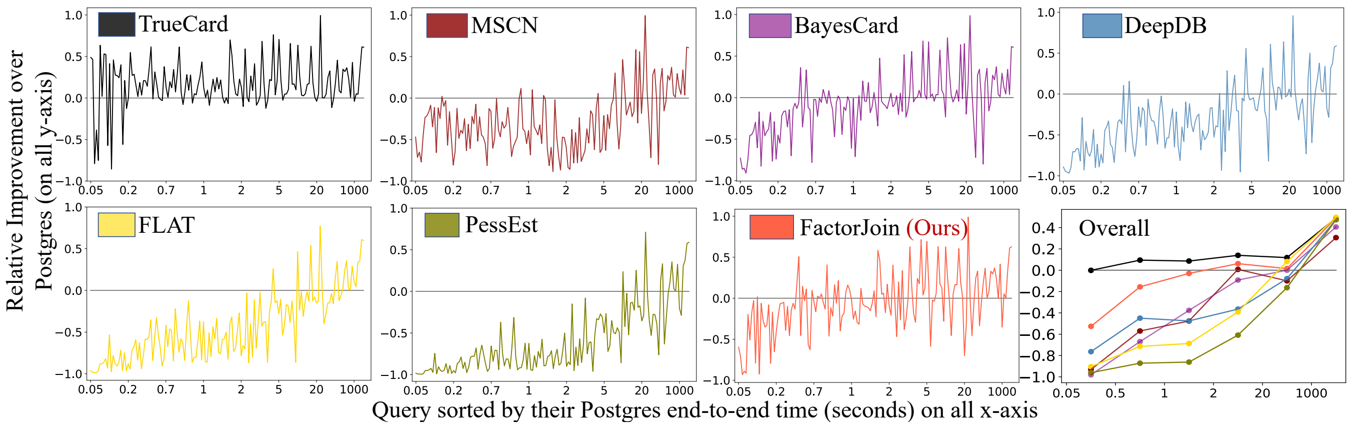

Figure 10: Per query performance of STATS-CEB.

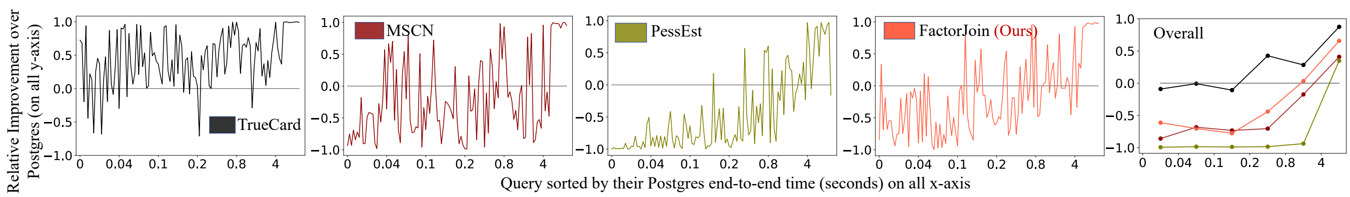

Figure 11: Per query performance of STATS-CEB.