LpBound: Pessimistic Cardinality Estimation using ℓ_p-Norms of Degree Sequences

HAOZHE ZHANG, University of Zurich, Switzerland
CHRISTOPH MAYER, University of Zurich, Switzerland
MAHMOUD ABO KHAMIS, RelationalAI, USA
DAN OLTEANU, University of Zurich, Switzerland
DAN SUCIU, University of Washington, USA

Cardinality estimation is the problem of estimating the size of the output of a query, without actually evaluating the query. The cardinality estimator is a critical piece of a query optimizer, and is often the main culprit when the optimizer chooses a poor plan.

This paper introduces LpBound, a pessimistic cardinality estimator for multi-join queries (acyclic or cyclic) with selection predicates and group-by clauses. LpBound computes a guaranteed upper bound on the size of the query output using simple statistics on the input relations, consisting of ℓ_p-norms of degree sequences. The bound is the optimal solution of a linear program whose constraints encode data statistics and Shannon inequalities. We introduce two optimizations that exploit the structure of the query in order to speed up the estimation time and make LpBound practical.

We experimentally evaluate LpBound against a range of traditional, pessimistic, and machine learning-based estimators on the JOB, STATS, and subgraph matching benchmarks. Our main finding is that LpBound can be orders of magnitude more accurate than traditional estimators used in mainstream open-source and commercial database systems. Yet it has comparable low estimation time and space requirements. When injected the estimates of LpBound, Postgres derives query plans at least as good as those derived using the true cardinalities.

CCS Concepts: • **Information systems → Query optimization**.

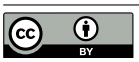
Additional Key Words and Phrases: Cardinality Estimation, Degree Sequence, Lp-norms

ACM Reference Format:
Haozhe Zhang, Christoph Mayer, Mahmoud Abo Khamis, Dan Olteanu, and Dan Suciu. 2025. LpBound: Pessimistic Cardinality Estimation using ℓ_p-Norms of Degree Sequences. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 184 (June 2025), 27 pages. https://doi.org/10.1145/3725321

1 Introduction

The *Cardinality Estimation* problem, or CE for short, is to estimate the output size of a query using only simple, precomputed statistics on the database. CE is one of the oldest and most important problems in databases and data management. It is used as the primary metric guiding cost-based query optimization, for making decisions about every aspect of query execution, ranging from broad logical optimizations like the join order, to deciding the number of servers to distribute

Authors' Contact Information: Haozhe Zhang, haozhe.zhang@uzh.ch, University of Zurich, Department of Informatics, Zurich, Switzerland; Christoph Mayer, christoph.mayer2@uzh.ch, University of Zurich, Department of Informatics, Zurich, Switzerland; Mahmoud Abo Khamis, mahmoud.abokhamis@relational.ai, RelationalAI, Berkeley, CA, USA; Dan Olteanu, dan.olteanu@uzh.ch, University of Zurich, Department of Informatics, Zurich, Switzerland; Dan Suciu, suciu@cs.washington.edu, University of Washington, Department of Computer Science & Engineering, Seattle, WA, USA.



ACM 2836-6573/2025/6-ART184
https://doi.org/10.1145/3725321

the data over, and to detailed physical optimizations, like the use of bitmap filters and memory allocation for hash tables.

Unfortunately, CE is notoriously difficult, and this affects significantly the performance of data management systems. Current systems use density-based estimators, which were pioneered by System R [27]. They make drastic simplifying assumptions (uniformity, independence, containment of values, and preservation of values), and when the query has many joins and many predicates, then they tend to have large errors, leading to poor decisions by the downstream system; for example, the independence assumption often leads to major underestimation [24]. Density-based CE also has limited support for queries with `group-by`: most existing systems yield poor estimates for the number of distinct groups [12]. Yet the main problem with traditional CE is that it does not come with any theoretical guarantees about its estimate: it may under-, or over-estimate, by a little or by a lot, without any warning. Several studies have shown repeatedly that errors in the cardinality estimator can significantly degrade the performance of most advanced database systems [23, 24]. To escape the simplifying assumptions of density-based CE, several estimators were put forward that learn a model of the underlying distribution in the database, e.g., [18, 31, 32, 36]. This is a promising line of work, yet as previously reported (and shown in our experiments), their deployability is poor [30] as they lack explainability, have very slow training time, large model size, and are difficult to transfer with comparable accuracy from one workload to new workloads. One reason for this is that they need to de-normalize the joined relations and add up to exponentially many extra columns to represent new features. Cardinality estimation thus remains one of the major open challenges in data management.

In this paper, we introduce LpBound, a cardinality estimator that offers a one-sided guarantee: the true cardinality is guaranteed to be below that returned by LpBound. That is, LpBound returns a guaranteed *upper bound* on the size of the query output. Moreover, LpBound can explain the computed upper bound in terms of a simple inequality, called a *q-inequality*. This one-sided guarantee can be of use in many applications, for example it can guarantee that a query does not run out of memory, or it can put an upper bound on the number of servers required to distribute the output data. The challenge with this approach is to not overestimate too much. In other words, we want to reduce this upper bound as much as possible, while still maintaining the theoretical one-sided guarantee. To achieve that, we introduce novel statistics on the database, and demonstrate that they lead to strictly improved upper bounds. As an extra bonus, LpBound applies equally well to `group-by` queries.

There have been a small number of implementations that compute upper bounds on the cardinality, commonly called *pessimistic cardinality estimation*, or PCE for short [6, 7, 26]. However, these systems were limited because they used only two types of statistics on the input data: relation cardinalities, $|R|$, and maximum degree (a.k.a. maximum frequency) of an attribute $R.X$: if the values of $R.X$ are $x_1, \ldots, x_N$, then the maximum degree is $\max_i |\sigma_{X=x_i}(R)|$. By using only limited input statistics, these first-generation PCE systems led to significant overestimates, and had worse accuracy than traditional, density-based CE systems.

To achieve better upper bounds, we use significantly richer statistics on the input database. Concretely, we use the *ℓ_p-norms of degree sequences* as inputs to LpBound. The *degree sequence* of an attribute $R.X$ is the sequence $\mathtt{deg}_R(X) = (d_1, d_2, \ldots, d_N)$, sorted in decreasing order, where d_i is the frequency of the value x_i. The ℓ_p-norm is $\left(\sum_i d_i^p\right)^{1/p}$. The ℓ_p-norms of degree sequences are related to *frequency moments* [4]: The p'th frequency moment is $\sum_i d_i^p$. These are commonly used in statistics and machine learning, since they capture important information about the data distribution. This information can be very useful for cardinality estimation too. It is also practical as it can be computed and maintained efficiently [4]. Yet, to the best of our knowledge, the ℓ_p-norms

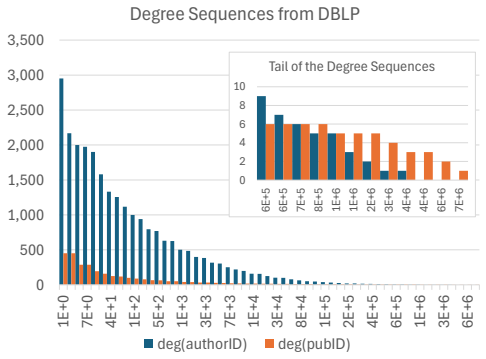


Fig. 1. The Authors-Publication relationship in DBLP (in 2023) with $24{\cdot}10^6$ records pairing $3.6{\cdot}10^6$ authors and $7.1 \cdot 10^6$ publications. The figure shows the degree sequences deg(authorID) and deg(pubID): The latter starts with a lower maximum degree, but has a longer tail (see inset). The author with rank 1 is H. Vincent Poor (2951 publications) and the publication with rank 1 is [28] (with 450 authors).

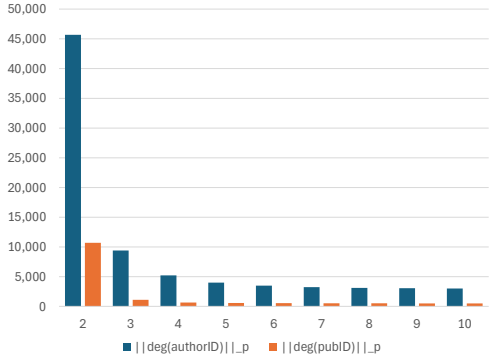


Fig. 2. Instead of storing the two degree sequences in Fig. 1, LpBound stores only some of their ℓ_p-norms, for example for $p \in \{1, \ldots, 10, \infty\}$ shown here. We do not show ℓ_1 (it is equal to the cardinality $24 \cdot 10^6$) and ℓ_∞ (it is the maximum degree, 2951 or 450 respectively). When p ranges from 1 to ∞, the ℓ_p-norm ranges from the relation's cardinality to the maximum degree of any value.

(or frequency moments) have not been used before for cardinality estimation. LpBound is, to the best of our knowledge, the first to use them for cardinality estimation. Fig. 1 shows the degree sequences of the Author-Publication relationship in the DBLP database, and Fig. 2 shows some of their ℓ_p-norms.

LpBound takes as input a query with equality joins, equality and range predicates, and group-by clause, and computes an upper bound on the query output size, by using precomputed ℓ_p-norms on the input database. LpBound offers a strong, theoretical guarantee: for any database that satisfies the given statistics, the query output size is guaranteed to be below the bound returned by LpBound. The bound is *tight*, in the sense that, if all we know about the input database are the given statistics, then there exists a worst-case input database with these statistics on which the query output is as large as the bound returned by LpBound. Finally, LpBound is able to explain the upper bound, in terms of a simple *q-inequality* relating the output size to the input statistics.

Contributions. In this paper we make four main contributions.

1. We introduce LpBound, a PCE that uses ℓ_p-norms of input relations (Sec. 3). LpBound is a principled framework to compute the upper bound, based on information theory, building upon, and expanding a long line of theoretical results [1–3, 5, 14]. We show how to extend previous results to accommodate group-by queries. LpBound works for both cyclic and acyclic queries, and therefore can be used as an estimator both for traditional SQL workloads, which tend to be acyclic, and for graph pattern matching or SparQL queries, which tend to be cyclic.

2. We describe how to use most common values and histograms to extend LpBound to conjunctions and disjunctions of equality and range predicates (Sec. 5). To support predicates, LpBound uses data structures that are very similar to those used by SQL engines, and therefore LpBound could be easily incorporated in those systems.

3. We introduce two optimization techniques for computing the upper bound, which run in polynomial time in the size of the query and the number of available statistics (Sec. 4). One works for acyclic queries only, while the other works for arbitrary conjunctive queries albeit on one-column degree

sequences. These techniques are essential for the practicality of LpBound, as cardinality estimation is often invoked thousands of times during query optimization, and it must run in times measured in milliseconds.

4. We conduct an extensive experimental evaluation of LpBound on real and synthetic workloads (Sec. 6). LpBound can be orders of magnitude more accurate than traditional estimators used in mainstream open-source and commercial database systems. Yet it has low estimation time and space requirements to remain practical. When injected the estimates of LpBound, Postgres derives query plans at least as good as those derived using the true cardinalities.

Related Work. Our paper builds on a long line of theoretical results that proved upper bounds on the size of the query output. The first such result appeared in a landmark paper by Atserias, Grohe, and Marx [5], which proved an upper bound in terms of the cardinalities of the input relations, known today as the *AGM Bound*. The AGM bound is not practical for real SQL workloads, which consist almost exclusively of acyclic queries, where the AGM bound is too large. For example, the AGM bound of a 2-way join is $|R \bowtie S| \leq |R| \cdot |S|$. The AGM bound was extended to account for functional dependencies [2, 14], and further extended to use the maximum degrees of attributes [3]: we refer to the latter as the `max-degree` *bound*. This line of work relies on information theory. A simplified version of the `max-degree` bound was incorporated into two pessimistic cardinality estimators [6, 16]. However, cardinalities and maximum degrees alone are still too limited to infer useful upper bounds for acyclic queries. For example, the `max-degree` bound of a 2-way join is $|R \bowtie_{X=Y} S| \leq \min(a \cdot |S|, |R| \cdot b)$, where a is the maximum degree of $R.X$ and b is the maximum degree of $S.Y$. Real data is often skewed and the maximum degree is large (the maximum degrees are 2951 and 450 in Fig. 1), and this led to large overestimates for more complex queries.

The first system to use degree sequences for pessimistic cardinality estimation was SafeBound [9, 10]. Since the degree sequences are often too large, SafeBound uses a lossy compression of them. It relies solely on combinatorics, it is limited to Berge-acyclic queries (see Sec. 2), and it does not support `group-by`. LpBound can be seen as a significant extension of SafeBound. By using information theory instead of combinatorics, LpBound computes the bound using ℓ_p-norms, without requiring the degree sequences, and also works for cyclic queries and queries with `group-by`.

Prior work [21] introduced a new query evaluation algorithm whose runtime is expressed is expressed in terms of ℓ_p-norms on degree sequences. Their result is limited to binary relations, a single value p for a given query, and to queries whose length of minimal cycles is $\geq p + 1$. An in-depth comparison wwith LpBound ith LpBound is given in prior work [1].

2 Background

Relations. We write $\texttt{Attrs}(R)$ for the set of attributes of a relation R. The domain of attributes $U \subseteq \texttt{Attrs}(R)$ is $\mathsf{Dom}(R.U) \stackrel{\text{def}}{=} \Pi_U(R)$.

Queries. LpBound supports single-block SQL queries:

```
SELECT [groupby-attrs]
FROM R_1, R_2, ..., R_m
WHERE [join-and-selection-predicates]
GROUP BY [groupby-attrs]
```

We ignore aggregates in `SELECT` because they do not affect the output cardinality. We do not support sub-queries. The predicates can be equality, range, and their conjunction and disjunction; `IN` and `LIKE` predicates can be supported with trivial effort (see Sec. 5). Queries with bag semantics are also supported, by replacing them with a full query. For example, the output of `SELECT A FROM ...`

$R =$

X	Y	Z
1	a	...
1	b	...
1	b	...
2	a	...
2	b	...
3	b	...
3	c	...
4	d	...

$$\begin{aligned}\deg_R(YZ|X) &= (3,2,2,1)\\ \deg_R(Y|X) &= (2,2,2,1)\\ \deg_R(Z|XY) &= (2,1,1,1,1,1,1)\\ \deg_R(XYZ|\emptyset) &= (8) = (|R|)\end{aligned}$$

Fig. 3. Examples of Degree Sequences.

(without GROUP BY) is a bag that has the same size as the output of SELECT * FROM ..., which is a set. Throughout the paper we assume that queries have set semantics. We use the conjunctive query notation instead of SQL:

$$Q(V_0) = R_1(V_1) \wedge R_2(V_2) \wedge \cdots \wedge R_m(V_m) \wedge [\text{predicates}] \tag{1}$$

where each $V_j \stackrel{\text{def}}{=} \texttt{Attrs}(R_j)$ is a set of variables, and $V_0 \subseteq V_1 \cup \cdots \cup V_m$ represents the group-by variables. We denote by $\texttt{Vars}(Q) \stackrel{\text{def}}{=} V_1 \cup \cdots \cup V_m = \{X_1, \ldots, X_n\}$ the set of all variables in the query. When $V_0 = \{X_1, \ldots, X_n\}$, then we say that Q is a *full* conjunctive query. Q is *acyclic* if its relations $R_1, \ldots, R_m$ can be placed on the nodes of a tree, such that, for every individual variable X_i, the set of tree nodes that contain X_i forms a connected component. Q is called *Berge-acyclic* if it is acyclic and any two relations share at most one variable. For example, the 3-way join query J_3 is Berge-acyclic, while the 3-clique query C_3 is cyclic:

$$J_3(X,Y,Z,U) = R(X,Y) \wedge S(Y,Z) \wedge T(Z,U) \tag{2}$$

$$C_3(X,Y,Z) = R(X,Y) \wedge S(Y,Z) \wedge T(Z,X) \tag{3}$$

Degree Sequences. Fix a relation instance R, and two sets of variables $X, Y \subseteq \texttt{Attrs}(R)$. The *degree sequence* from X to Y in R is the sequence $\deg_R(Y|X) \stackrel{\text{def}}{=} (d_1, d_2, \ldots, d_N)$ obtained as follows. Compute the domain of X, $\text{Dom}(R.X) = \{x_1, \ldots, x_N\}$, denote by $d_i = |\sigma_{X=x_i}(\Pi_{XY}(R))|$ the degree (or frequency) of x_i, and sort the values in the domain $\text{Dom}(R.X)$ such that their degrees are decreasing $d_1 \geq d_2 \geq \cdots \geq d_N$. We call i the *rank* of the element x_i. Note that $\deg_R(Y|X)$ and $\deg_R(XY|X)$ are the same, where XY denotes the union $X \cup Y$. When $X = \emptyset$, then the degree sequence has length 1, $\deg_R(Y|\emptyset) = (|\text{Dom}(R.Y)|)$. When the functional dependency $X \rightarrow Y$ holds (for example, if X is a key), then $\deg_R(Y|X) = (1, 1, \ldots, 1)$. When $|X| \leq 1$, then we call the degree sequence *simple*, and when $XY = \texttt{Attrs}(R)$, then we call the degree sequence *full* and denote it by $\deg_R(*|X)$, or just $\deg_R(X)$ (we used this notation in Fig. 1). In this paper we only consider simple degree sequences. The degree sequence also applies to the case when R is a bag, not necessarily a set. The ℓ_p-norm of a sequence $\boldsymbol{d} = (d_1, d_2, \ldots)$ is $||\boldsymbol{d}||_p \stackrel{\text{def}}{=} \left(\sum_i d_i^p\right)^{1/p}$, where $p \in (0, \infty]$. When p increases towards ∞, $||\boldsymbol{d}||_p$ decreases and converges to $||\boldsymbol{d}||_\infty \stackrel{\text{def}}{=} \max_i d_i$, see Fig. 2.

Fig. 3 illustrates some simple examples of degree sequences: $\deg_R(YZ|X)$ is both simple and full, and we can write it as $\deg_R(*|X)$ or just $\deg_R(X)$. The degree sequence $\deg_R(Z|XY)$ is not simple.

Density-based CE. The traditional, density-based cardinality estimator [13] is limited to selections and joins. It makes the assumptions mentioned in the introduction and computes the estimate

bottom-up on the query plan, for example:

$$\textsc{Est}(\sigma_{X=\text{value}}(R)) = \frac{|R|}{|\mathsf{Dom}(R.X)|}$$
$$\textsc{Est}(R \bowtie_{X=Y} S) = \frac{|R| \cdot |S|}{\max(|\mathsf{Dom}(R.X)|, |\mathsf{Dom}(S.Y)|)}$$

The ratio $\frac{|R|}{|\mathsf{Dom}(R.X)|}$ is the average degree, $\mathtt{Avg}(\mathsf{deg}_R(*|X))$.

Queries with `group-by` are treated differently by different systems. We describe briefly how they are handled by two open-source systems, illustrating on the following `group-by` query:

$$JG_3(X,U) = R(X,Y) \wedge S(Y,Z) \wedge T(Z,U) \tag{4}$$

DuckDB ignores the `group-by` clause, and estimates the size of JG_3 to be the same as that of the full join J_3 (Eq. (2)). Postgres estimates it as the minimum between the full join, and the product of the domains of the `group-by` variables, $|\mathsf{Dom}(R.X)| \cdot |\mathsf{Dom}(T.U)|$.

Theoretical Upper Bounds. An *upper bound* for a conjunctive query Q is a numerical value, which is computed in terms of statistics on the input database, such as the output size of the query is is guaranteed to be below that bound. The upper bound is *tight* if there exists a database instance, satisfying the statistics, such that the query's output is as large as the bound.[1] The AGM bound [5] is a tight upper that uses only the cardinalities $|R_1|, \ldots, |R_m|$; in other words, it uses only the ℓ_1-norms of full degree sequences. A non-negative sequence $w_1, \ldots, w_m$ is called a *fractional edge cover* of the query Q in Eq. (1) if every variable X_i is "covered", meaning that $\sum_{j \in [m]: X_i \in V_j} w_j \geq 1$. The AGM bound states that $|Q| \leq |R_1|^{w_1} \cdot |R_2|^{w_2} \cdots |R_m|^{w_m}$ for any fractional edge cover. It is useful for cyclic queries like C_3 above (see Sec. 3.1), but for acyclic queries it degenerates to a product of cardinalities, because the optimal edge cover is integral. For example, the AGM bound of the 3-way join in Eq. (2) is $|J_3| \leq |R| \cdot |T|$, because the optimal fractional edge cover is[2] $w_R = 1, w_S = 0, w_T = 1$.

The `max-degree` bound introduced in [3] generalizes the AGM bound by using both cardinalities and maximum degrees; in other words, it uses both ℓ_1 and ℓ_∞ norms of degree sequences. When restricted to acyclic queries, the `max-degree` bound represents an improvement over the AGM, but it is still less accurate than a density-based estimate. For example, the bound for J_3 is the minimum of the following four quantities (see also [7]):

$$\begin{aligned}
|J_3| &\leq |R| \cdot |T| \\
|J_3| &\leq |R| \cdot ||\mathsf{deg}_S(Z|Y)||_\infty \cdot ||\mathsf{deg}_T(U|Z)||_\infty \\
|J_3| &\leq |S| \cdot ||\mathsf{deg}_R(X|Y)||_\infty \cdot ||\mathsf{deg}_T(U|Z)||_\infty \\
|J_3| &\leq |T| \cdot ||\mathsf{deg}_S(Y|Z)||_\infty \cdot ||\mathsf{deg}_R(X|Y)||_\infty
\end{aligned} \tag{5}$$

For comparison, the traditional, density-based estimator for J_3 is:

$$\textsc{Est}(J_3) = \frac{|R| \cdot |S| \cdot |T|}{\max(|\mathsf{Dom}(R.Y)|, |\mathsf{Dom}(S.Y)|) \cdot \max(|\mathsf{Dom}(S.Z)|, |\mathsf{Dom}(T.Z)|)}$$

When $|\mathsf{Dom}(R.Y)| \leq |\mathsf{Dom}(S.Y)|$ and $|\mathsf{Dom}(S.Z)| \leq |\mathsf{Dom}(T.Z)|$ then the estimator becomes $|R| \cdot \mathtt{Avg}(\mathsf{deg}_S(Z|Y)) \cdot \mathtt{Avg}(\mathsf{deg}_T(U|Z))$, which is the same as the `max-degree` bound in Eq. (5) with the maximum degree replaced by the average degree.

SafeBound [9, 10] uses simple, full degree sequences and computes a tight upper bound of a Berge-acyclic, full conjunctive query. For example, if $\mathsf{deg}_R(*|X) = (a_1 \geq a_2 \geq \cdots)$ and $\mathsf{deg}_S(*|Y) =$

[1] Up to some small, query-dependent constant.

[2] Every fractional edge cover must satisfy $w_R \geq 1$ in order to cover X, and $w_T \geq 1$ to cover U; then w_S can be arbitrary. Therefore, $|R|^{w_R}|S|^{w_S}|T|^{w_T} \geq |R| \cdot |T|$.

$(b_1 \geq b_2 \geq \cdots)$, then SafeBound will infer the following bound on a 2-way join: $|R \bowtie_{X=Y} S| \leq \sum_i a_i b_i$. When applied to the 3-way join J_3 SafeBound returns a much better bound than the degree bound (5), but that bound is not described by a closed-form formula; it is only given by an algorithm. The limitations of SafeBound are its lack of explainability, its restriction to Berge-acyclic queries, and its reliance on compression heuristics for the degree sequences.

Information Theory. Let X be a finite random variable, with outcomes $x_1, \ldots, x_N$, and probability function Pr. Its *entropy* is:

$$h(X) \stackrel{\text{def}}{=} -\sum_{i=1,N} \Pr(x_i) \log \Pr(x_i)$$

where log is in base 2. It holds that $0 \leq h(X) \leq \log N$, and $h(X) = \log N$ iff Pr is uniform, i.e., $\Pr(x_1) = \cdots = \Pr(x_N) = 1/N$.

Let $X_1, \ldots, X_n$ be n finite, jointly distributed random variables. They can be described by a finite relation $R(X_1, \ldots, X_n)$, representing their support, and a probability function s.t. for each tuple $t \in R$, $\Pr(t) \geq 0$ and $\sum_{t \in R} \Pr(t) = 1$. For every subset U of variables, $h(U)$ denotes the entropy of the marginal distribution of the random variables in U. For example, we have $h(X_1X_3)$, $h(X_2X_4X_5)$, etc. This defines a vector h with 2^n dimensions, which is called an *entropic vector*. The *conditional entropy* is defined as

$$h(V|U) \stackrel{\text{def}}{=} h(UV) - h(U) \tag{6}$$

The following hold for all subsets of variables $U, V \subseteq \texttt{Attrs}(R)$:

$$h(V) \leq \log |\text{Dom}(R.V)| \qquad h(V|U) \leq \log ||\deg_R(V|U)||_\infty \tag{7}$$

Every entropic vector $\boldsymbol{h}$ satisfies the *basic Shannon inequalities*:

$$h(\emptyset) = 0$$

$$\text{Monotonicity:} \quad h(U \cup V) \geq h(U) \tag{8}$$

$$\text{Submodularity:} \quad h(U) + h(V) \geq h(U \cup V) + h(U \cap V) \tag{9}$$

Every vector $\boldsymbol{h}$ that satisfies the basic Shannon inequalities is called a *polymatroid*. Every entropic vector is a polymatroid, but the converse is not true [33, 35].

3 The LpBound Cardinality Estimator

Our system LpBound is a significant extension of previous pessimistic cardinality estimators, in that it computes a tight upper bound of the query Q by using ℓ_p-norms of simple degree sequences. LpBound can explain its upper bound in terms of a simple inequality, called a *q-inequality*. We introduce LpBound gradually, by first describing the q-inequalities, and showing later how to compute the optimal bound. Throughout this section, we assume that the query has no predicates: we discuss predicates in Sec. 5.

3.1 Q-Inequalities for Full Queries

For upper bounds on a full conjunctive query in terms of ℓ_p-norms, we use inequalities described in [1]. As a simple warmup, consider the 2-way join, which we write as:

$$J_2(X, Y, Z) = R(X, Y) \wedge S(Y, Z) \tag{10}$$

LpBound uses the following q-inequalities:

$$|J_2| \leq |R| \cdot |S| \tag{11}$$
$$|J_2| \leq |R| \cdot ||\mathsf{deg}_S(*|Y)||_\infty \tag{12}$$
$$|J_2| \leq ||\mathsf{deg}_R(*|Y)||_\infty \cdot |S| \tag{13}$$
$$|J_2| \leq ||\mathsf{deg}_R(*|Y)||_2 \cdot ||\mathsf{deg}_S(*|Y)||_2 \tag{14}$$

The first bound is the AGM bound; the next two are the `max-degree` bound, and are always lower (i.e. better) than the AGM bound. The last bound is new, and follows from the Cauchy-Schwartz inequality. LpBound always returns the smallest value of all q-inequalities. It does not need to enumerate all of them; instead it computes the bound differently (explained below in Sec. 3.4), then returns as explanation the single q-inequality that produces that bound.

For the 3-way join J_3 from Eq. (2), LpBound uses many more q-inequalities. It includes all those considered by the `max-degree` bound (Eq. (5)) and many more. We show here only two q-inequalities:

$$\begin{aligned}
|J_3| &\leq ||\mathsf{deg}_R(X|Y)||_2 \cdot |S|^{1/2} \cdot ||\mathsf{deg}_T(U|Z)||_2 \\
|J_3| &\leq |R|^{1/3} \cdot ||\mathsf{deg}_R(X|Y)||_2^{2/3} \cdot ||\mathsf{deg}_S(Z|Y)||_2^{2/3} \cdot ||\mathsf{deg}_T(U|Z)||_3
\end{aligned} \tag{15}$$

To the best of our knowledge, such inequalities have not been used previously in cardinality estimation. We prove (15) in Sec. 3.3. LpBound also improves significantly the bounds of cyclic queries, for example it considers these q-inequalities for the 3-clique C_3:

$$\begin{aligned}
|C_3| &\leq (|R| \cdot |S| \cdot |T|)^{1/2} \\
|C_3| &\leq \left(||\mathsf{deg}_R(Y|X)||_2^2 \cdot ||\mathsf{deg}_S(Z|Y)||_2^2 \cdot ||\mathsf{deg}_T(X|Z)||_2^2\right)^{1/3} \\
|C_3| &\leq \left(||\mathsf{deg}_R(Y|X)||_3^3 \cdot ||\mathsf{deg}_S(Y|Z)||_3^3 \cdot |T|^5\right)^{1/6}
\end{aligned} \tag{16}$$

The first is the AGM bound corresponding to the fractional edge cover $w_R = w_S = w_T = \frac{1}{2}$. The other two are novel and surprising.

3.2 LpBound for `group-by` Queries

Similar q-inequalities hold for queries with `group-by`. We illustrate for the query JG_3 in Eq. (4).

Every q-inequality that holds for the full conjunctive query also holds for the `group-by` query, in other words $|JG_3| \leq |J_3|$, and all upper bounds for J_3 also apply to JG_3; this is used by DuckDB.

Further q-inequalities can be obtained by dropping variables that do not occur in `group-by`, as done in Postgres. For example, we can drop the variables Y, Z from JG_3 and obtain the query:

$$JG_3'(X, U) = R'(X) \wedge T'(U)$$

for which we can infer:

$$|JG_3| \leq |JG_3'| \leq |R'| \cdot |T'| = |\mathsf{Dom}(R.X)| \cdot |\mathsf{Dom}(T.U)|$$

However, LpBound uses many more q-inequalities, which are not necessarily derived using the two heuristics above. For example, consider the following star-join with `group-by`:

$$\mathrm{StarG}(X_1, X_2) = R_1(X_1, Z) \wedge R_2(X_2, Z) \wedge S(Y, Z)$$

LpBound infers (among others) the following inequality:

$$|\mathrm{StarG}| \leq |S|^{1/3} \cdot ||\mathsf{deg}_{R_1}(X_1|Z)||_3 \cdot ||\mathsf{deg}_{R_2}(X_2|Z)||_3 \tag{17}$$

This q-inequality does not hold for the full conjunctive query[3] and it involves all query variables. We prove (17) below.

3.3 Proofs of Q-Inequalities

Consider n finite random variables $X_1, \ldots, X_n$, and let relation $R(X_1, \ldots, X_n)$ be their set of outcomes (see Sec. 2). Then, for any subsets of variables $U, V \subseteq \texttt{Attrs}(R)$ and any $p \in (0, \infty]$, the following holds [1]:

$$\frac{1}{p} h(U) + h(V|U) \leq \log ||\texttt{deg}_R(V|U)||_p \tag{18}$$

Inequalities (7) are special cases of (18), where $p = 1$ or $p = \infty$.

Inequality (18) is very important. It connects an information-theoretic term in the LHS with statistics on the input database in the RHS. All q-inequalities inferred by LpBound follow from (18) and the basic Shannon inequalities. We illustrate with two examples.

First, we prove the q-inequality (15) for J_3. Assume three input relations $R(X, Y), S(Y, Z), T(Z, U)$, and denote by $N \stackrel{\text{def}}{=} |J_3|$ the size of the query's output. Consider the uniform probability distribution with outcomes J_3: every tuple $t = (x, y, z, u) \in J_3$ has the same probability, $\Pr(t) = 1/N$. Therefore, their entropy is $h(XYZU) = \log |J_3|$ (by uniformity), and (15) follows from:

$$\begin{aligned}
&\log|R| + 2\log||\texttt{deg}_R(X|Y)||_2 + 2\log||\texttt{deg}_S(Z|Y)||_2 + 3\log||\texttt{deg}_T(U|Z)||_3 \geq \\
&\geq h(XY) + 2\left(\frac{1}{2}h(Y) + h(X|Y)\right) + 2\left(\frac{1}{2}h(Y) + h(Z|Y)\right) + 3\left(\frac{1}{3}h(Z) + h(U|Z)\right) \\
&= h(XY) + (h(XY) + h(X|Y)) + (h(YZ) + h(Z|Y)) + (h(UZ) + 2h(U|Z)) \\
&= (h(XY) + h(Z|Y) + h(U|Z)) + (h(XY) + h(UZ)) + (h(X|Y) + h(YZ) + h(U|Z)) \\
&\geq (h(XY) + h(Z|XY) + h(U|XYZ)) + h(XYZU) + (h(X|YZ) + h(YZ) + h(U|XYZ)) \\
&= 3h(XYZU) = 3\log|J|
\end{aligned}$$

The first inequality is an application of (18). The equality right after is by (6). The second inequality uses submodularity, for example $h(Z|Y) \geq h(Z|XY)$ follows from $h(YZ) - h(Y) \geq h(XYZ) - h(XY)$, or $h(XY) + h(YZ) \geq h(XYZ) + h(Y)$.

Second, we prove the q-inequality (17). The setup is similar: assume some input relation instances $R_1(X_1, Z), R_2(X_2, Z), S(Y, Z)$, let $\text{StarG}(X_1, X_2)$ be the output of the query, and denote by $N \stackrel{\text{def}}{=} |\text{StarG}|$. Define $\text{Star}(X_1, X_2, Y, Z)$ to be the result of the full join. We need a probability distribution on Star whose marginal on X_1, X_2 is uniform. There are many ways to define such a distribution, we consider the following. Order the tuples in Star arbitrarily; then, for each tuple $t = (x_1, x_2, y, z)$, if there exists some earlier tuple with the same values x_1, x_2 then set $\Pr(t) = 0$, otherwise set $\Pr(t) = 1/N$. At this point, we continue similarly to the previous example:

[3]Proof: consider the instance $R_1 = R_2 = \{(1, 1)\}$, $S = \{(1, 1), (2, 1), \ldots, (N, 1)\}$. The full join returns an output of size N, while the RHS of (17) is $N^{2/3}$.

$$\begin{aligned}
&\frac{1}{3}\log|S| + \log||\mathsf{deg}_{R_1}(X_1|Z)||_3 + \log||\mathsf{deg}_{R_2}(X_2|Z)||_3 \geq \\
&\quad \geq \frac{1}{3}h(YZ) + \frac{1}{3}h(Z) + h(X_1|Z) + \frac{1}{3}h(Z) + h(X_2|Z) \\
&\quad \geq \frac{1}{3}h(Z) + \frac{1}{3}h(Z) + h(X_1|Z) + \frac{1}{3}h(Z) + h(X_2|Z) \\
&\quad = h(Z) + h(X_1|Z) + h(X_2|Z) = h(X_1Z) + h(X_2|Z) \\
&\quad \geq h(X_1Z) + h(X_2|X_1Z) = h(X_1X_2Z) \geq h(X_1X_2) = \log|\text{StarG}|
\end{aligned}$$

3.4 $\mathsf{LP}_{\mathsf{base}}$: The Basic Algorithm of LpBound

LpBound takes as input a query Q (as in Eq. (1)) and a set of statistics on the input database consisting of ℓ_p-norms on degree sequences, and returns: (1) a numerical upper bound B such that $|Q| \leq B$ whenever the input database satisfies these statistics; (2) an explanation consisting of a q-inequality on $|Q|$, which, for the particular numerical values of the ℓ_p-norms implies $|Q| \leq B$; and (3) a proof of the Shannon inequality needed to prove the q-inequality. For that, LpBound solves a Linear Program (LP) called $\mathsf{LP}_{\mathsf{base}}$ defined as follows:

- **The Real-valued Variables** are all unknowns $h(U) \geq 0$, $\forall U \subseteq \mathtt{Vars}(Q)$. There are 2^n real-valued variables.
- **The Objective** is to maximize $h(V_0)$, where V_0 is the set of the `group-by` variables of the query in Eq. (1), under the following two types of linear constraints.
- **The Statistics Constraints** are linear constraints of the form in Eq. (18), one for each ℓ_p-norm of a degree sequence that has been computed on the input database.
- **The Shannon Constraints** are all basic Shannon inequalities of the form in Eq. (8) and (9).

LpBound uses the off-the-shelf solver HiGHS 1.7.2 [19] to solve both $\mathsf{LP}_{\mathsf{base}}$ and its dual linear program. The optimal solution of $\mathsf{LP}_{\mathsf{base}}$ consists of 2^n values $h^*(U)$, one for each set of query variables U. The optimal solution of the dual consists of non-negative weights $w^* \geq 0$, one for every statistics constraint, and non-negative weights $s^* \geq 0$, one for each basic Shannon inequality. LpBound returns the following: (1) The bound $B \stackrel{\text{def}}{=} 2^{h^*(V_0)}$, (2) the q-inequality $|Q| \leq \prod \left(||\mathsf{deg}_R(V|U)||_p\right)^{w^*}$ where the product ranges over all statistics constraints: this uses only the weights associated to the Statistics Constraints, and (3) all basic Shannon inequalities together with their weight s^*: these form the required proof of the q-inequality. We prove in the full paper:

Theorem 3.1. *For any input query Q, LpBound is correct:*

(1) *The quantity B returned by LpBound is a tight upper bound on $|Q|$, meaning that $|Q|$ never exceeds B if the input database satisfies the given statistics, and there exists an input database satisfying the given statistics on which $|Q|$ is as large as B (up to a small query-dependent constant).*
(2) *The q-inequality returned by LpBound holds in general. For the particular values of the statistics $||\mathsf{deg}_R(V|U)||_p$, the inequality implies $|Q| \leq B$.*
(3) *The basic Shannon inequalities multiplied with their associated weights s^* form the proof of the q-inequality.*

Recall that the statistics only use simple degree sequences; without this assumption the tightness statement no longer holds.

Example 3.2. Consider the 3-way join J_3, shown in (2), and assume that, for each relation R, S, T and each attribute, LpBound has access to five precomputed ℓ_p norms: $\ell_1, \ell_2, \ell_3, \ell_4, \ell_\infty$. (Notice that

ℓ_1 is the same as the cardinality: $||\mathsf{deg}_R(*|Y)||_1 = |R|$.) Then the optimal bound to J_3 is given by the following $\mathsf{LP}_{\text{base}}$ linear program, with $2^4 = 16$ variables $h(\emptyset), h(X), h(Y), \ldots, h(XYZU)$:

$$
\begin{aligned}
&\texttt{maximize } h(XYZU) \texttt{ subject to}\\
&\quad h(XY) \le \log|R|\\
&\quad \frac{1}{2}h(Y) + h(X|Y) \le \log||\mathsf{deg}_R(X|Y)||_2\\
&\qquad\qquad \ldots \text{same for all other } \ell_p \text{ norms}\\
&\quad h(X) + h(Y) \ge h(XY) + h(\emptyset)\\
&\quad h(XY) + h(YZ) \ge h(XYZ) + h(Y)\\
&\qquad\qquad \ldots \text{continue with all basic Shannon inequalities}
\end{aligned}
$$

A standard LP package returns both the optimal of this LP, $h^*(U)$, and the optimal of its dual, w^*, s^*. The query's upper bound is $2^{h^*(XYZU)}$. The q-inequality is $|Q| \le |R|^{w_1^*} \cdot ||\mathsf{deg}_R(X|Y)||_2^{w_2^*} \cdots$ where $w_1^*, w_2^*, \ldots$ are the dual variables associated with the statistical constraints. Finally, the dual variables s^* associated to the basic Shannon inequalities provide the proof of the information-theoretic inequality needed to prove the q-inequality.

4 Improving the Estimation Time

LpBound needs to compute the upper bound in milliseconds in order to be of use for query optimization. To achieve this, we start by applying two simple optimizations to the Basic Algorithm $\mathsf{LP}_{\text{base}}$ in Sec. 3.4, which, recall, uses 2^n numerical variables. (1) for each atom $R_j(V_j)$ of the query, we consolidate all variables X_i that do not occur anywhere else into a single variable, and (2) we retain only the *Elemental Basic Shannon Inequalities*[4] [33] in the list of constraints, which are known to be complete. Even with these optimizations, $\mathsf{LP}_{\text{base}}$ takes 100 ms already for queries with $n \approx 10$ logical variables $X_1, \ldots, X_n$ (Fig. 11). We describe below two improvements.

4.1 $\mathsf{LP}_{\text{Berge}}$: Berge-Acyclic Queries

Our first algorithm works under two restrictions: the query needs to be Berge-acyclic (Sec. 2), and all degree constraints must be full and simple. These restrictions are actually quite generous: the JOBjoin, JOBlight, JOBrange, and STATS benchmarks used in Sec. 6 satisfy them. Recall that $V = \{X_1, \ldots, X_n\}$ are the variables of the query Q of the form in Eq. (1), and $R_1(V_1), \ldots, R_m(V_m)$ are the atoms of Q. For each variable X_i, let a_i denote the number of atoms that contain it. We denote by E_Q the following entropic expression:

$$E_Q = \sum_{j=1}^{m} h(V_j) - \sum_{i=1}^{n} (a_i - 1) h(X_i) \tag{19}$$

The linear program called $\mathsf{LP}_{\text{Berge}}$ is the following:

- **The Real-valued Variables** are $h(X_1), \ldots, h(X_n)$ and $h(V_1), \ldots, h(V_m)$. Thus, instead of 2^n real-valued variables $h(U)$, we only have one for each query variable X_j, and one for each set V_j corresponding to an atom $R_j(V_j)$, for a total of $m + n$.
- **The Objective** is to maximize E_Q, under the following constraints.

[4]Eq. (9) is elemental if it is of the form $h(X_iW) + h(X_jW) \ge h(X_iX_jW) + h(W)$ where X_i, X_j are single variables and W a set of variables; (8) is elemental if it is of the form $h(V) \ge h(V - \{X_i\})$ where $V = \{X_1, \ldots, X_n\}$ is the set of all variables.

- **Statistics Constraints:** all constraints in Eq. (18) are included. This is possible because the degree sequence is full and simple, and LHS can be written as $\frac{1}{p}h(U) + h(V|U) = h(UV) - \frac{p-1}{p}h(U)$, where U is a single variable, and UV is the set of variables of some relation.
- **Additivity Constraints:** instead of all Shannon inequalities, we have $1 + |V_j|$ constraints for each atom $R_j(V_j)$ (for a total of $m + \sum_j |V_j|$ constraints):

$$h(V_j) \le \sum_{i:X_i \in V_j} h(X_i) \quad \text{and} \quad h(X_i) \le h(V_j), \forall X_i \in V_j$$

Theorem 4.1. *Given a full Berge-acyclic conjunctive query and full and simple degree constraints on the input database, the optimal values of* LP_{base} *and* LP_{Berge} *are equal.*

The proof uses techniques from information theory and is included in the technical report [34].

Example 4.2. We illustrate $\mathsf{LP}_{\text{Berge}}$ on the 3-way join query J_3 in Eq. (2), and assume for simplicity that the only available statistics are the cardinalities (i.e., the ℓ_1-norm of any full degree sequence): $|R| = |S| = |T| = M$, therefore, the AGM bound applies: $|J_3| \le |R| \cdot |T| = M^2$. The $\mathsf{LP}_{\text{Berge}}$ is the following (where $m \stackrel{\text{def}}{=} \log M$):

$$\begin{aligned}
&\texttt{maximize } E_{J_3} \stackrel{\text{def}}{=} h(XY) + h(YZ) + h(ZU) - h(Y) - h(Z)\\
&\text{subject to}\\
&h(XY) \le m,\ h(YZ) \le m,\ h(ZU) \le m\\
&h(X) \le h(XY),\ h(Y) \le h(XY),\ h(XY) \le h(X) + h(Y)\ //\ \text{for } R(XY)\\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{similarly for } S(YZ) \text{ and } T(ZU)
\end{aligned}$$

We only use 7 real-valued variables: we do not have real-valued variables for $h(XYZ)$ or $h(YU)$ etc. One optimal solution is $h^*(X) = \cdots = h^*(U) = m/2$, $h^*(XY) = h^*(YZ) = h^*(ZU) = m$, and $E^*_{J_3} = 2m$, implying $|J_3| \le M^2$. Notice that the additivity constraints are important in order to obtain a tight bound: if we dropped them, then the linear program admits the feasible solution $h^{**}(X) = \cdots = h^{**}(U) = 0$, $h^{**}(XY) = h^{**}(YZ) = h^{**}(ZU) = m$, and $E^{**}_Q = 3m$, leading to a weaker bound $|J_3| \le M^3$. Thus, the additivity constraints are unavoidable. Statistics beyond cardinalities can easily be added, for example, an ℓ_4-norm constraint on $\mathsf{deg}_S(Z|Y)$ becomes $\frac{1}{4}h(Y) + h(Z|Y) = h(YZ) - \frac{3}{4}h(Y) \le \log ||\mathsf{deg}_S(Z|Y)||_4$.

$\mathsf{LP}_{\text{Berge}}$ can be adapted to Berge-acyclic queries with group-by variables. Given a Berge-acyclic query Q with group-by variables V_0, we can derive a Berge-acyclic query Q' by removing from Q the variables that are not in V_0 and are not join variables. The full Berge-acyclic query Q'', which is obtained from Q' by promoting all variables in Q' to group-by variables, has output size at least that of Q'. The quantity returned by $\mathsf{LP}_{\text{Berge}}$ for Q'' is thus a valid upper bound on the size of Q.

Example 4.3. Consider the Berge-acyclic group-by query StarG from Sec. 3.2. We rewrite it into

$$\text{StarG}''(X_1, X_2, Z) = R_1(X_1, Z) \wedge R_2(X_2, Z) \wedge S'(Z)$$

$\mathsf{LP}_{\text{Berge}}$ maximizes the quantity $E_{|\text{StarG}''|} = h(X_1Z) + h(X_2Z) + h(Z) - 2h(Z)$, under statistics and additivity constraints. This yields a better bound than (17), because the statistics constraints imply:

$$\begin{aligned}
&\frac{1}{3}\log|\mathsf{Dom}(S.Z)| + \log||\mathsf{deg}_{R_1}(X_1|Z)||_3 + \log||\mathsf{deg}_{R_2}(X_2|Z)||_3\\
\ge &\frac{1}{3}h(Z) + \left(\frac{1}{3}h(Z) + h(X_1|Z)\right) + \left(\frac{1}{3}h(Z) + h(X_2|Z)\right) = E_{|\text{StarG}''|}
\end{aligned}$$

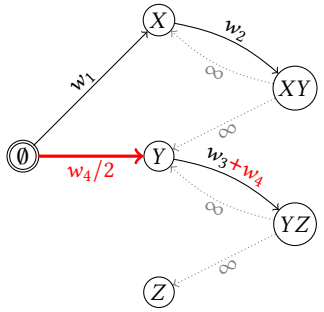


Fig. 4. Example for $\mathsf{LP}_{\text{flow}}$.

which leads to the following q-inequality, improving over (17):

$$|\text{StarG}| \le |\text{StarG}''| \le |\text{Dom}(S.Z)|^{1/3} \cdot ||\text{deg}_{R_1}(X_1|Z)||_3 \cdot ||\text{deg}_{R_2}(X_2|Z)||_3$$

4.2 $\mathsf{LP}_{\text{flow}}$: Using Network Flow

Our second algorithm works for *any* conjunctive query (not necessarily acyclic) and *any* simple degree constraints. It consists of a new linear program, $\mathsf{LP}_{\text{flow}}$, that uses a number of real-valued variables that is quadratic in the query size: this is much better than the exponential number in $\mathsf{LP}_{\text{base}}$, and slightly worse than the linear number in $\mathsf{LP}_{\text{Berge}}$. $\mathsf{LP}_{\text{flow}}$ reduces the problem to a collection of network flow problems. It generalizes the flow-based linear program introduced in [20] for the `max-degree` bound to the general statistics considered by LpBound.

$\mathsf{LP}_{\text{flow}}$ is different from both $\mathsf{LP}_{\text{base}}$ and $\mathsf{LP}_{\text{Berge}}$. We describe it on an example, which illustrates both the original algorithm from [20] and our generalization to ℓ_p-norms. An in-depth account is given in the technical report [34].

Example 4.4. Consider the 2-way join J_2 in Eq. (10), with statistics $|\text{Dom}(R.X)|$, $||\text{deg}_R(Y|X)||_\infty$, and $||\text{deg}_S(Z|Y)||_\infty$. Our target is to find coefficients w_1, w_2, w_3 that make the following q-inequality valid and minimize the bound:

$$|J_2| \le |\text{Dom}(R.X)|^{w_1} \cdot ||\text{deg}_R(Y|X)||_\infty^{w_2} \cdot ||\text{deg}_S(Z|Y)||_\infty^{w_3} \tag{20}$$

For that, the following needs to be a valid information inequality:

$$h(XYZ) \le w_1 h(X) + w_2 h(Y|X) + w_3 h(Z|Y) \tag{21}$$

The key insight from [20] is that checking the validity of such inequality (where all degree constraints are *simple*) is equivalent to constructing a flow network $G = (\text{Nodes}, \text{Edges})$, and checking whether each variable X, Y, Z is independently receiving a maximum flow of at least 1. In our example, the flow network G is shown in Fig. 4 (ignore the **red** part referring to w_4 for now), where the nodes are the source $\emptyset$, individual variables $\{X\}, \{Y\}, \{Z\}$, and sets $\{X, Y\}, \{Y, Z\}$ corresponding to available degree sequences $\text{deg}_R(Y|X), \text{deg}_S(Z|Y)$. The edges are of two types:

- **Forward edges** like $X \to XY$ with capacity w_2. This represents the term $w_2 h(Y|X)$.
- **Backward edges** like $XY \to X$ with capacity ∞. This represents the monotonicity $h(X) \le h(XY)$.

For inequality (21) to be valid,

- X needs to receive a flow of at least 1. Intuitively, this means $w_1 \ge 1$.
- Independently, Y needs to receive a flow of at least 1. Intuitively, there is only one path from the source $\emptyset$ to Y, which is $\emptyset \to X \to XY \to Y$, and this implies that $\min(w_1, w_2, \infty) \ge 1$. Formally, however, we need to setup a standard network flow LP: there is one flow variable $f_{a,b}$

for each edge (a, b), with a capacity constraint $f_{a,b} \le w_{a,b}$, and there is one flow-preservation constraint for each node (other than source and target); e.g., the constraint at node X is $f_{\emptyset,X} + f_{XY,X} - f_{X,XY} = 0$.

- Independently, Z needs to receive a flow of at least 1. Similar to above, this implies that $\min(w_1, w_2, \infty, w_3, \infty) \ge 1$, but formally we need a separate network flow LP.

To capture all network flows using a single LP, we simply create three separate real-valued flow variables for each edge (a, b), namely $f_{a,b;X}, f_{a,b;Y}, f_{a,b;Z}$. The $\mathsf{LP}_{\text{flow}}$ is shown below (ignore the **red** text referring to w_4 for now):

$$\begin{aligned}
\min \quad & w_1 \log |\mathsf{Dom}(R.X)| + w_2 \log ||\mathsf{deg}_R(Y|X)||_\infty \\
& + w_3 \log ||\mathsf{deg}_S(Z|Y)||_\infty + w_4 \log ||\mathsf{deg}_S(Z|Y)||_2 \\
\text{s.t.} \quad & w_1, w_2, w_3, w_4 \ge 0 \\
& (f_{a,b;X})_{(a,b)\in \text{Edges}} \text{ form a flow } \emptyset \to X \text{ of capacity} \ge 1 \\
& (f_{a,b;Y})_{(a,b)\in \text{Edges}} \text{ form a flow } \emptyset \to Y \text{ of capacity} \ge 1 \\
& (f_{a,b;Z})_{(a,b)\in \text{Edges}} \text{ form a flow } \emptyset \to Z \text{ of capacity} \ge 1 \\
& f_{\emptyset,X;*} \le w_1, \quad f_{X,XY;*} \le w_2, \quad f_{Y,YZ;*} \le w_3 + w_4, \quad f_{\emptyset,Y;*} \le w_4/2
\end{aligned} \tag{22}$$

There are $3n \sum_j |V_j|$ total variables, because the network has $2 \sum_j |V_j|$ edges, and for each edge (a, b) we need to create one capacity variable $w_{a,b}$, and n real-valued variables: $f_{a,b;X_i}$, $i = 1, n$.

We next outline a generalization of the above algorithm to handle bounds on arbitrary ℓ_p-norms of degree sequences. Continuing the above example, suppose that we are also given $||\mathsf{deg}_S(Z|Y)||_2$. The RHS of the q-inequality (20) now has an additional factor of $||\mathsf{deg}_S(Z|Y)||_2^{w_4}$ where w_4 is a new coefficient. Similarly, the RHS of inequality (21) now has two additional terms $+\frac{w_4}{2} h(Y) + w_4 h(Z|Y)$. Accordingly, the flow network from Fig. 4 is extended with extra edges, depicted in **red**. In particular, we have an extra edge from $\emptyset$ to Y with capacity $w_4/2$, and an extra edge from Y to YZ adding a capacity of w_4, on top of the existing capacity of w_3. These extra edges lead to new paths that can be used to send flow to Y and Z. As a result, the objective function of the above linear program is extended with the red term. The capacity constraints on the red edges also change: They become $f_{\emptyset,Y;X} \le w_4/2$, $f_{\emptyset,Y;Y} \le w_4/2$, $f_{\emptyset,Y;Z} \le w_4/2$, and similarly $f_{Y,YZ;X} \le w_3 + w_4$ etc.

The above bound can be straightforwardly generalized to handle `group-by` by only considering flows $f_{a,b;X_i}$ where X_i is a `group-by`.

We prove the following theorem in the technical report [34]:

Theorem 4.5. *Given any conjunctive query Q and simple degree constraints on the input database, the optimal values of LP_{base} and LP_{flow} are equal.*

4.3 Putting them Together

Given a query Q, LpBound checks if Q is Berge-acyclic and if all statistics are full (they are always simple), and, in that case it uses $\mathsf{LP}_{\text{Berge}}$ to compute the bound, since its size is only linear in the size of Q and the statistics. Otherwise, it uses $\mathsf{LP}_{\text{flow}}$, whose size is quadratic in the size of the query.

5 Support for Selection Predicates

LpBound can support arbitrary selection predicates on a relation. As long as we can provide ℓ_p-norms on the degree sequences of the join columns for those tuples that satisfy the selection predicate, LpBound can use these norms in the statistics constraints. In the following, we discuss the case of equality and range predicates, and their conjunction and disjunction; `IN` and `LIKE` predicates can be accommodated using data structures like for SafeBound [10].

As data structures to support predicates, LpBound uses simple and effective adaptations of existing data structures in databases: Most Common Values (MCVs) and histograms. Yet instead of a count for each MCV or histogram bucket, LpBound keeps a set of ℓ_p-norms on the degree sequences of the tuples for that MCV or histogram bucket. The simplicity and ubiquity of these data structures make LpBound easy to incorporate in database systems.

In the following, let a relation $R(\mathbf{X}, \mathbf{Y}, A)$ with join attributes $\mathbf{X} = \{X_1, \ldots, X_n\}$, a predicate attribute A, and other attributes $\mathbf{Y}$.

Equality Predicate. For each MCV a of A, we compute ℓ_p-norms for the full and simple degree sequences $\deg_R(*|X_i, A = a)$ for $i = 1, n$. The number of MCVs can significantly affect the accuracy of LpBound (Fig. 10), as it does for SafeBound and Postgres.

We also construct one degree sequence $\mathbf{d}_i$ for all non-MCVs of A and each $i = 1, n$. Let r_i be the maximum number of X_i-values per non-MCV of A and $\mathbf{d}_i$ be the degree sequence of the r_i largest degrees of X_i-values. We compute a set of ℓ_p-norms of each degree sequence $\mathbf{d}_i$. An alternative, more expensive approach is to compute ℓ_p-norms for each non-MCV and take their max for each p.

To estimate for the equality predicate $A = v$, we use the ℓ_p-norms for the degree sequences $\deg_R(*|X_i, A = v)$ if v is an MCV. Otherwise, we use the ℓ_p-norms for the degree sequences $\mathbf{d}_i$.

Range Predicate. Range predicates are supported in LpBound using a hierarchy of histograms: Each layer is a histogram whose number of buckets is half the number of buckets of the histogram at the layer below. We ensure that the histogram at each layer covers the entire domain range of the attribute A. For each histogram bucket with boundaries $[s_i, e_i]$, we create ℓ_p-norms on the full and simple degree sequences $\deg_R(*|X_i, A \in [s_i, e_i])$.

To estimate for the range predicate $A \in [s, e]$, we find the smallest histogram bucket that contains the range $[s, e]$ from the predicate and then use the ℓ_p-norms from that bucket.

Multiple Predicates. In case of a conjunction of predicates, we take as ℓ_p-norm the minimum of the ℓ_p-norms for the predicates, for each p. This is correct as the records must satisfy all predicates and in particular the most selective one. In case of a disjunction, we take as the ℓ_p-norm the sum of the ℓ_p-norms for the predicates, for each p. This *computed* quantity upper bounds the *desired* ℓ_p-norm of the degree sequence for those tuples that satisfy the disjunction of the predicates, yet we cannot compute the latter norm unless we evaluate the predicates. To see this, observe that the desired ℓ_p-norm is less than or equal to the ℓ_p-norm of the degree sequence, which is obtained by the entry-wise sum of the degree sequences for the predicates. By Minkowski inequality, the latter norm is less than or equal to the computed norm.

Optimizations. A challenge for LpBound is to estimate the cardinality of a join, where one operand is orders of magnitude larger than the other operands and has many dangling key values. This happens when a join operand has a selective predicate. By using norms that incorporate degrees of dangling key values, LpBound returns a large overestimate. To address this challenge, it combines two orthogonal optimizations: *predicate propagation* and *prefix degree sequences*. Predicate propagation is used in case of a predicate on a primary-key (PK) relation that is joined with a foreign-key (FK) relation. We propagate the predicate and its attribute through the join to the FK relation without increasing its size. The new predicate on the FK relation is then supported using MCVs and histograms to yield smaller and more accurate ℓ_p-norms. For instance, assume we have a table $R(K, A)$ with primary key K and attribute A on which we have a predicate $\phi(A)$. We also have a table $S(K, B)$ with foreign key K and some attribute B. By propagating $\phi(A)$ from R to S, we mean that we join the two relations to obtain a new relation $S'(K, B, A)$. This relation S' has the same cardinality as S, yet every K-value in S' is now accompanied by the A-value from R. We can

now construct MCVs and histograms on the data column A in S'. A variant of this optimization is also used by SafeBound [10].

In case of a large degree sequence, LpBound also keeps its length (ℓ_0-norm) and the ℓ_p-norms on its prefixes with the 2^i largest degrees, for $i \geq 0$. Then, for a join, LpBound first fetches the ℓ_0-norm of each of the operands. The minimum m of these ℓ_0-norms tells us the maximum number of key values that join at each operand. LpBound uses m to pick the ℓ_p-norms for the i-th prefix[5] of the degree sequences of each of the join operands, for $2^{i-1} \leq m \leq 2^i$.

6 Experimental Evaluation

In this section, we experimentally answer the following questions: *How accurate are LpBound's cardinality estimates? Are LpBound's estimation time and space requirements sufficiently low for it to be practical? Can LpBound's estimates help avoid inefficient query plans, in case faster query plans exist?* Our findings are as follows.

1. LpBound can be orders of magnitude more accurate than traditional estimators used in mainstream open-source and commercial database systems. Yet LpBound has low estimation time (within a couple of ms) and space requirements (a few MBs), which are comparable with those of traditional estimators.

2. Learned estimators can be more accurate than LpBound, according to the errors reported in a prior extensive benchmarking effort [15][6]. This is by design as their models are trained to overfit the specific dataset and possibly the query pattern. Downsides are reported in the literature, including: poor generalization to new datasets and query patterns; non-trivially large training times (including hyper-parameter tuning) and extra space, even an order of magnitude larger than the dataset itself [15].

3. By configuring Postgres to use the estimates of LpBound for the 20 longest-running queries in our benchmarks, we obtained faster query plans than those originally picked by Postgres.

6.1 Experimental Setup

Competitors. We use the *traditional estimators* from open-source systems Postgres 13.14 and DuckDB 0.10.1 and a commercial system DbX. We use two *pessimistic cardinality estimators*: SafeBound [10] and our approach LpBound. We use two classes of *learned cardinality estimators*: (1) The *PGM-based cardinality estimators* BayesCard [31], DeepDB [18], and FactorJoin [30]; and (2) the *ML-based estimators* Flat [36] and NeuroCard [32]. For the latter, we refer to their performance as reported in [15]. We checked with the authors of SafeBound, BayesCard, and FactorJoin that we used the best configurations for their systems and for DeepDB.

Benchmarks. Table 1 shows the characteristics of the queries used in the experiments. They are based on the benchmarks: JOB [25] over the IMDB dataset (3.7GB); STATS[7] over the Stats Stack Exchange network dataset (38MB); and SM (subgraph matching) over the DBLP dataset (26.8MB edge relation and 3.5MB vertex relation) [29]. The SM queries are cyclic, all other queries are acyclic. For IMDB, we use JOBlight and JOBrange queries from previous work [22, 32], which have both equality and range predicates. We further created JOBjoin queries without predicates. We also created JOBlight-gby, JOBrange-gby and STATS-gby queries, which are JOBlight, JOBrange and

[5]The degrees typically decrease exponentially and sequence prefixes for $i > 4$ have norms close to those for the entire degree sequence. For each large degree sequence, we therefore only keep the norms for the first 4 prefixes and for the entire sequence.

[6]These estimation models are copyrighted and not available. Training and tuning the models requires knowledge that is not available (confirmed by authors of [15]).

[7]https://relational-data.org/dataset/STATS

Benchmark	#queries	#rels	#preds	query type
JOBjoin	31	5-14	0	snowflake & full
JOBlight	70	2-5	1-4	star & full
JOBrange	1000	2-5	1-4	star & full
JOBlight-gby	170	2-5	1-4	star & group-by
JOBrange-gby	877	2-5	1-4	star & group-by
STATS	146	2-8	2-16	acyclic & full
STATS-gby	370	2-8	2-16	acyclic & group-by
SM	400	33-84	22-56	cyclic & full

Table 1. Benchmarks used in the experiments.

STATS queries with group-by clauses consisting of at most one attribute per relation[8]: We classify them into three groups of roughly equal size: small domain (domain sizes of the group-by attributes are ≤ 150); large domain (domain sizes > 150); and a mixture of both. The SM queries use 11-28 copies of the edge relation and 2 vertex relation copies per edge relation copy, with one equality predicate per vertex relation copy.

Metrics. We report the estimation error, which is defined as $\frac{\text{Estimated Cardinality}}{\max(1,\text{True Cardinality})}$, where the estimated cardinality is divided by the true cardinality of the query output or by 1 if the true cardinality is 0. The estimation error is greater (less) than 1 in case of over (under)-estimation. We report the (wall-clock) estimation time of the estimators. For LpBound, this is the time to construct and solve the linear programs and does not include the time to load the precomputed statistics and parse the query. We also report the end-to-end query execution time of the 20 longest-running queries using Postgres when injected the estimates of any of the considered estimators. We also report the computation time and extra space needed for the data statistics and ML models used for estimation.

System configuration. We used an Intel Xeon Silver 4214 (48 cores) with 193GB memory, running Debian GNU/Linux 10 (buster). For Postgres, we used the recommended configuration [25]: 4GB shared memory, 2GB work memory, 32GB implicit OS cache, and 6 max parallel workers. We enabled indices on primary/foreign keys. We used the default configuration for data statistics for each estimator. LpBound uses DuckDB to compute the statistics and HiGHS 1.7.2 [19] for solving the linear programs.

6.2 Estimation Errors

Acyclic queries. LpBound has a smaller error range than the traditional estimators and SafeBound for acyclic queries. Fig. 5 plots the estimation errors for the acyclic queries. All systems except LpBound and SafeBound both underestimate and overestimate. The traditional estimators broadly use as estimation the multiplication of the relation sizes and of the selectivities of the query predicates. The selectivity of a join predicate is the inverse of the minimum of the domain sizes of the two join attributes (so average degree, as opposed to maximum degree, is used). For equality and range predicates, Most Common Values (for Postgres) and histograms (for Postgres and DbX) are used. DuckDB has a fixed selectivity of 0.2 for a range predicate. For ML-based estimators, we use the estimates reported in [15], as the models are not available. These models were designed to overfit JOBlight and subsequently fine-tuned to STATS, albeit with a poorer accuracy.

We also report on the estimation errors of the PGM-based estimators. BayesCard and DeepDB do very well on JOBlight; this is the only workload on which their implementation works. FactorJoin

[8]This is not a restriction, LpBound can support arbitrary group-by clauses. This is our methodology for generating query workloads with GROUP-BY.

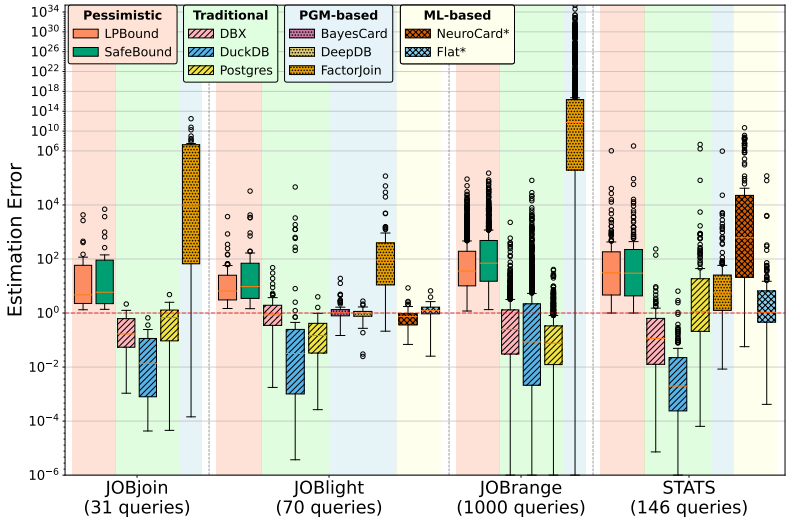


Fig. 5. Estimation errors for JOBJoin, JOBLight, JOBRange, and STATS. For the starred ML-based estimators, we use the errors for JOBLight and STATS reported in the literature [15].

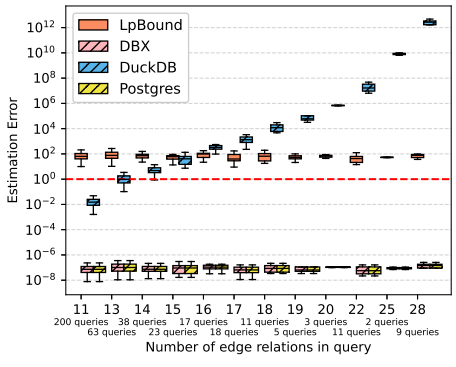


Fig. 6. Estimation errors for the SM cyclic queries.

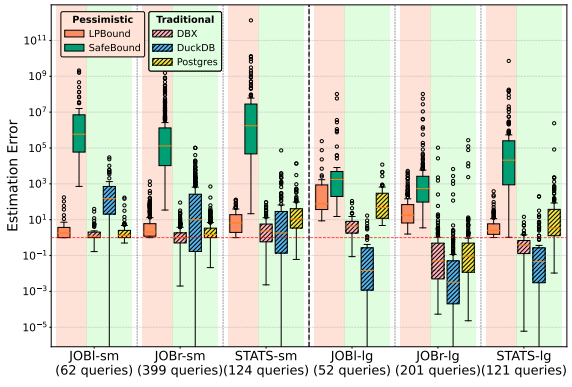


Fig. 7. Estimation errors for group-by queries.

builds high-dimensional probability distributions over the attributes of each relation to capture their correlation. This building task uses random sampling for JOB and the more accurate BayesCard for STATS. FactorJoin faces a trade-off between good accuracy and fast estimation time. To keep the latter practical, it approximates the learned high-dimensional distributions by the product of one-dimensional distributions for JOB[9] and of two-dimensional distributions for STATS. These choices influence the estimation error: It is far more accurate for STATS than for JOBjoin due to the choice of BayesCard over sampling and 2-dimensional over 1-dimensional factorization. The errors for JOBrange are larger possibly due to the larger number (up to 3) of predicates per relation.

[9]The implementation of FactorJoin does not support 2D distributions for JOB.

Cyclic queries. LpBound is the most accurate estimator for the SM cyclic queries. Fig. 6 shows the errors of LpBound and the traditional estimators, grouped by the number of edge relations in the query. The learned estimators do not work for cyclic queries.[10]

As we increase linearly the number n of edge relations from 11 to 28, the number of join conditions between the edge relations increases quadratically in n. This poses difficulties to the traditional estimators, which exhibit two distinct behaviors.

The estimate of Postgres and DbX is 1 for all SM queries and their error is the inverse of the query output size. This is an underestimation by 7-8 orders of magnitude. The estimation is obtained by multiplying: the size of the edge relation n times; the selectivity of each of the n^2 join conditions; the size of the vertex relation $2n$ times; the selectivity of the $2n$ join conditions between the edge and vertex relations; and the selectivity of the $2n$ equality predicates in the $2n$ vertex relations. The product of the relation sizes is much smaller than the inverse of the product of these selectivities, so the estimation is a number below 1, which is then rounded to 1.

The estimate of DuckDB increases exponentially in the number of edge relations, eventually leading to an overestimation by over 12 orders of magnitude. Its estimation ignores most of the join conditions, but accounts for each of the n copies of the edge relation, as explained next. To estimate, DuckDB first constructs a graph, where each node is a relation in the query and there are two edges between any two nodes representing relations that are joined in the query: one edge per attribute participating in the join. Each edge is weighted by the inverse of the domain size of the attribute. DuckDB then takes a minimum-weight spanning tree of this graph. A significant factor in the estimation is then the multiplication of the (n edge and $2n$ vertex) relation sizes at the nodes and of the weights of the edges in the spanning tree ($3n - 1$ domain sizes of one or the other column in the edge or vertex relations). For each of the relations in the query, the estimate has thus a factor proportional to the fraction of the relation size over an attribute's domain size.

Group-by queries. The range of the estimation errors for group-by queries is the smallest for LpBound. Except for LpBound, Postgres, and DbX, the systems ignore the group-by clause and estimate the cardinality for the full query. Fig. 7 shows the errors for the small and large domain classes of JOBlight, JOBrange, and STATS group-by queries (the mixed domain class behaves very similarly to the large domain class). For small domain sizes (first half of figure), LpBound and Postgres use the product of the domain sizes, which is close to the true cardinalities. For large domain sizes (second half), the true cardinalities remain smaller than for the full queries, yet Postgres estimates are for the full queries. This explains why the error boxes are shifted up relative to those in Fig. 5. SafeBound and DuckDB estimate the full query and have large errors.

6.3 Estimation Times

LpBound has a very low estimation time (a few ms) thanks to its LP optimizations. At the other extreme, ML-based estimators can be 1-2 orders of magnitude slower even when taking their average estimation time per subquery instead of the estimation time for all subqueries.

To produce a plan for a query with n relations, a query optimizer uses the cardinality estimates for some of the k-relation sub-queries for $2 \leq k \leq n$. Following prior work [15, 30], we use the sub-queries produced by Postgres's planner for a given query. The range (min-max) of the number of sub-queries is: 8-2018 for JOBjoin; 1-26 for JOBlight; and 1-75 for STATS. The times for JOBrange are not reported, but we expect them to be close to those for JOBlight. SM is not supported by the ML-based estimators and SafeBound.

[10]The implementation of FactorJoin does not support SM queries.

Estimator	JOBjoin		JOBlight		STATS	
	Time	Space	Time	Space	Time	Space
LpBound	0.48 / 10.5	0.04	0.36 / 1.5	1.25	0.49 / 1.6	4.76
SafeBound	0.85 / 147.9	0.07	1.28 / 13.0	1.75	1.89 / 5.6	5.94
DbX$^+$	- / 371.7	-	- / 35.3	-	- / 13.3	-
DuckDB$^+$	- / 99.4	-	- / 535.2	-	- / 30.3	-
Postgres$^+$	- / 19.8	<0.001	- / 3.4	0.001	- / 18.7	0.011
FactorJoin	0.66 / 202	31.6	16.7 / 166.5	22.8	35.3 / 626	8.2
BayesCard	- / -	-	3.0 / 21.7	1.6	- / -	-
DeepDB	- / -	-	4.3 / 28.6	34.0	- / -	-
NeuroCard*	- / -	-	18.0 / -	6.9	23.0 / -	337.0
Flat*	- / -	-	8.6 / -	3.4	175.0 / -	310.0

Table 2. Time (ms): average wall-clock times to compute estimates for (i) a sub-query of a query, averaged over all sub-queries of queries / (ii) a query and all its connected sub-queries, averaged over all queries. Space (MB): extra space for data statistics and models. The times ($^+$) are for the entire query optimization task. The numbers (*) are from prior work [15] and only available for JOBlight and STATS. (-) means unavailable data or unsupported workload.

We report two estimation times per benchmark: (i) the time to compute the estimate for a single sub-query, averaged over all sub-queries of all queries, and (ii) the time to compute the estimates for a query and all its sub-queries, averaged over all queries.

Table 2 reports the estimation times of the estimators. It was not possible to get the estimation times for the traditional estimators, so we report instead their times for the entire query optimization task to give a context for the other reported times; thy should have the lowest estimation times. The type (i) times for the starred ML-based estimators are from [15]. We expect their type (ii) times to be at least an order of magnitude larger than their type (i) times, given the average number of sub-queries per query.

FactorJoin computes the estimates for all sub-queries of a query in a bottom-up traversal of a left-deep query plan. This computation does not parallelize well, however. Its type (i) time is therefore much lower than the type (ii) time.

LpBound can effectively parallelize the LP solving for the sub-queries of a query. Even though one can extend an already constructed LP to accommodate new relations and statistics, we found that it is faster to avoid estimation dependencies between the related sub-queries and estimate for them independently in parallel.

6.4 Space Requirements

The extra space used by LpBound for statistics is 1.6x less than of SafeBound and 1.2-93x less than of the ML-based estimators.

Table 2 shows the amount of extra space needed to store the data statistics or machine learning models used by the estimators.

The traditional estimators use modest extra space. Postgres uses 100 MCVs per predicate column: Increasing the number of MCVs leads to very large estimation time, as it computes the join output size at estimation time for the MCVs. It also uses 100 buckets per histogram and sampling-based estimates of domain sizes for columns. DuckDB only uses (very accurate and computed using hyperloglog [11, 17]) domain size estimates, no MCVs, and no histograms. DbX uses histograms with 200 buckets and no MCVs.

SafeBound uses a compressed representation of the degree sequences and 2056 MCVs on the predicate columns.

Estimator	JOBjoin	JOBlight	JOBrange	STATS	SM
LpBound-ℓ_1	4.57	12.88	44.38	27.84	0.65
LpBound	15.19	20.24	57.9	31.58	1.56
SafeBound	88.56	162.09	209.23	32.04	-
FactorJoin	10068.8	4990.9	5042.7	360.92	-
BayesCard	-	493.36	-	-	-
DeepDB	-	1191.17	-	-	-
NeuroCard *	-	3600	-	-	-
Flat *	-	3060	-	-	-

Table 3. Time (sec) to compute the required statistics for the pessimistic and PGM-based estimators. LpBound-ℓ_1 is LpBound with ℓ_1-norms only. (-) means the system cannot estimate for the respective workload. (*) means the times are from prior work [15], as the code is not available.

LpBound uses up to[11] 5000 MCVs on the predicate columns in JOB, albeit for less space than SafeBound. Both SafeBound and LpBound use hierarchical histograms on data columns with 128 buckets. LpBound stores ℓ_p-norms within each histogram bucket, while SafeBound stores ℓ_1-norms (counts) only. Although not reported in the table, LpBound needs 1.12MB for SM and 8MB for JOBrange. JOBrange has queries with more predicates, which need support, and more columns with large domains.

The models used by NeuroCard and Flat are a feature-rich representation of the datasets. They take more space than the statistics used by the other estimators. For STATS, these models take 10x more space than the dataset itself [15].

6.5 Time to Compute the Statistics

Table 3 gives the times to compute the statistics or models required by the estimators. Overall, the compute time for LpBound is at least one order of magnitude smaller[12] than for the PGM-based estimators. The computation of the statistics used by LpBound is fully expressed in SQL and executed using DuckDB. Such statistics are: the MCVs, the histograms, the ℓ_p-norms ($p \in \{1, \ldots, 10, \infty\}$) for each MCV, histogram bucket, and full relation, and the two optimizations (FKPK and prefix) from Sec. 5. About 80% of LpBound's time is spent on the two optimizations. To better understand the effect of the number of norms, we also report the times for LpBound when restricted to the ℓ_1-norm only. This shows that increasing from 1 to 11 norms only increases the compute time 1.1–3.3 times. DeepDB and BayesCard take 86% and respectively 67% of their compute time for training, the remaining time is for constructing an auxiliary data structure to support efficient sampling. FactorJoin spends most of its time (98%) to construct statistics to speed up the estimation, while relatively very short time (2%) is spent on training the model. The times for NeuroCard and Flat are as reported in prior work [15] for training without hyper-parameter tuning.

6.6 From Cardinality Estimates to Query Plans

When injected the estimates of LpBound, Postgres derives query plans at least as good as those derived using the true cardinalities.

[11] Only 2/8 predicate attributes have domain sizes (134k, 235k) greater than 2k in JOBlight; for JOBrange, there are 3/13 such domains (15k, 23k, 134k). For STATS, the domain sizes are at most 100. For SM, the predicates are on the label attribute from the vertex relation and with domain size 15. Each edge relation joins with two copies of the vertex relation, so we use two predicates to indirectly filter the edge relation. We use 15×15 MCVs to capture all possible combinations of the two predicates.

[12] All estimators in Table 3 except LpBound compute their statistics using Python.

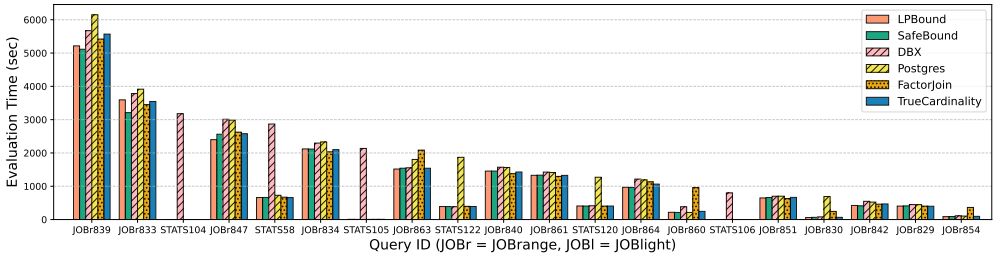


Fig. 8. Postgres (wall-clock) evaluation time for the 20 most expensive queries in JOBlight, JOBrange, and STATS, when injected the estimates of LpBound, SafeBound, DBX, and Postgres or the trues cardinalities for all subqueries of the query. The runtimes for STATS 104, 105, 106 are very small when using the estimates of all systems but DBX and therefore not visible.

This result was expected and aligns with observations from prior work [6, 10]. We verified this for the 20 queries in JOBlight, JOBranges, and STATS (Fig. 8), which took longest to execute using Postgres when the query plan was generated based on the estimations of LpBound, SafeBound, DBX, Postgres, or FactorJoin. We used Postgres for query execution as it easily allows to inject external estimates into its query optimizer[13]. Remarkably, the estimates of LpBound can lead to better Postgres query plans than using true cardinalities, e.g., for the 9 most expensive JOBrange queries in the figure. Prior work [25] also reported this surprising behavior that using true cardinalities, Postgres does not necessarily pick better query plans. SafeBound leads to better plans than LpBound for the top-2 most expensive queries. The estimates of DBX and Postgres lead in many cases to much slower query plans: for STATS104 (STATS122), DBX (Postgres) estimates lead to a plan that is more than 3000x (4x) slower than for the other estimators. For three STATS queries (104, 105, 106), the DBX estimates yield a very slow plan; the runtimes of the plans using the estimates of the other systems are not visible in the plot.

6.7 Performance Considerations for LpBound

How Many ℓ_p-Norms? Using the norms for $p \in [1, 10] \cup \{\infty\}$ gives the best trade-off between the space requirements, the estimation error, and the estimation time for the JOBlight queries (Fig. 9). This was verified to hold also for the other benchmarks. Further norms can still lower the estimation error, but only marginally, and at the expense of more space and estimation time.

How many Most Common Values (MCVs)? Using sufficiently many MCVs to support the estimation for selection predicates can effectively reduce the overall estimation error. For each of the top-k MCVs of a predicate attribute, LpBound stores one set of ℓ_p-norms. It also stores one further set of norms for all remaining attribute values (Sec. 5). For small-domain attributes, e.g., `COMPANY_TYPE`, it is often feasible to have MCVs for each domain value. This significantly improves the estimation accuracy. For large-domain attributes, e.g., `COMPANY_ID`, it not not practical to do so. To decide on the number k of MCVs, one can plot the estimation error as a function of k and pick k so that the improvement in estimation error for larger k is below a threshold, e.g., 1%. Fig. 10 shows that $k \leq 2500$ can yield on average to clear accuracy improvements for JOBlight; this is similar for JOBrange (not shown).

Optimizations for LpBound's Linear Programs. The optimizations introduced for solving LpBound's linear programs are essential for its practicality. Fig. 11 shows the estimation time of

[13]https://github.com/ossc-db/pg_hint_plan

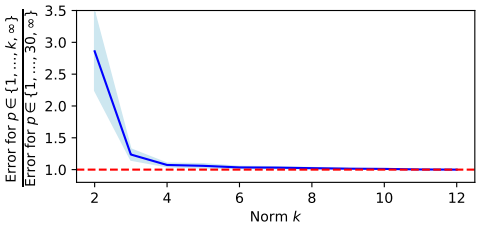


Fig. 9. The amount of useful norms follows the law of diminishing returns: Plotting the division of estimation errors for the norms $\{1,\ldots,k,\infty\}$ and $\{1,\ldots,30,\infty\}$, averaged over the 70 JOBlight queries.

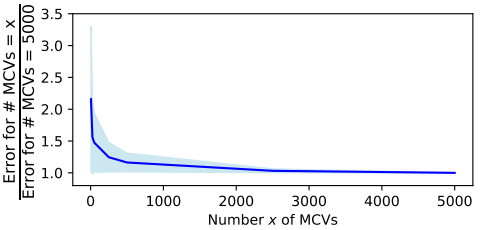


Fig. 10. Effect of the number of MCVs on estimation error for LpBound on JOBlight.

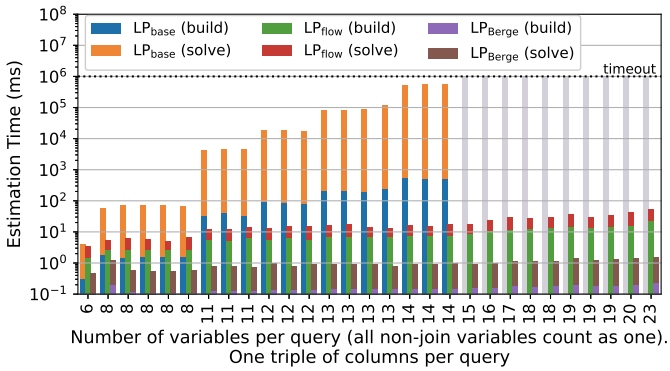


Fig. 11. Estimation times for LpBound on JOBjoin using LP_{base} and its optimizations LP_{flow} and LP_{Berge}.

LpBound using LP_{base}, LP_{flow}, and LP_{Berge}. We used JOBjoin, as its queries are Berge-acyclic and have the largest number of relations and variables among the considered benchmarks, and therefore can stress test and compare the efficiency of the three approaches. LP_{base} takes over 1000 seconds to build and solve a linear program with 2^{15} entropic terms and times out beyond this. LP_{flow} uses a network flow of size at most 15^2 and finishes in under 70 ms for each JOBjoin query. Most of its time is spent constructing the network. LP_{Berge} consistently takes under 2ms for each query.

7 Conclusion and Future Work

In this paper we introduced LpBound, a pessimistic cardinality estimator. It uses two ingredients: ℓ_p-norms of degree sequences of the join columns and information inequalities. The advantages of LpBound over the learned estimators such as FactorJoin, BayesCard, and DeepDB, NeuroCard, and Flat are that it provides: guaranteed upper bounds; low estimation time and estimation error when applied to workloads not seen before; fast construction of the necessary statistics; and a rich query language support with (cyclic and acyclic) equality joins, equality and range predicates, and group-by variables. This language support goes significantly beyond the star or even acyclic queries supported by the competing estimators benchmarked in Section 6.

While LpBound's estimation time is slightly larger than that of traditional estimators, it can nevertheless have lower estimation errors and lead to significantly improved query performance: Fig. 8 shows that the runtime improvement can be up to 3000 seconds for some queries, while its estimation time is only a few milliseconds.

There are two major limitations of LpBound, as introduced in this paper. First, it can non-trivially overestimate the cardinality of joins of highly miscalibrated relations. We introduced two optimizations in Sec. 5 to mitigate this problem. Second, it does not yet support range (and theta) joins, complex and negated predicates, and nested queries. LpBound's flexible framework can in principle accommodate such query constructs, yet this is not immediate and deserves an in-depth treatment in future work. For example, an arbitrary predicate could be accommodated using appropriate data structures that can identify the ranges of tuples that satisfy the predicate and that can be adjusted to store norms on the degree sequences within such ranges. A LIKE predicate can be accommodated, for instance, using a 3-gram index to select ranges of tuples that satisfy the predicate, similar to SafeBound [10]. To guarantee that LpBound returns an upper bound on the true cardinality of the query, the returned ranges must include all matching tuples.

To support nested queries, LpBound needs to become compositional, i.e., to take ℓ_p-norms on input relations and return upper bounds on ℓ_p-norms on the query output. Given a nested query Q, LpBound needs to first compute upper bounds on ℓ_p-norms on the relations representing the sub-queries of Q and then use these bounds to estimate the cardinality of Q.

Future work also needs to address the efficient maintenance of LpBound's estimation under data updates. The following three observations outline a practical approach to achieve this. First, the q-inequality $|Q| \le \prod \left(||\mathsf{deg}_R(V|U)||_p\right)^{w^*}$, where the product ranges over all available statistics constraints, holds with *the same weights* w^* even when the norms $||\mathsf{deg}_R(V|U)||_p$ change. This means that we do not need to solve the linear program after every data update to obtain a valid upper bound on the cardinality of a query Q. Second, the ℓ_p-norms used by LpBound can be expressed in SQL (as for Experiment 6.5) and maintained efficiently under data updates using the view maintenance mechanism of the underlying database system, e.g., delta queries. Third, we envision the use of ℓ_p-sketches [8] for an efficient, albeit approximate, maintenance of the ℓ_p-norms.

Acknowledgments

This work was partially supported by NSF-BSF 2109922, NSF IIS 2314527, NSF SHF 2312195, SNSF 200021-231956, and RelationalAI. The authors would like to acknowledge Luis Torrejón Machado for their help with the preliminary experiments.

References

[1] Mahmoud Abo Khamis, Vasileios Nakos, Dan Olteanu, and Dan Suciu. 2024. Join Size Bounds using ℓ_p-Norms on Degree Sequences. *Proc. ACM Manag. Data* 2, 2 (PODS) (2024), 96. doi:10.1145/3651597

[2] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. 2016. Computing Join Queries with Functional Dependencies. In *PODS*. 327–342. doi:10.1145/2902251.2902289

[3] Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. 2017. What Do Shannon-type Inequalities, Submodular Width, and Disjunctive Datalog Have to Do with One Another?. In *PODS*. 429–444. doi:10.1145/3034786.3056105 Extended version available at http://arxiv.org/abs/1612.02503.

[4] Noga Alon, Yossi Matias, and Mario Szegedy. 1996. The Space Complexity of Approximating the Frequency Moments. In *STOC*. 20–29. doi:10.1145/237814.237823

[5] Albert Atserias, Martin Grohe, and Dániel Marx. 2013. Size Bounds and Query Plans for Relational Joins. *SIAM J. Comput.* 42, 4 (2013), 1737–1767. doi:10.1137/110859440

[6] Walter Cai, Magdalena Balazinska, and Dan Suciu. 2019. Pessimistic Cardinality Estimation: Tighter Upper Bounds for Intermediate Join Cardinalities. In *SIGMOD*. 18–35. doi:10.1145/3299869.3319894

[7] Jeremy Chen, Yuqing Huang, Mushi Wang, Semih Salihoglu, and Kenneth Salem. 2022. Accurate Summary-based Cardinality Estimation Through the Lens of Cardinality Estimation Graphs. *Proc. VLDB Endow.* 15, 8 (2022), 1533–1545. doi:10.14778/3529337.3529339

[8] G. Cormode and K. Yi. 2020. *Small Summaries for Big Data.* Cambridge University Press. https://books.google.com/books?id=6in-DwAAQBAJ

[9] Kyle Deeds, Dan Suciu, Magda Balazinska, and Walter Cai. 2023. Degree Sequence Bound for Join Cardinality Estimation. In *ICDT*. 8:1–8:18. doi:10.4230/LIPICS.ICDT.2023.8

[10] Kyle B. Deeds, Dan Suciu, and Magdalena Balazinska. 2023. SafeBound: A Practical System for Generating Cardinality Bounds. *Proc. ACM Manag. Data* 1, 1 (SIGMOD) (2023), 53:1–53:26. doi:10.1145/3588907

[11] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: The Analysis of a Near-Optimal Cardinality Estimation Algorithm. In *Analysis of Algorithms*. 127–146.

[12] Michael J. Freitag and Thomas Neumann. 2019. Every Row Counts: Combining Sketches and Sampling for Accurate Group-By Result Estimates. In *CIDR*. http://cidrdb.org/cidr2019/papers/p23-freitag-cidr19.pdf

[13] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. 2009. *Database Systems - The Complete Book (2. Ed.)*. Pearson Education.

[14] Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. 2012. Size and Treewidth Bounds for Conjunctive Queries. *J. ACM* 59, 3 (2012), 16:1–16:35. doi:10.1145/2220357.2220363

[15] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2021. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *Proc. VLDB Endow.* 15, 4 (2021), 752–765. doi:10.14778/3503585.3503586

[16] Axel Hertzschuch, Claudio Hartmann, Dirk Habich, and Wolfgang Lehner. 2021. Simplicity Done Right for Join Ordering. In *CIDR*. http://cidrdb.org/cidr2021/papers/cidr2021_paper01.pdf

[17] Stefan Heule, Marc Nunkesser, and Alexander Hall. 2013. HyperLogLog in Practice: Algorithmic Engineering of a State of the Art Cardinality Estimation Algorithm. In *EDBT*. 683–692. doi:10.1145/2452376.2452456

[18] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, Not from Queries! *Proc. VLDB Endow.* 13, 7 (2020), 992–1005. doi:10.14778/3384345.3384349

[19] Qi Huangfu and J. A. J. Hall. 2018. Parallelizing the Dual Revised Simplex Method. *Math. Program. Comput.* 10, 1 (2018), 119–142. doi:10.1007/S12532-017-0130-5

[20] Sungjin Im, Benjamin Moseley, Hung Q. Ngo, Kirk Pruhs, and Alireza Samadian. 2022. Optimizing Polymatroid Functions. *CoRR* abs/2211.08381 (2022). arXiv:2211.08381

[21] Sai Vikneshwar Mani Jayaraman, Corey Ropell, and Atri Rudra. 2021. Worst-case Optimal Binary Join Algorithms under General ℓ_p Constraints. *CoRR* abs/2112.01003 (2021). arXiv:2112.01003

[22] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *CIDR*. http://cidrdb.org/cidr2019/papers/p101-kipf-cidr19.pdf

[23] Kukjin Lee, Anshuman Dutt, Vivek R. Narasayya, and Surajit Chaudhuri. 2023. Analyzing the Impact of Cardinality Estimation on Execution Plans in Microsoft SQL Server. *Proc. VLDB Endow.* 16, 11 (2023), 2871–2883. doi:10.14778/3611479.3611494

[24] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9, 3 (2015), 204–215. doi:10.14778/2850583.2850594

[25] Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2018. Query Optimization Through the Looking Glass, and What We Found Running the Join Order Benchmark. *VLDB J.* 27, 5 (2018), 643–668. doi:10.1007/s00778-017-0480-7

[26] Amine Mhedhbi, Chathura Kankanamge, and Semih Salihoglu. 2021. Optimizing One-time and Continuous Subgraph Queries using Worst-case Optimal Joins. *ACM Trans. Datab. Syst.* 46, 2 (2021), 6:1–6:45. doi:10.1145/3446980

[27] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. 1979. Access Path Selection in a Relational Database Management System. In *SIGMOD*. 23–34. doi:10.1145/582095.582099

[28] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, Alice Xiang, Alicia Parrish, Allen Nie, Aman Hussain, Amanda Askell, Amanda Dsouza, Ambrose Slone, Ameet Rahane, Anantharaman S. Iyer, Anders Andreassen, Andrea Madotto, Andrea Santilli, Andreas Stuhlmüller, Andrew M. Dai, Andrew La, Andrew K. Lampinen, Andy Zou, Angela Jiang, Angelica Chen, Anh Vuong, Animesh Gupta, Anna Gottardi, Antonio Norelli, Anu Venkatesh, Arash Gholamidavoodi, Arfa Tabassum, Arul Menezes, Arun Kirubarajan, Asher Mullokandov, Ashish Sabharwal, Austin Herrick, Avia Efrat, Aykut Erdem, Ayla Karakas, B. Ryan Roberts, Bao Sheng Loe, Barret Zoph, Bartlomiej Bojanowski, Batuhan Özyurt, Behnam Hedayatnia, Behnam Neyshabur, Benjamin Inden, Benno Stein, Berk Ekmekci, Bill Yuchen Lin, Blake Howald, Bryan Orinion, Cameron Diao, Cameron Dour, Catherine Stinson, Cedrick Argueta, Cèsar Ferri Ramírez, Chandan Singh, Charles Rathkopf, Chenlin Meng, Chitta Baral, Chiyu Wu, Chris Callison-Burch, Chris Waites, Christian Voigt, Christopher D. Manning, Christopher Potts, Cindy Ramirez, Clara E. Rivera, Clemencia Siro, Colin Raffel, Courtney Ashcraft, Cristina Garbacea, Damien Sileo, Dan Garrette, Dan Hendrycks, Dan Kilman, Dan Roth, Daniel Freeman, Daniel Khashabi, Daniel Levy, Daniel Moseguí González, Danielle Perszyk, Danny Hernandez, Danqi Chen, Daphne Ippolito, Dar Gilboa, David Dohan, David Drakard, David Jurgens, Debajyoti Datta, Deep Ganguli, Denis Emelin, Denis Kleyko, Deniz Yuret, Derek Chen, Derek Tam, Dieuwke Hupkes, Diganta Misra, Dilyar Buzan, Dimitri Coelho Mollo, Diyi Yang, Dong-Ho Lee, Dylan Schrader, Ekaterina Shutova, Ekin Dogus Cubuk, Elad Segal, Eleanor Hagerman, Elizabeth Barnes, Elizabeth Donoway, Ellie Pavlick, Emanuele Rodolà, Emma Lam, Eric Chu, Eric

Tang, Erkut Erdem, Ernie Chang, Ethan A. Chi, Ethan Dyer, Ethan J. Jerzak, Ethan Kim, Eunice Engefu Manyasi, Evgenii Zheltonozhskii, Fanyue Xia, Fatemeh Siar, Fernando Martínez-Plumed, Francesca Happé, François Chollet, Frieda Rong, Gaurav Mishra, Genta Indra Winata, Gerard de Melo, Germán Kruszewski, Giambattista Parascandolo, Giorgio Mariani, Gloria Wang, Gonzalo Jaimovitch-López, Gregor Betz, Guy Gur-Ari, Hana Galijasevic, Hannah Kim, Hannah Rashkin, Hannaneh Hajishirzi, Harsh Mehta, Hayden Bogar, Henry Shevlin, Hinrich Schütze, Hiromu Yakura, Hongming Zhang, Hugh Mee Wong, Ian Ng, Isaac Noble, Jaap Jumelet, Jack Geissinger, Jackson Kernion, Jacob Hilton, Jaehoon Lee, Jaime Fernández Fisac, James B. Simon, James Koppel, James Zheng, James Zou, Jan Kocon, Jana Thompson, Janelle Wingfield, Jared Kaplan, Jarema Radom, Jascha Sohl-Dickstein, Jason Phang, Jason Wei, Jason Yosinski, Jekaterina Novikova, Jelle Bosscher, Jennifer Marsh, Jeremy Kim, Jeroen Taal, Jesse H. Engel, Jesujoba Alabi, Jiacheng Xu, Jiaming Song, Jillian Tang, Joan Waweru, John Burden, John Miller, John U. Balis, Jonathan Batchelder, Jonathan Berant, Jörg Frohberg, Jos Rozen, José Hernández-Orallo, Joseph Boudeman, Joseph Guerr, Joseph Jones, Joshua B. Tenenbaum, Joshua S. Rule, Joyce Chua, Kamil Kanclerz, Karen Livescu, Karl Krauth, Karthik Gopalakrishnan, Katerina Ignatyeva, Katja Markert, Kaustubh D. Dhole, Kevin Gimpel, Kevin Omondi, Kory Mathewson, Kristen Chiafullo, Ksenia Shkaruta, Kumar Shridhar, Kyle McDonell, Kyle Richardson, Laria Reynolds, Leo Gao, Li Zhang, Liam Dugan, Lianhui Qin, Lidia Contreras Ochando, Louis-Philippe Morency, Luca Moschella, Lucas Lam, Lucy Noble, Ludwig Schmidt, Luheng He, Luis Oliveros Colón, Luke Metz, Lütfi Kerem Senel, Maarten Bosma, Maarten Sap, Maartje ter Hoeve, Maheen Farooqi, Manaal Faruqui, Mantas Mazeika, Marco Baturan, Marco Marelli, Marco Maru, María José Ramírez-Quintana, Marie Tolkiehn, Mario Giulianelli, Martha Lewis, Martin Potthast, Matthew L. Leavitt, Matthias Hagen, Mátyás Schubert, Medina Baitemirova, Melody Arnaud, Melvin McElrath, Michael A. Yee, Michael Cohen, Michael Gu, Michael I. Ivanitskiy, Michael Starritt, Michael Strube, Michal Swedrowski, Michele Bevilacqua, Michihiro Yasunaga, Mihir Kale, Mike Cain, Mimee Xu, Mirac Suzgun, Mitch Walker, Mo Tiwari, Mohit Bansal, Moin Aminnaseri, Mor Geva, Mozhdeh Gheini, Mukund Varma T., Nanyun Peng, Nathan A. Chi, Nayeon Lee, Neta Gur-Ari Krakover, Nicholas Cameron, Nicholas Roberts, Nick Doiron, Nicole Martinez, Nikita Nangia, Niklas Deckers, Niklas Muennighoff, Nitish Shirish Keskar, Niveditha Iyer, Noah Constant, Noah Fiedel, Nuan Wen, Oliver Zhang, Omar Agha, Omar Elbaghdadi, Omer Levy, Owain Evans, Pablo Antonio Moreno Casares, Parth Doshi, Pascale Fung, Paul Pu Liang, Paul Vicol, Pegah Alipoormolabashi, Peiyuan Liao, Percy Liang, Peter Chang, Peter Eckersley, Phu Mon Htut, Pinyu Hwang, Piotr Milkowski, Piyush Patil, Pouya Pezeshkpour, Priti Oli, Qiaozhu Mei, Qing Lyu, Qinlang Chen, Rabin Banjade, Rachel Etta Rudolph, Raefer Gabriel, Rahel Habacker, Ramon Risco, Raphaël Millière, Rhythm Garg, Richard Barnes, Rif A. Saurous, Riku Arakawa, Robbe Raymaekers, Robert Frank, Rohan Sikand, Roman Novak, Roman Sitelew, Ronan LeBras, Rosanne Liu, Rowan Jacobs, Rui Zhang, Ruslan Salakhutdinov, Ryan Chi, Ryan Lee, Ryan Stovall, Ryan Teehan, Rylan Yang, Sahib Singh, Saif M. Mohammad, Sajant Anand, Sam Dillavou, Sam Shleifer, Sam Wiseman, Samuel Gruetter, Samuel R. Bowman, Samuel S. Schoenholz, Sanghyun Han, Sanjeev Kwatra, Sarah A. Rous, Sarik Ghazarian, Sayan Ghosh, Sean Casey, Sebastian Bischoff, Sebastian Gehrmann, Sebastian Schuster, Sepideh Sadeghi, Shadi Hamdan, Sharon Zhou, Shashank Srivastava, Sherry Shi, Shikhar Singh, Shima Asaadi, Shixiang Shane Gu, Shubh Pachchigar, Shubham Toshniwal, Shyam Upadhyay, Shyamolima (Shammie) Debnath, Siamak Shakeri, Simon Thormeyer, Simone Melzi, Siva Reddy, Sneha Priscilla Makini, Soo-Hwan Lee, Spencer Torene, Sriharsha Hatwar, Stanislas Dehaene, Stefan Divic, Stefano Ermon, Stella Biderman, Stephanie Lin, Stephen Prasad, Steven T. Piantadosi, Stuart M. Shieber, Summer Misherghi, Svetlana Kiritchenko, Swaroop Mishra, Tal Linzen, Tal Schuster, Tao Li, Tao Yu, Tariq Ali, Tatsu Hashimoto, Te-Lin Wu, Théo Desbordes, Theodore Rothschild, Thomas Phan, Tianle Wang, Tiberius Nkinyili, Timo Schick, Timofei Kornev, Titus Tunduny, Tobias Gerstenberg, Trenton Chang, Trishala Neeraj, Tushar Khot, Tyler Shultz, Uri Shaham, Vedant Misra, Vera Demberg, Victoria Nyamai, Vikas Raunak, Vinay V. Ramasesh, Vinay Uday Prabhu, Vishakh Padmakumar, Vivek Srikumar, William Fedus, William Saunders, William Zhang, Wout Vossen, Xiang Ren, Xiaoyu Tong, Xinran Zhao, Xinyi Wu, Xudong Shen, Yadollah Yaghoobzadeh, Yair Lakretz, Yangqiu Song, Yasaman Bahri, Yejin Choi, Yichi Yang, Yiding Hao, Yifu Chen, Yonatan Belinkov, Yu Hou, Yufang Hou, Yuntao Bai, Zachary Seid, Zhuoye Zhao, Zijian Wang, Zijie J. Wang, Zirui Wang, and Ziyi Wu. 2023. Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models. *Trans. Mach. Learn. Res.* 2023 (2023). https://openreview.net/forum?id=uyTL5Bvosj

[29] Shixuan Sun and Qiong Luo. 2020. In-Memory Subgraph Matching: An In-depth Study. In *SIGMOD*. 1083–1098. doi:10.1145/3318464.3380581

[30] Ziniu Wu, Parimarjan Negi, Mohammad Alizadeh, Tim Kraska, and Samuel Madden. 2023. FactorJoin: A New Cardinality Estimation Framework for Join Queries. *Proc. ACM Manag. Data* 1, 1 (SIGMOD) (2023), 41:1–41:27. doi:10.1145/3588721

[31] Ziniu Wu and Amir Shaikhha. 2020. BayesCard: A Unified Bayesian Framework for Cardinality Estimation. *CoRR* abs/2012.14743 (2020). arXiv:2012.14743 https://arxiv.org/abs/2012.14743

[32] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: One Cardinality Estimator for All Tables. *Proc. VLDB Endow.* 14, 1 (2020), 61–73. doi:10.14778/3421424.3421432

[33] Raymond W. Yeung. 2008. *Information Theory and Network Coding* (1 ed.). Springer Publishing Company.

[34] Haozhe Zhang, Christoph Mayer, Mahmoud Abo Khamis, Dan Olteanu, and Dan Suciu. 2025. LpBound: Pessimistic Cardinality Estimation using ℓ_p-Norms of Degree Sequences. *CoRR* abs/2502.05912 (2025). doi:10.48550/ARXIV.2502.05912

[35] Zhen Zhang and Raymond W Yeung. 1998. On Characterization of Entropy Function via Information Inequalities. *IEEE Trans. Inf. Theory* 44, 4 (1998), 1440–1452.

[36] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, Jingren Zhou, and Bin Cui. 2021. FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation. *Proc. VLDB Endow.* 14, 9 (2021), 1489–1502. doi:10.14778/3461535.3461539

Received October 2024; revised January 2025; accepted February 2025