Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors

Guido Moerkotte[1] Thomas Neumann[2] Gabriele Steidl[1]

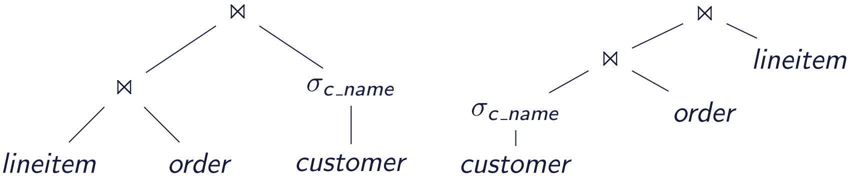
[1]University of Mannheim

[2]Max-Planck Institute for Informatics

August 27, 2009

Motivation

Query optimization, in particular join ordering, has a large impact.



6,001,215 intermediate results

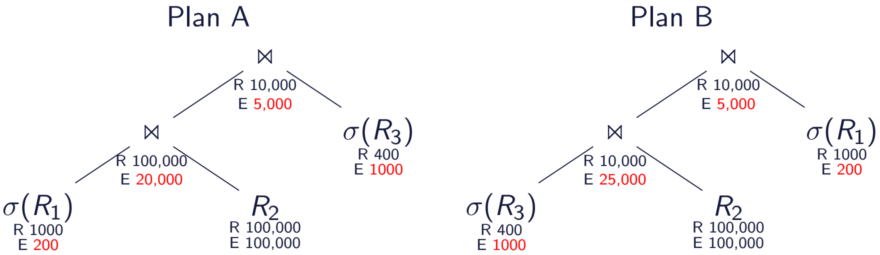
6 intermediate results

- optimization relies upon a cost model
- a central component: **cardinality/selectivity estimation**
- usually relies upon statistical synopses

Motivation (2)

Estimation errors can have very unfortunate consequences.

R = real cardinalities, E = estimated cardinalities



R 100,000 intermediate tuples
E 20,000 intermediate tuples

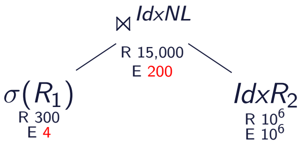
R 10,000 intermediate tuples
E 25,000 intermediate tuples

- errors amplify each other
- errors propagate multiplicative

How can we limit the effect of estimation errors?

Motivation (3)

Misestimations can be even more painful with physical cost models:



- R_1 (without σ), could be large, e.g. $|R_1| = 10^7$
- then such estimation errors can easily happen for mildly skewed data

We observed such numbers in a commercial database system!

- estimates get more and more "dangerous" the smaller they get
- but overestimations are just as bad
- must take the effect on plans into account

Overview

Measuring Estimation Errors

When estimating cardinalities **error are inevitable** in general

- synopsis usually try to minimize errors
- different error metrics to choose from

For simplicity, consider point queries of the form $\sigma_{a=const}(R)$

- let $f(x)$ be $|\sigma_{a=x}(R)|$ for $x \in D_a = \Pi_a(R)$
- let $\hat{f}(x)$ be the estimated derived from a synopsis

Popular error metrics (= optimization goals)

$$l_2 = \sqrt{\sum_{x \in D_a} (f(x) - \hat{f}(x))^2}$$

$$l_\infty = \max_{x \in D_a} |f(x) - \hat{f}(x)|$$

Minimizing these error can lead to **arbitrarily bad plans**!

Defining the Q(uotient) Error

- as errors propagate multiplicative, the metric should be multiplicative
- it should be symmetric regarding over- and underestimation

We define the q-error as follows

$$l_q = \max_{x \in D_a} \frac{\max(f(x), \hat{f}(x))}{\min(f(x), \hat{f}(x))}$$

(we also use the notation $||\frac{\hat{f}}{f}||_Q$ for l_q)

- true cardinality 10, estimation 100 $\Rightarrow l_q = 10$
- true cardinality 10, estimation 1 $\Rightarrow l_q = 10$
- note that l_q is the maximum over the whole domain

Knowing the q-error allows for deriving **bounds on the resulting plan**!

Optimality of the Resulting Plan

Notation: Using the estimates $\hat{f}$ instead of the true cardinalities f transforms a cost function C into $\hat{C}$.
f_i is the selectivity of $\sigma_{p_i}(R_i)$, $f_{i,j}$ the join selectivity $R_i \bowtie R_j$.

Theorem: Let C be a cost function with ASI property. For a given chain query in n relations, let P be the optimal left-deep plan without cross products under C, and $\hat{P}$ be the optimal left-deep plan without cross products under $\hat{C}$. If for all $1 \leq k \leq n$

$$||\frac{\hat{f}_k}{f_k}||_Q < \min_{i \neq j-1} \sqrt{||\frac{f_i f_{i,i+1}|R_i|}{f_j f_{j,j-1}|R_j|}||_Q} =: q,$$

then $C(\hat{P}) = C(P)$.

I.e., the optimizer will find the **real optimal solution** by using the estimates if the l_q error is bound by q.

Optimality of the Resulting Plan (2)

We could show similar optimality bounds for

- star queries
- tree queries
- star queries using sort-merge joins (non-ASI cost function)

In these cases we could show optimality if l_q is bound by a suitable value q.

- for grace-hash-join not proof yet
- but a numerical study suggests the same

We cannot always meet bounds in practice (e.g., for ties in selectivity), but **first metric with proofs of optimality deriving from estimation errors**.

Cost Bounds Implied by Q

Theorem: Let $C = C_{\mathsf{SMJ}}$ or $C = C_{\mathsf{GHJ}}$. For a given query in n relations, let P be the optimal plan under C, and $\hat{P}$ be the optimal plan under $\hat{C}$. Then

$$C(\hat{P}) \leq q^4 C(P),$$

where q is defined as

$$q = \max_{x \subseteq X} ||\hat{S}_x / S_x||_Q,$$

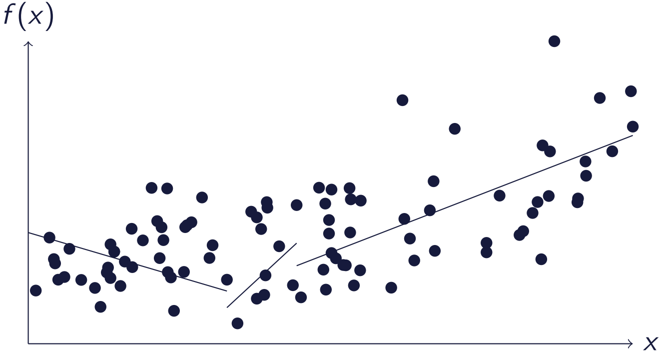
with X being the set of relations to be joined. That is, q is the maximum estimation error taken over all intermediate results.

Overview

1. Motivation
2. Bounding the Impact of Cardinality Estimation Errors
3. **Constructing Synopses**
4. Evaluation
5. Conclusion

What kind of Synopsis?

- construct synopsis that minimize the *q*-error
- our algorithm can construct bucket histograms with polynomials



degree 0 $f(x) = c$ is very common
degree 1 $f(x) = mx + c$ is often more effective (with given space)
degree ≥ 2 does not pay off most of the time

We concentrate on line segments here.

About the construction

Algorithm operates in two layers:

1. single bucket construction
   - fits a line to given data points
2. multi-bucket constructions
   - chooses the bucket boundaries
   - uses single-bucket construction as building block

Algorithm constructs the **optimal fit** for the *q*-error

- math is somewhat involved and lengthy
- approach is even more general than just polynomials
- formal proof of optimality is in the paper
- will only sketch the idea for linear fits here

Single-Bucket Construction

We fit a function $\alpha x + \beta$ to data points $x_1, \ldots, x_n$

1. $n = 1$ (trivial)
   $\alpha = 0$, $\beta = f(x_1)$
2. $n = 2$ (trivial)
   $\alpha = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$, $\beta = f(x_1) - \alpha x_1$
3. $n = 3$ (solve equation system)

$$
\begin{aligned}
\alpha &= (x_3 - x_1)\sqrt{\frac{f(x_2)}{(x_3 - x_1)[f(x_3)(x_3 - x_1) + (f(x_3) - f(x_1))(x_3 - x_2)]}} \\
\beta &= \alpha f(x_3)\frac{x_3 - x_1}{f(x_3) - f(x_1)} - \alpha x_3 \text{ if } \alpha \neq 0 \\
&= f(x_1)\sqrt{\frac{f(x_2)}{f(x_1)}} \text{ if } \alpha = 0
\end{aligned}
$$

Single-Bucket Construction (2)

For $n > 3$ we use an iterative algorithm:

1. pick three data points, compute a linear fit
   - gives a lower bond for the estimation error
2. find the data point with the maximum estimation error
   - we can potentially improve by fitting to it
3. chose it as base point, replaces one of the three
4. repeat until the maximum error remains the same

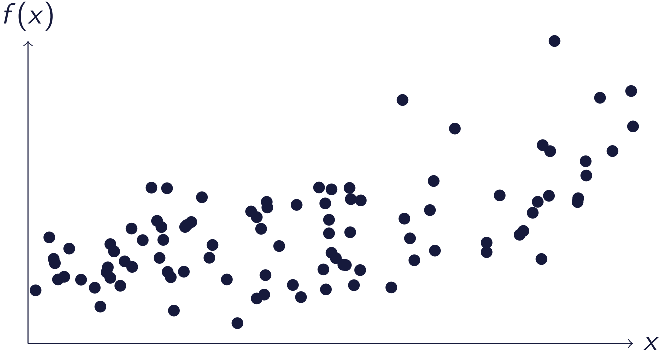
Algorithm in the paper converges faster but is more complex.

Iterative approach, very fast in practice.

Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

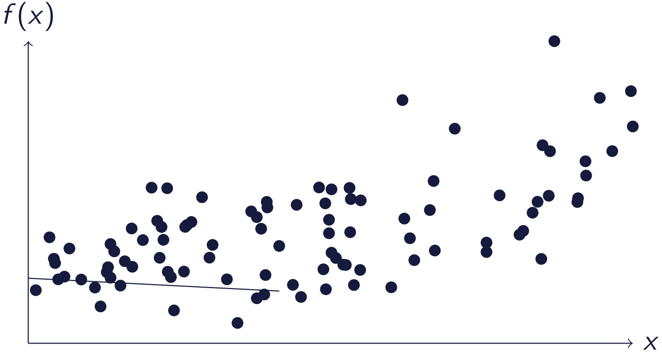
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

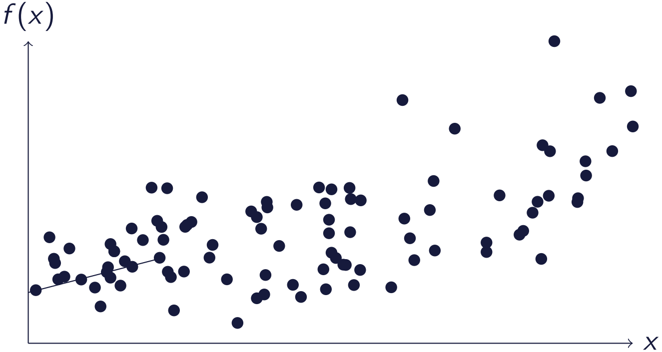
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

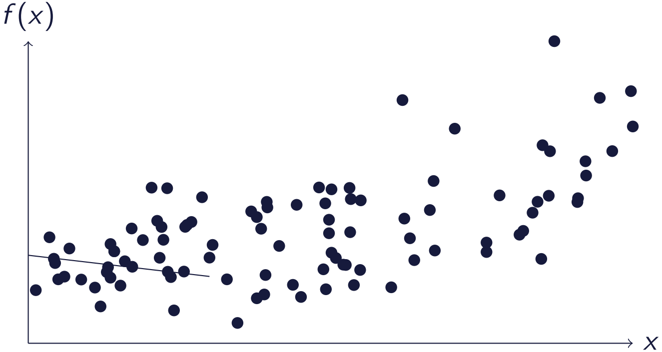
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

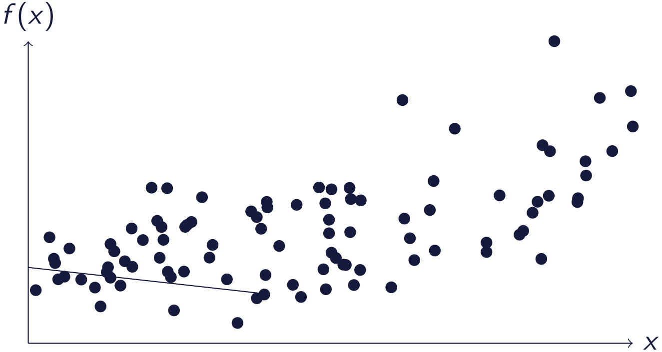
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

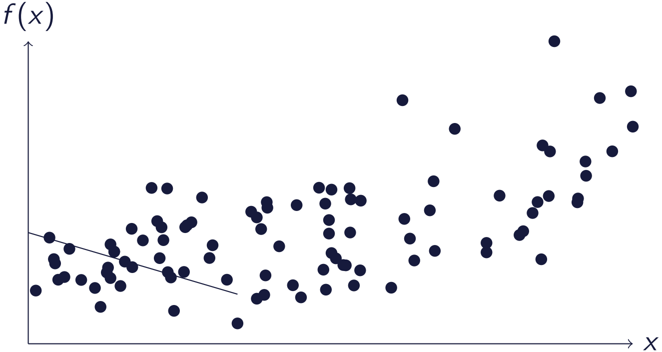
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

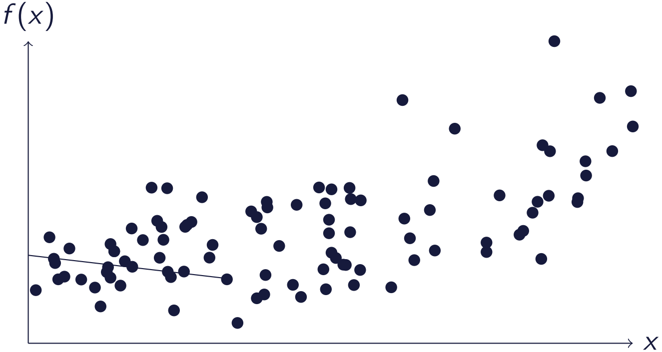
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

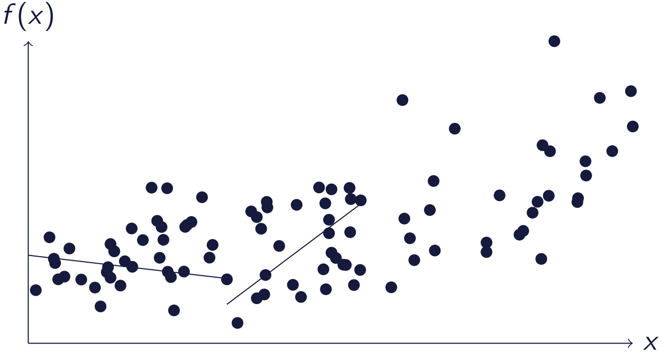
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

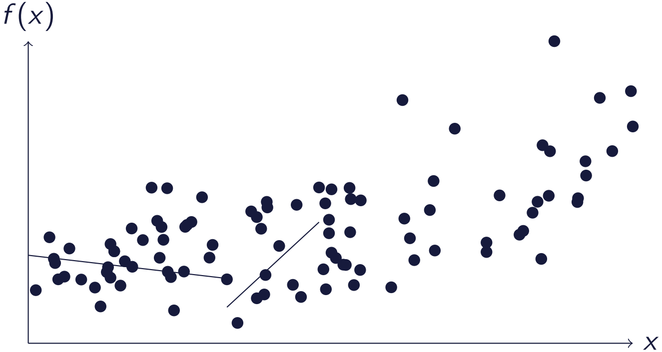
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

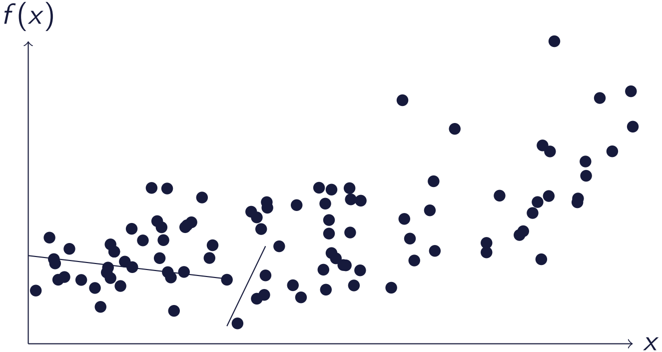
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

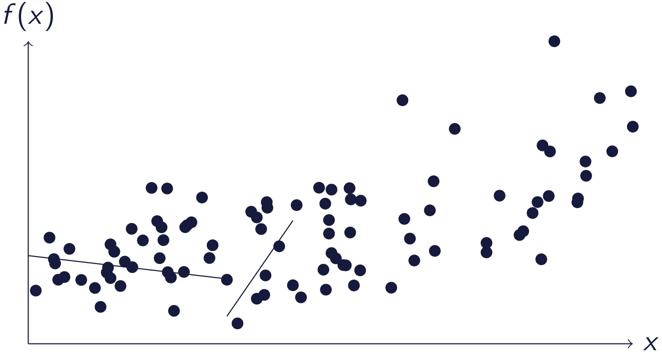
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

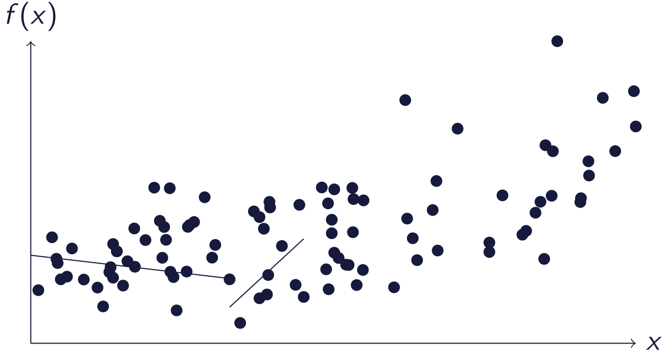
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

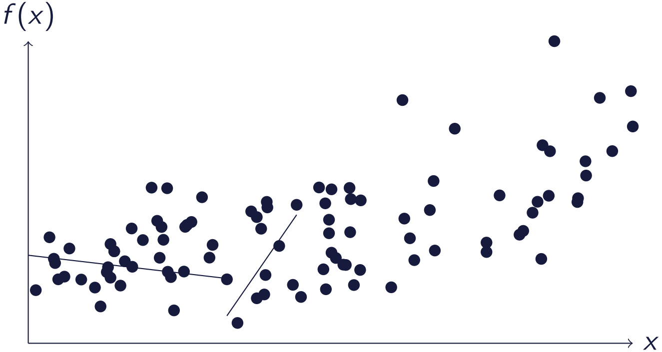
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

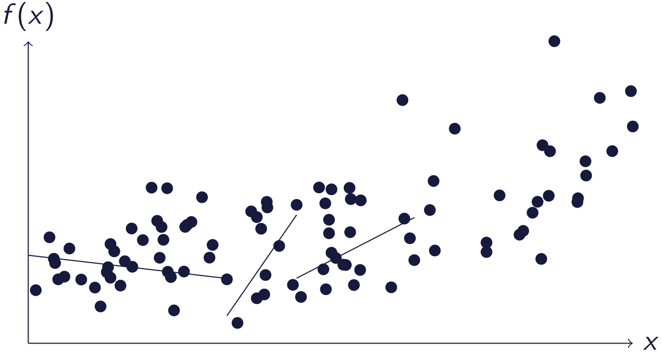
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \le \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

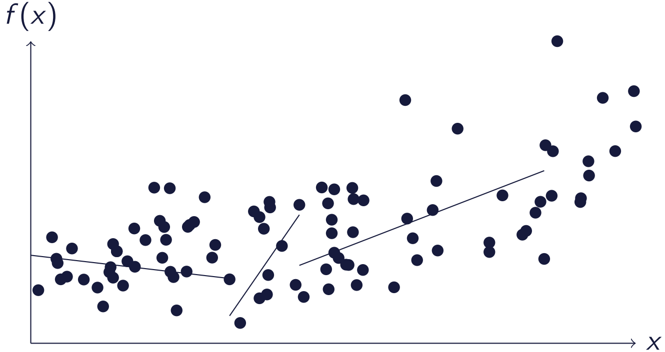
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

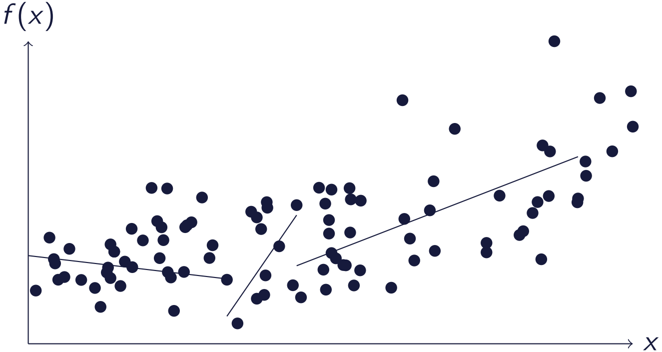
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

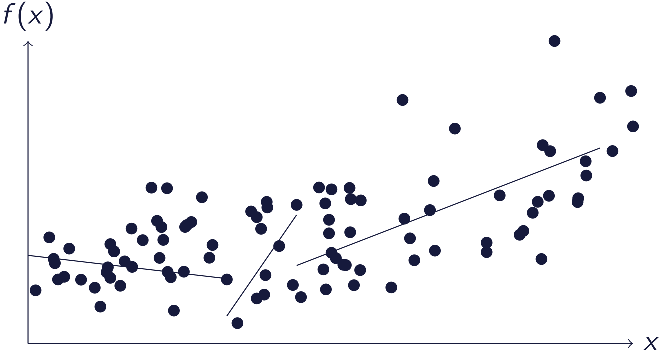
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

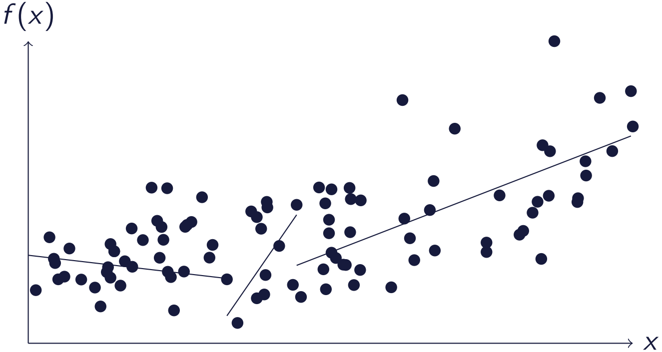
- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Multi-Bucket Construction

We place bucket boundaries by using the single-bucket construction

- the first bucket start is x_1
- the estimation error grows monotonic with the bucket size
- use binary search to find the maximum size such that $l_q \leq \epsilon$
- construct additional buckets until domain is covered
- perform binary search over ϵ until space budget is met



Overview

1. Motivation
2. Bounding the Impact of Cardinality Estimation Errors
3. Constructing Synopses
4. **Evaluation**
5. Conclusion

Evaluation

We compared different approaches:

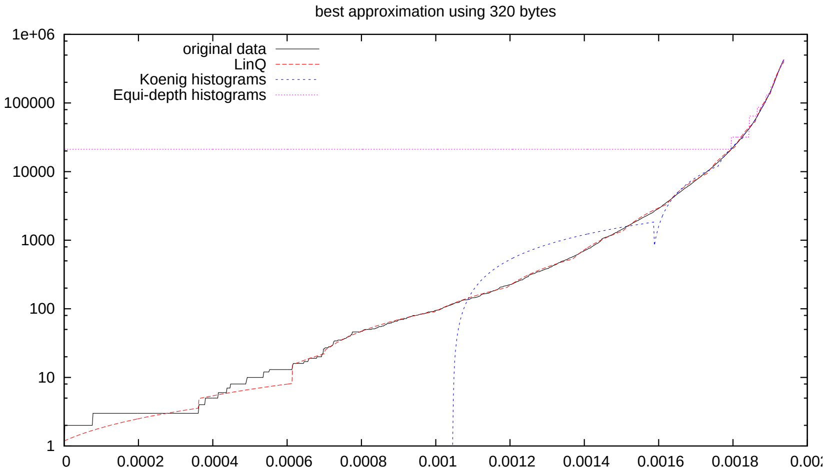
- piecewise LinQ (our approach)
- V-Optimal histograms
- Wavelets
- Sampling
- Koenig histograms (piecewise linear, fits l_2)
- equi-depth histograms

All were given the same space budget

Data set: CDF of TF*IDF*PageRank scores from TREC-12

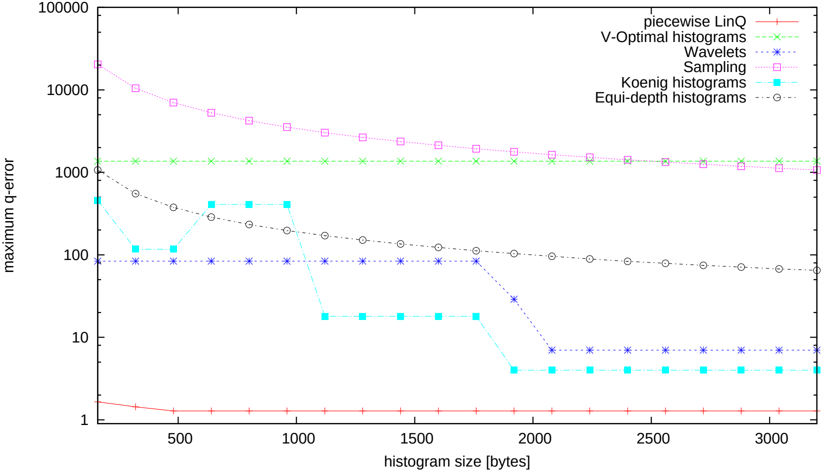
- 427,940 data points
- more experiments in the paper

Fit with a given Space Budget



Approximation error of other approaches varies greatly over the domain.

Q-Errors of Different Approaches



Note the logarithmic scale and that the plan costs are bound by q^4!

Conclusion

For cardinality estimation l_q is a much more meaningful metric

- if the q-error can be bound suitably **optimality is guaranteed**
- if the bounds cannot be met, the **cost degradation is bound** by q^4

We developed a synopsis construction algorithm for minimizing l_q

- fast iterative algorithm for single-bucket construction
- can fit a wide class of functions
- efficient multi-bucket construction using binary search
- provides much better approximations than other approaches
- in particular, more **robust**, plan quality is bound!