Robust Query Driven Cardinality Estimation Under Changing Workloads

Negi et al., 2023
**Background**

Query optimization

**Cardinality**: # of rows returned by a query or sub-query

Use estimates instead of real cardinality values

- Efficient

Good estimates = Good query plans

Bad estimates = Bad query plans

<table>
<thead>
<tr>
<th></th>
<th>TRUE</th>
<th>Est</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Plan 1</th>
<th>Plan 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>$</td>
<td>A \times B</td>
</tr>
</tbody>
</table>
Background

Traditionally done by histograms (or sampling)

Pros:
- Efficient
- Cheap

Cons:
- Uniformity assumption
- Independence assumption
Learned Cardinality Estimation

Machine learning (ML) based cardinality estimation

Data driven
- Learn from underlying data

Query driven
- Learn from past queries
Data Driven Cardinality Estimation

Model joint distribution over all attributes in the database

+ No simplifying assumptions
+ Robust
- Large model size
- Slow inference time
- Does not support all kinds of queries
Query Driven Cardinality Estimation

Learn regression model which map queries to their corresponding cardinalities from past workloads

+ Lightweight
+ Fast
+ Extends to all kinds of join patterns and filters
- Do not generalize to workload drift
- Need large number of queries to retrain the model
Workload Drift

New or changing workloads

- New filters
- New columns
- New tables
- Data updates
Robust Query Driven Cardinality Estimation

Emulate workload drift *during training*

Use initial cardinality estimate from DBMS

*Correct* this initial cardinality estimate
Query Representation

Need to represent (featurize) a query in a compatible format for the neural network

Query Features

- Tables, joins, columns
- One-hot vectors

Data Features

- DBMS cardinality estimate from the query
Emulating Workload Drift

Query masking

- Rely more on data features
  - Randomly zero out each query feature with probability $p$
  - Similar to dropout technique in ML

Dropout — Srivastava et al. (2014)
Capturing Correlations

Join bitmap

- Capture correlations across tables
  - Sample on primary key
  - Correlated sample on foreign key
  - Apply filters to the correlated samples
  - One bitmap vector

(a) Example training query.

(b) Example evaluation query.
Handling Data Updates

Shuffling bitmap

- Capture join correlations for updated data
  - Shuffle the bitmap indices at each step
  - The indices lose their meaning
  - Can instead distinguish between broad patterns
    - Many or few rows being selected

Use static bitmaps for non-update scenarios
Limitations

Query masking

- Changing data correlations reduces the effect of query masking

Figure 9: Data drift for the correlated gaussians. (a) presents the training data, $corr = 0.9$. (b) shows a drift, $corr = -0.5$. 
Results

Compares two learned models

- Multi Set Convolutional Networks (MSCN)
- Robust-MSCN (authors)
Results

Use workloads based on IMDb dataset

Training

- Join Order Benchmark (JOB)
- Cardinality Estimation Benchmark (CEB)
- JOBLight-train

Evaluation

- JOB
- CEB
Results

Baselines the models are compared against

- True cardinalities
  - Represents the best query runtime performance
- PostgreSQL estimates
  - Also provided as data features to the models
- Microsoft SQL-Server estimates
  - Represents the best traditional estimator
  - More sophisticated than PostgreSQL
Results

Figure 10: Runtime performance on PostgreSQL of baselines, and models trained on CEB, JOB, or JOBLight-train.
Results

Figure 15: Models trained on old IMDb versions and JOBLight-train workload. Evaluation is on full IMDb.
Figure 1: End to end query latencies of the MSCN model vs. our Robust-MSCN model in different workload drift scenarios. True cardinalities or PostgreSQL estimates are baselines.
Results

Figure 16: Ablation study. Each label (y-axis) is a difference from the Robust-MSCN model trained on JOBLight-train.
**Conclusion**

Robust-MSCN adapts gracefully to workload drift

- Still needs some retraining (periodically)

Anchored to the baseline DBMS performance

Never worse than PostgreSQL. Better than other query-driven approaches