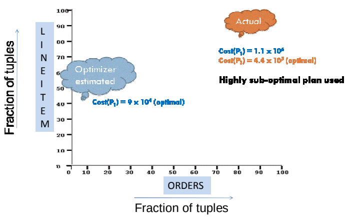
Identifying Robust Plans in PostgreSQL

Database Systems Laboratory
Indian Institute of Science

THE PROBLEM
Find a plan which is ROBUST against selectivity errors

Motivation




```
select C.custkey, C.name, C.acctbal, N.name
from Customer C, Orders O, Lineitem L, Nation N
where C.custkey = O.custkey and
      L.orderkey = O.orderkey and
      C.nationkey = N.nationkey and
      O.totalprice < 2833 and
      L.extendedprice < 28520
```

Simplified version of Q10 of the TPC-H benchmark

Issue:
In practice, compile-time predicate selectivity estimates are often significantly in error with respect to the actual values encountered during query execution.

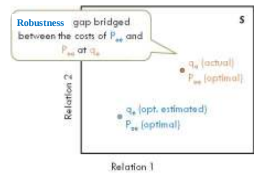
Reasons:
- Outdated statistics
- Attribute Value Independence assumption
- Coarse summaries …

OUTLINE OF OUR STRATEGY
Replace the cheapest plan with another plan on the basis of ROBUSTNESS

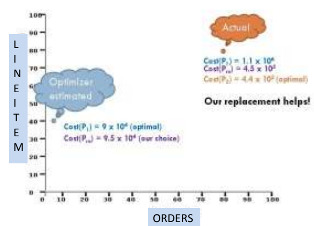
Plan Replacement

Aim for resistance, rather than cure!

- pick plans that perform reasonably well even in the presence of arbitrarily large selectivity errors, i.e., pick ROBUST plans



For each point in S, identify plans:
- Guaranteed to be near-optimal in the absence of selectivity error
- Likely to be comparatively stable across the entire selectivity space



HOW TO MEASURE ROBUSTNESS
1. How to evaluate Robustness of selected plan
2. How to estimate Robustness of a candidate plan

Robustness Metric

Selectivity Error Resistance Factor (SERF) – measures the performance gap bridged between P_{oe} and P_{oa} by P_{re}

$$SERF(q_e, q_a) = 1 - \frac{c(P_{re}, q_a) - c(P_{oa}, q_a)}{c(P_{oe}, q_a) - c(P_{oa}, q_a)}$$

SERF values can range from (- ∞ , 1]. Interpretation:
- (0, 1] – beneficial replacement
- [-λ_g, 0] – neither helps nor hurts
- (- ∞, -λ_g) – harmful replacement

Aggregate SERF (AggSERF) – captures the aggregate impact of plan replacements in the entire selectivity space

$$AggSERF = \frac{\sum_{q_e \in rep(S)} \sum_{q_a \in ezo_{oe}(S)} SERF(q_e, q_a)}{\sum_{q_e \in S} \sum_{q_a \in ezo_{oe}(S)} 1}$$

Estimating Robustness

SERF can be used to evaluate the Robustness of a selected plan, but to estimate potential robustness of a candidate plan, we use Benefit Index.

$$\text{Benefit Index}(candidate\ plan) = \frac{sum\ of\ corner\ costs\ of\ cheapest\ plan}{sum\ of\ corner\ costs\ of\ candidate\ plan}$$

Corner costs: cost of plan at corners of the selectivity space.
Note: only candidate plans with BI >1 are useful

Robust Plan Selection Problem

Input: An SQL Query with one or more error-sensitive relations and Cost increase thresholds – λ_l, λ_g

Implement a plan replacement strategy such that:

1. $\frac{c(P_{re}, q_e)}{c(P_{oe}, q_e)} \leq (1 + \lambda_l)$
2. $\forall q_a \in$ S s.t. $q_a \neq q_e$, $\frac{c(P_{re}, q_a)}{c(P_{oe}, q_a)} \leq (1 + \lambda_g)$, or equivalently, MinSERF $\geq -\lambda_g$.
3. The contribution to the AggSERF metric is maximized.

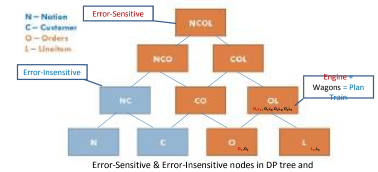
HOW TO IDENTIFY ROBUST PLANS
A modification to the dynamic programming routine to identify Robust plans

EXPAND algorithm

At error-sensitive nodes of the DP-lattice, retain robust candidates along with the cheapest plan

Two aspects:
- Plan enumeration (to generate candidates)
- Plan retention (retain only a useful subset)

} Plan Expansion



Error-Sensitive & Error-Insensitive nodes in DP tree and the Plan Enumeration Process

Plan Retention:
Four-stage pruning mechanism at each error-sensitive node in the DP-lattice:
- Local Cost Check (eliminates wagons with unreasonable local cost)
- Global Safety Check (ensures reasonable cost across the space)
- Global Benefit Check (ensures only beneficial wagons are retained)
- Cost-Safety-Benefit Skyline Check (eliminates redundant wagons)

Final Selection:
At Root node, choose the plan with maximum Benefit Index.

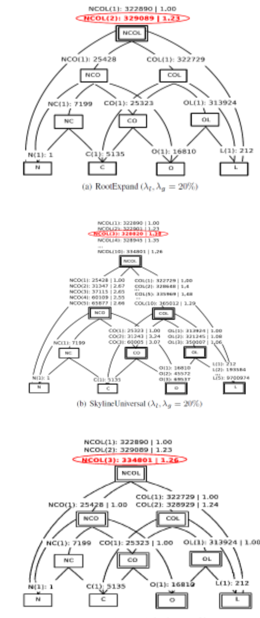
EXPAND – family of algorithms:
A number of replacement algorithms possible by different choices of λ_l^x and λ_g^x

Optimization Algorithm	Leaf Node λ_l^x, λ_g^x	Internal Node λ_l^x, λ_g^x	Root Node λ_l^x, λ_g^x	Root Node δ_g
Standard DP	0	0	0	–
RootExpand	0	0	λ_l, λ_g	≥ 1
NodeExpand	λ_l, λ_g	λ_l, λ_g	λ_l, λ_g	≥ 1
SkylineUniversal	∞	∞	λ_l, λ_g	≥ 1

ROBUSTNESS-RESOURCES TRADEOFF
The choices differ in *amount* of Plan Expansion.

Algorithmic Choices

We can ask for more Robustness if we are willing to spend more resources. So, there is tradeoff between performance & overheads

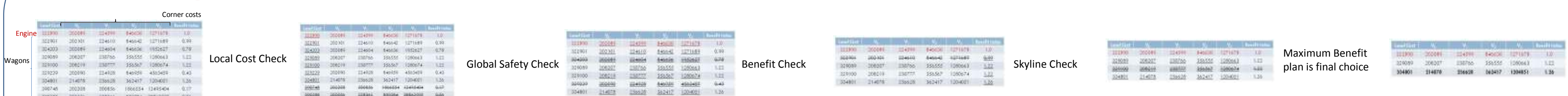


(a) RootExpand ($\lambda_l, \lambda_g = 20\%$)

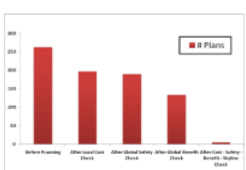
(b) SkylineUniversal ($\lambda_l, \lambda_g = 20\%$)

(c) NodeExpand ($\lambda_l, \lambda_g = 20\%$)

Worked out example



Pruning Example

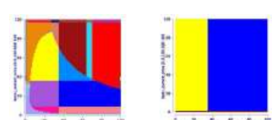


Performance Study:

~40% replacement
~80% replacement
Good Stability
Very good Help%

Query Template	RootExpand REP %	RootExpand Agg SERF	RootExpand Max SERF	RootExpand Help %	NodeExpand REP %	NodeExpand Agg SERF	NodeExpand Max SERF	NodeExpand Help %	SkylineUniversal REP %	SkylineUniversal Agg SERF	SkylineUniversal Max SERF	SkylineUniversal Help %	SEER REP %	SEER Agg SERF	SEER Max SERF	SEER Help %
QT5	84	0.54	1	55	85	0.54	1	55	85	0.54	1	55	47	0.61	1	64
QT8	42	0.11	1	1	84	0.13	1	3	–	–	–	–	39	-0.09	1	1
QT10	32	0.20	1	19	98	0.21	1	20	98	0.21	1	20	37	0.21	1	20
AIQT5	87	0.37	1	36	99	0.37	1	38	–	–	–	–	87	0.38	1	39
3DQT8	47	0.17	1	16	69	0.18	1	18	–	–	–	–	59	0.17	1	18
3DQT10	30	0.37	1	67	99	0.39	1	71	99	0.39	1	71	24	0.36	1	41
AI3DQT8	30	0.18	1	21	98	0.19	1	21	–	[illegible]	[illegible]	–	55	0.12	1	15
AI3DQT10	30	0.11	1	13	99	0.13	1	19	–	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1	13
DSQT7	93	0.28	1	28	93	0.28	1	28	93	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1	28
DSQT18	12	0.31	1	33	58	0.48	1	49	–	–	–	–	57	0.48	1	49
DSQT26	30	0.48	1	50	30	0.49	1	50	30	0.48	1	49	29	0.49	1	49
AIDSQT18	11	0.03	1	3	75	0.07	1	5	–	–	–	–	68	0.04	1	8

NodeExpand's and RootExpand's performance competitive with SEER, SkylineUniversal



(a) DP: 28 plans

(c) NodeExpand: 3 plans

Anorexia is additional benefit

Conclusion:

1. We introduced robustness in cost-based optimizers with the intention of minimizing the impact of selectivity errors
2. Our results show that a significant degree of robustness can be obtained with relatively minor conceptual changes to current optimizers
3. NodeExpand proved to be an excellent all-round choice, simultaneously delivering good robustness, anorexic plan diagrams and acceptable computational overheads