Depth Estimation for Ranking Query Optimization

Karl Schnaitter, UC Santa Cruz
Joshua Spiegel, BEA Systems, Inc.
Neoklis Polyzotis, UC Santa Cruz

Relational Ranking Queries

```
SELECT h.hid, r.rid, e.eid
FROM Hotels h, Restaurants r, Events e
WHERE h.city = r.city AND r.city = e.city
RANK BY 0.3/h.price + 0.5*r.rating + 0.2*isMusic(e)
LIMIT 10
```

- A *base score* for each table in [0,1]

Hotels: b_H(h) = 1/h.price

Restaurants: b_R(r) = r.rating

Events: b_E(e) = isMusic(e)

- Combined with a *scoring function S*

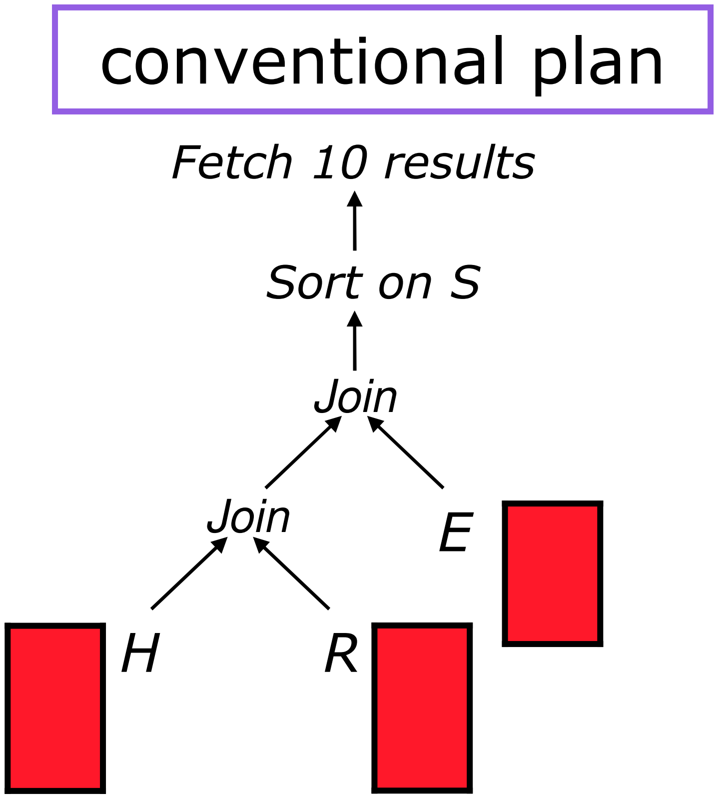
$$S(b_H, b_R, b_E) = 0.3*b_H + 0.5*b_R + 0.2*b_E$$

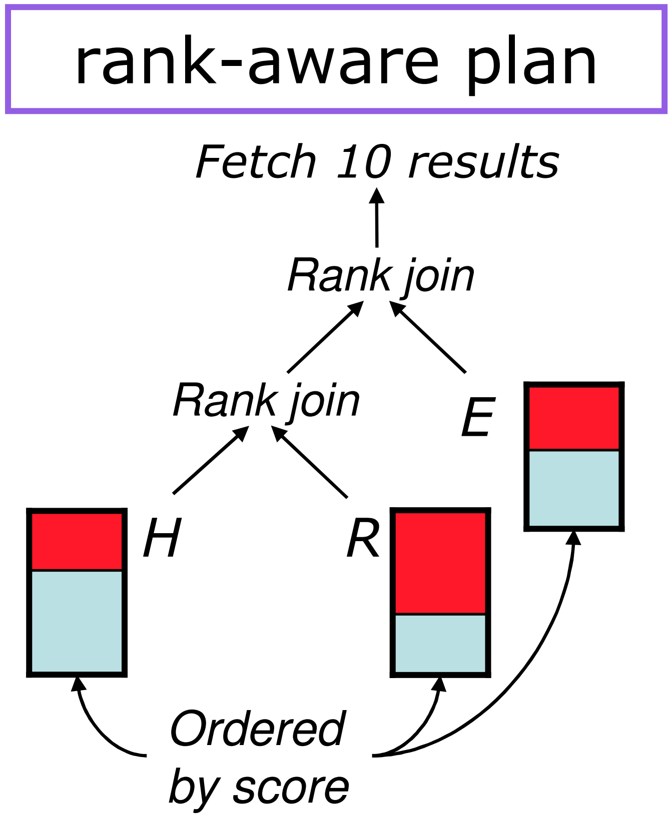
- Return top *k* results based on *S*

In this case, $k = 10$

Ranking Query Execution

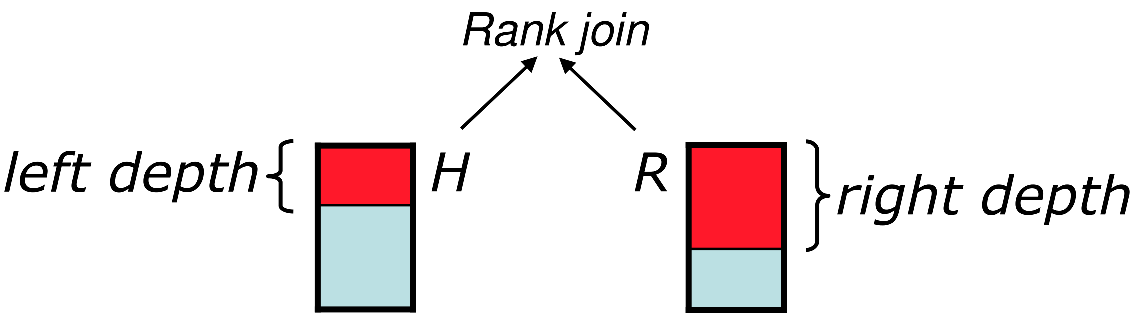
```
SELECT h.hid, r.rid, e.eid
FROM Hotels h, Restaurants r, Events e
WHERE h.city = r.city AND r.city = e.city
RANK BY 0.3/h.price + 0.5*r.rating + 0.2*isMusic(e)
LIMIT 10
```





Depth Estimation

- *Depth*: number of accessed tuples
  - Indicates execution cost
  - Linked to memory consumption



- *The problem:* Estimate depths for each operator in a rank-aware plan

Depth Estimation Methods

- Ilyas et al. (SIGMOD 2004)
  - Uses probabilistic model of data
  - Assumes relations of equal size and a scoring function that sums scores
  - Limited applicability
- Li et al. (SIGMOD 2005)
  - Samples a subset of rows from each table
  - Independent samples give a poor model of join results

Our Solution: DEEP

- **DE**pth **E**stimation for **P**hysical plans
- Strengths of DEEP
  - A principled methodology
    - Uses statistical model of data distribution
    - Formally computes depth over statistics
  - Efficient estimation algorithms
  - Widely applicable
    - Works with state-of-the-art physical plans
    - Realizable with common data synopses

Outline

- Preliminaries
- DEEP Framework
- Experimental Results

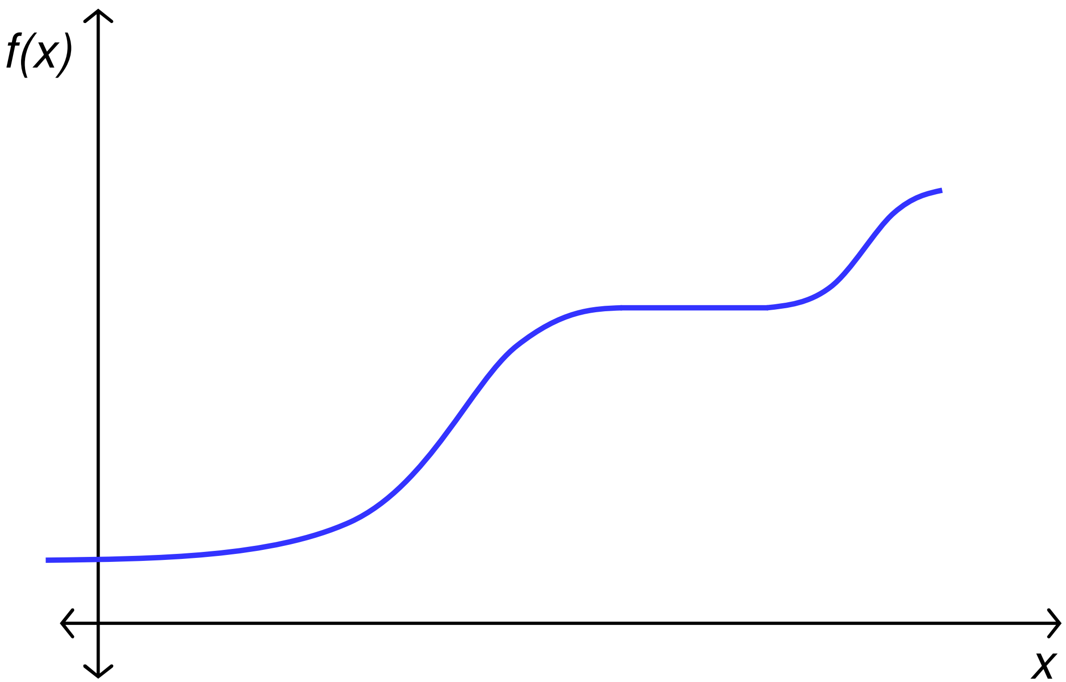
Outline

- Preliminaries
- DEEP Framework
- Experimental Results

Monotonic Functions

- A function $f(x_1,\ldots,x_n)$ is *monotonic* if

  ✂$i(x_i \leq y_i)$ ⬇ $f(x_1,\ldots,x_n) \leq f(y_1,\ldots,y_n)$



Monotonic Functions

- A function $f(x_1,\ldots,x_n)$ is *monotonic* if

  $\forall i (x_i \leq y_i) \Rightarrow f(x_1,\ldots,x_n) \leq f(y_1,\ldots,y_n)$

- Most scoring functions are monotonic
  - E.g. sum, product, avg, max, min
- Monotonicity enables bound on score
  - In example query, score was

    0.3/h.price + 0.5*r.rating + 0.2*isMusic(e)
  - Given a restaurant *r*, upper bound is

    0.3*1 + 0.5*r.rating + 0.2*1

Hash Rank Join [IAE04]

- The *Hash Rank Join* algorithm
  - Joins inputs sorted by score
  - Returns results with highest score
- Main ideas
  - Alternate between inputs based on *pull strategy*
  - Score bounds allow early termination

Bound: 1.8

L	a	b_L
	x	1.0
	y	0.8
	. . .	

Bound: 1.7

R	a	b_R
	y	1.0
	z	0.9
	w	0.7
	. . .	

Query: Top result from $L \bowtie R$ with scoring function $S(b_L, b_R) = b_L + b_R$

Result: y
Score: 1.8

HRJN* [IAE04]

- The HRJN* pull strategy:
  a) Pull from the input with highest bound
  b) If (a) is a tie, pull from input with the smaller number of pulls so far
  c) If (b) is a tie, pull from the left

Bound: ~~2.0~~ 1.8

L	a	b_L
	x	1.0
	y	0.8

Bound: ~~2.0~~ ~~1.9~~ 1.7

R	a	b_R
	y	1.0
	z	0.9
	w	0.7

Query: Top result from $L \bowtie R$ with scoring function $S(b_L, b_R) = b_L + b_R$

Result: y
Score: 1.8

Outline

- Preliminaries
- DEEP Framework
- Experimental Results

Supported Operators

Evidence in favor of HRJN*

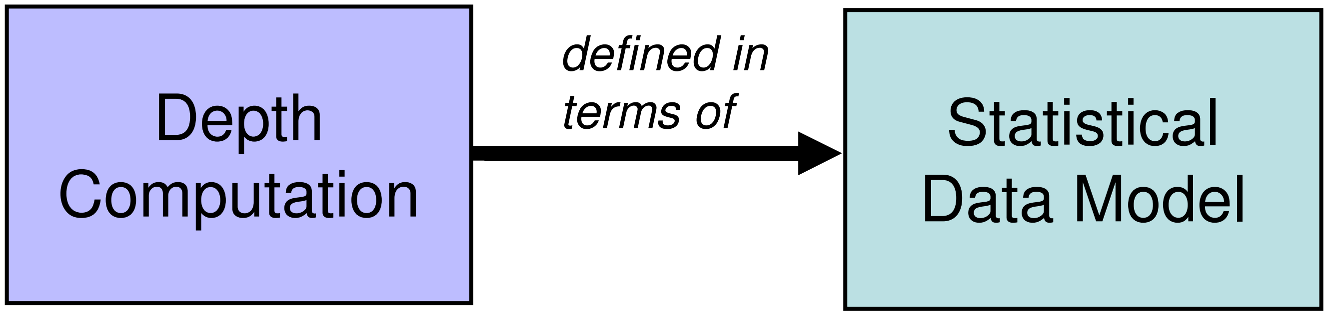
- Pull strategy has strong properties
  - Within constant factor of optimal cost
  - Optimal for a significant class of inputs
  - More details in the paper
- Efficient in experiments [IAE04]

DEEP explicitly supports HRJN*

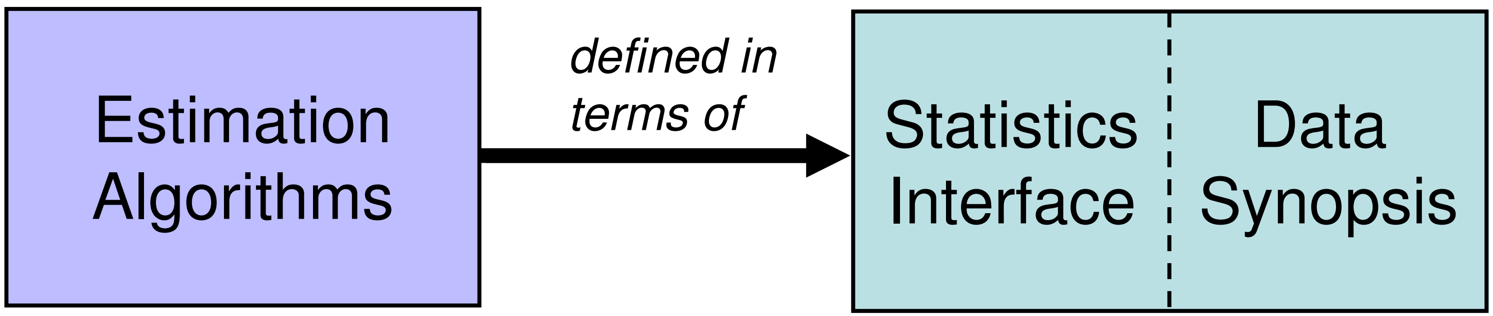
- Easily extended to other join operators
- Selection operators too

DEEP: Conceptual View

Formalization



Implementation



Statistics Model

- Statistics yield the distribution of scores for base tables and joins

F_L

b_L	$F_L(b_L)$
1.0	5
0.9	2
0.8	3
0.6	12
0.4	8

F_R

b_R	$F_R(b_R)$
1.0	3
0.7	1
0.5	2

$F_{L \bowtie R}$

b_L	b_R	$F_{L \bowtie R}(b_L,b_R)$
1.0	1.0	6
1.0	0.5	4
1.0	0.7	3
0.9	0.7	2
0.6	0.7	2

Statistics Interface

- DEEP accesses statistics with two methods
  - *getFreq*(**b**): Return frequency of **b**
  - *nextScore*(**b**,*i*): Return next lowest score on dimension *i*

	b_L	b_R	$F_{L \bowtie R}(b_L,b_R)$
	1.0	1.0	6
	1.0	0.5	4
b	1.0	0.7	3
	0.9	0.7	2
	0.6	0.7	2

getFreq(**b**) = 3
nextScore(**b**,1)=0.9
nextScore(**b**,2)=0.5

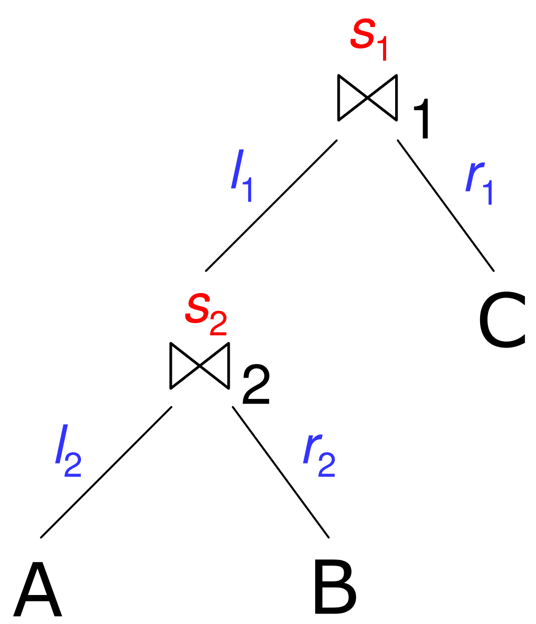
- The interface allows for efficient algorithms
  - Abstracts the physical statistics format
  - Allows statistics to be generated on-the-fly

Statistics Implementation

- Interface can be implemented over common types of data synopses
- Can use a histogram if
  a) Base score function is invertible, or
  b) Base score measures distance
- Assume uniformity & independence if
  a) Base score function is too complex, or
  b) Sufficient statistics are not available

Depth Estimation Overview

Top-*k* query plan



Estimates made	Value
1. Score of the k^{th} best tuple out of $\bowtie_1$	s_1
2. Depths of $\bowtie_1$ needed to output score of s_1	l_1 and r_1
3. Score of the l_1^{th} best tuple out of $\bowtie_2$	s_2
4. Depths of $\bowtie_2$ needed to output score of s_2	l_2 and r_2

Estimating Terminal Score

- Suppose we want the 10th best score

- Idea
  - Sort by total score
  - Sum frequencies

b_L	b_R	$F_{L \bowtie R}(b_L,b_R)$
1.0	1.0	6
1.0	0.5	4
1.0	0.7	3
0.9	0.7	2
0.6	0.7	2

$b_L + b_R$	$F_{L \bowtie R}(b_L,b_R)$	sum
2.0	6	6
1.7	3	9
$\boxed{1.6}$	2	11
1.5	4	
1.3	2	

$S_{\text{term}} = 1.6$

Estimation Algorithm

- Idea: Only process necessary statistics

b_L	b_R	$F_{L \bowtie R}(b_L, b_R)$
1.0	1.0	6
1.0	0.5	4
1.0	0.7	3
0.9	0.7	2
0.6	0.7	2

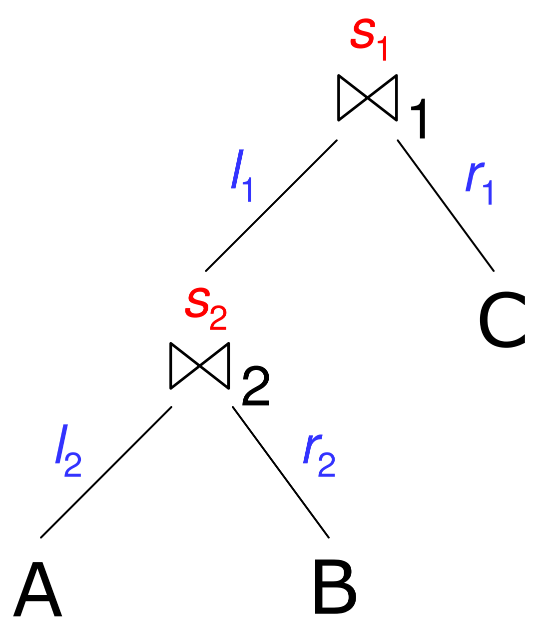
	1	**0.7**	**0.5**
1	6	3	4
0.9		$\boxed{2}$	
0.8			
0.6		2	

$S_{term} = 1.6$

- Algorithm relies solely on *getFreq* and *nextScore*
  - Avoids materializing complete table
- Worst-case complexity equivalent to sorting table
  - More efficient in practice

Depth Estimation Overview

Top-k query plan

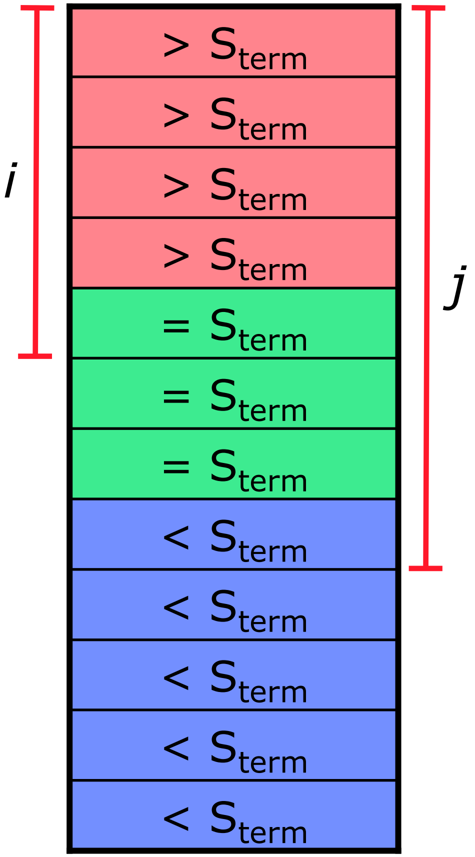


Estimates made	Value
1. Score of the k^{th} best tuple out of $\bowtie_1$	s_1
2. Depths of $\bowtie_1$ needed to output score of s_1	l_1 and r_1
3. Score of the $l_1{}^{th}$ best tuple out of $\bowtie_2$	s_2
4. Depths of $\bowtie_2$ needed to output score of s_2	l_2 and r_2

Estimating Depth for HRJN*

Theorem: $i \leq$ depth of HRJN* $\leq j$

Input Score Bounds



Example: $S_{term} = 1.6$

b_L	$F_L(b_L)$
1.0	5
0.9	2
0.8	3
0.6	12
0.4	8

$\rightarrow$

$b_L + 1$	$F_L(b_L)$
2.0	5
1.9	2
1.8	3
1.6	4
1.4	8

$11 \leq$ depth ≤ 15

- Estimation algorithm
  - Access via *getFreq* and *nextScore*
  - Similar to estimation of S_{term}

Outline

- Preliminaries
- DEEP Framework
- Experimental Results

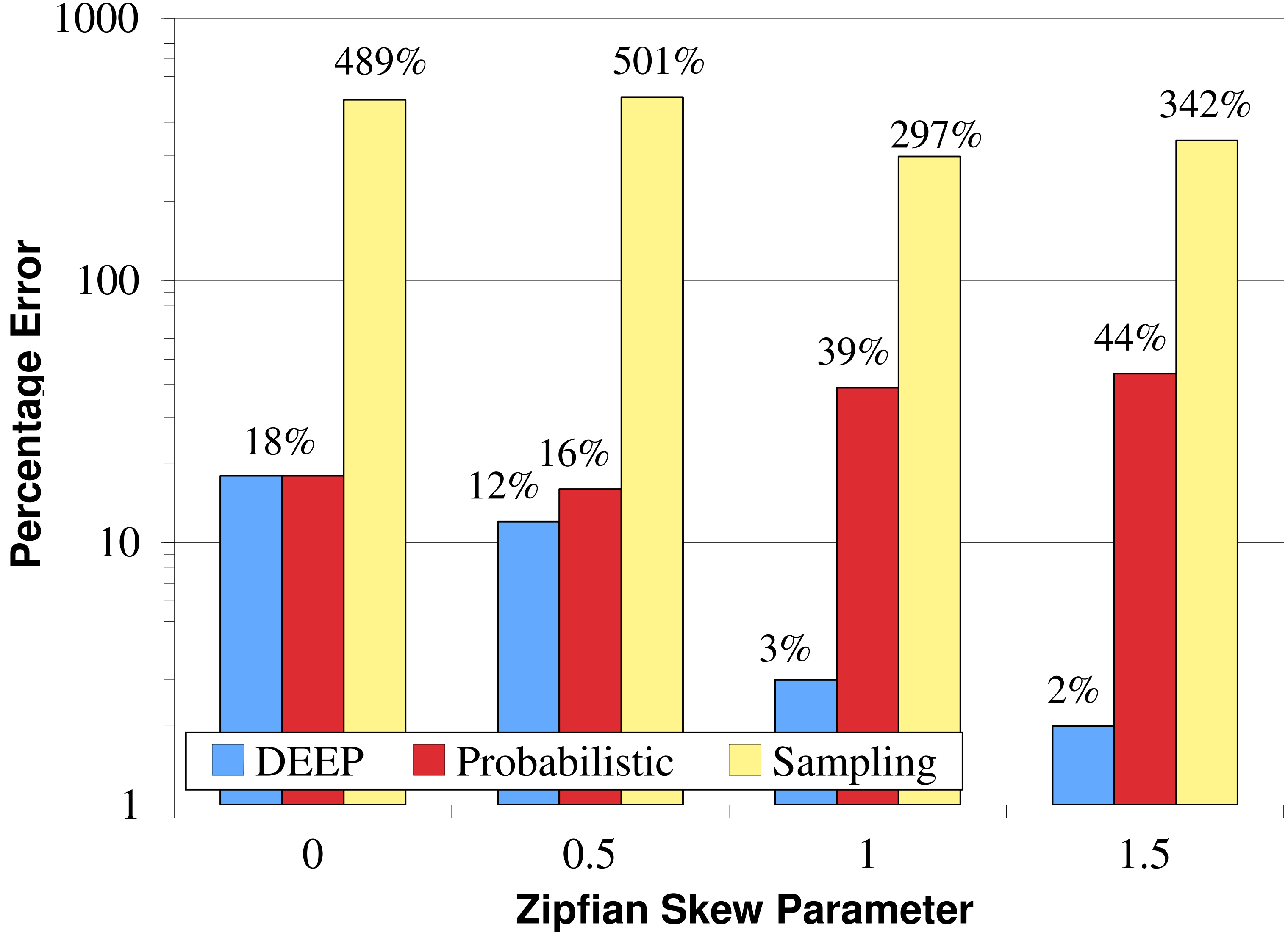
Experimental Setting

- TPC-H data set
  - Total size of 1 GB
  - Varying amount of skew
- Workloads of 250 queries
  - Top-10, top-100, top-1000 queries
  - One or two joins per query
- Error metric: *absolute relative error*

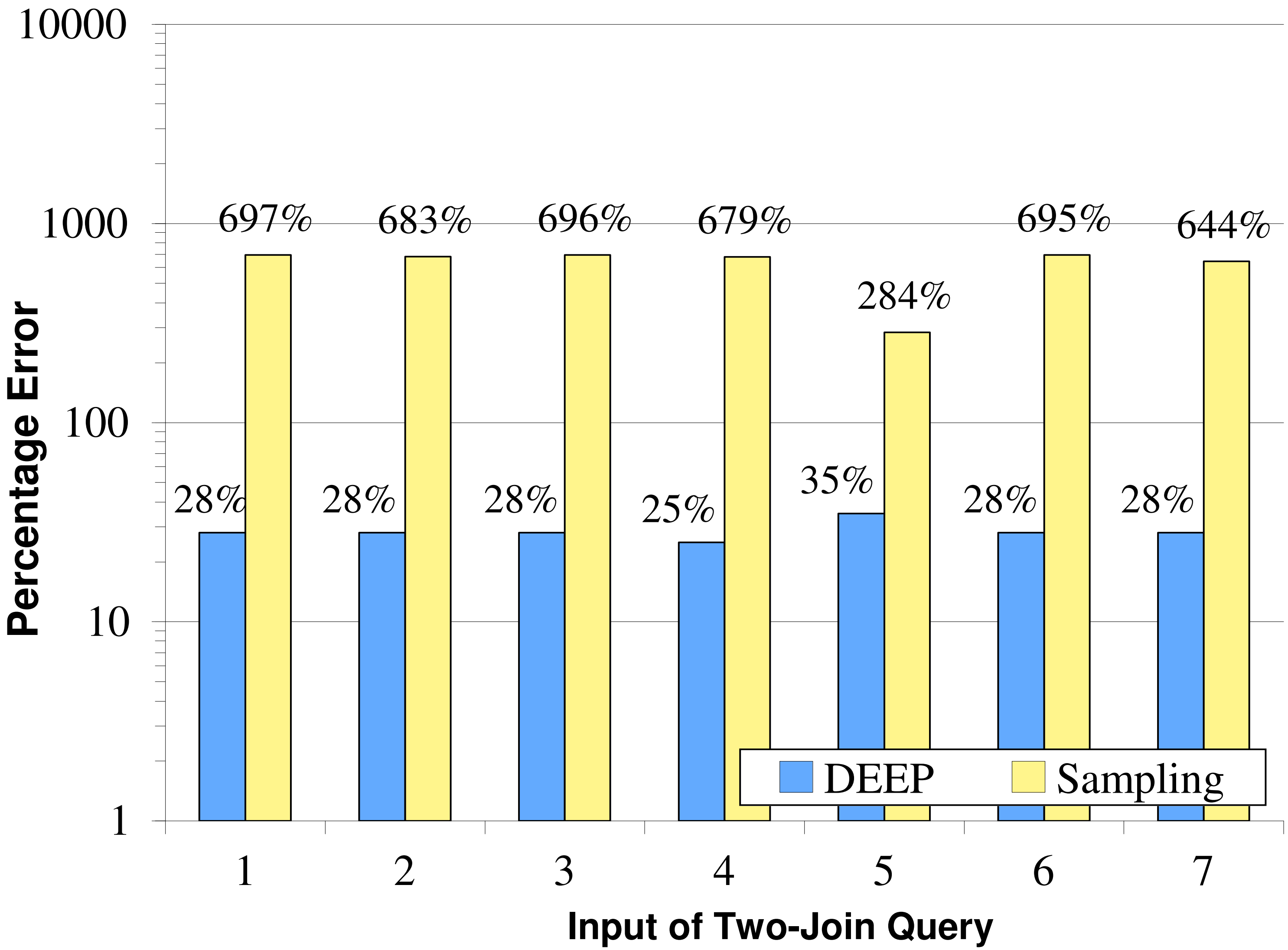
Depth Estimation Techniques

- DEEP
  - Uses 150 KB TuG synopsis [SP06]
- Probabilistic [IAE04]
  - Uses same TuG synopsis
  - Modified to handle single-join queries with varying table sizes
- Sampling [LCIS05]
  - 5% sample = 4.6 MB

Error for Varying Skew



Error at Each Input



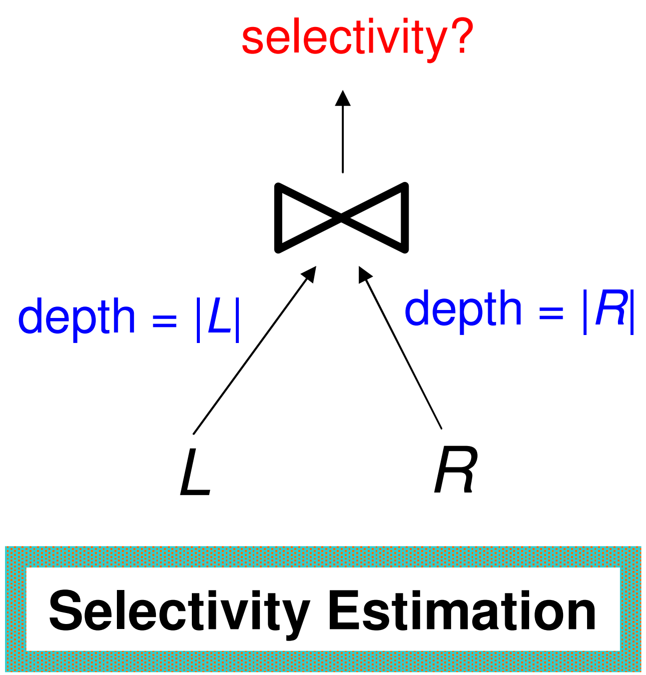
Conclusions

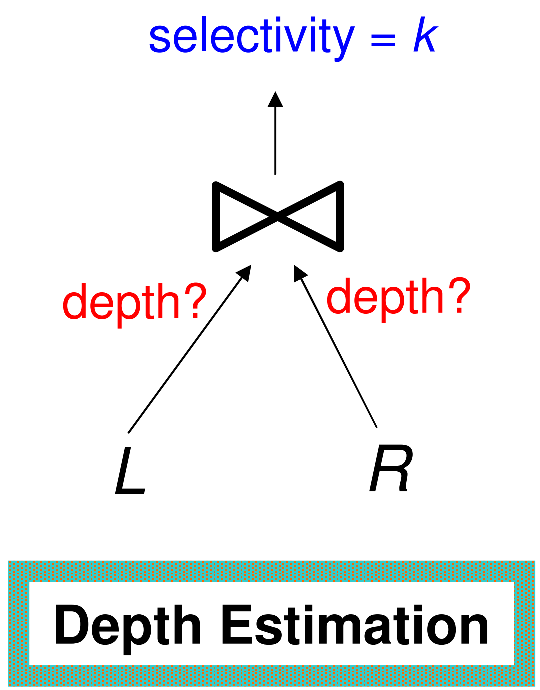
- Depth estimation is necessary to optimize relational ranking queries
- DEEP is a principled and practical solution
  - Takes data distribution into account
  - Applies to many common scenarios
  - Integrates with data summarization techniques
- New theoretical results for HRJN*
- Next steps
  - Accuracy guarantees
  - Data synopses for complex base scores (especially text predicates)

Thank You

Related Work

- Selectivity estimation is a similar idea
- It is the inverse problem





Other Features

- DEEP can be extended to NLRJ and selection operators
- DEEP can be extended to other pulling strategies
  - Block-based HRJN*
  - Block-based alternation

Analysis of HRJN*

- Within the class of all HRJN variants:
  - HRJN* is optimal for many cases
    - With no ties of score bound between inputs
    - With no ties of score bound within one input
  - HRJN* is instance optimal