# Approximate String Joins

Divesh Srivastava
AT&T Labs-Research

---

# The Need for String Joins

- Substantial amounts of data in existing RDBMSs are strings
- There is a need to correlate data stored in different tables
  - Applications: data cleaning, data integration
- Example: Find common customers across different services

| Service A |
| --- |
| Divesh Srivastava |
| John Paul McDougal |
| KIA International |
| Euroaft Corp. |
| Hatronic Inc. |
| Comp. Sci. Dept. |
| … |

| Service B |
| --- |
| Eurodraft Corp. |
| Hatronic Corp. |
| Euroaft |
| Divesh Shrivastava |
| John P. McDougal |
| Dept. of Comp. Sci. |
| KIA |

# Problems with Exact String Joins

- Typing mistakes, abbreviations, different conventions
- Standard equality joins do not "forgive such mistakes"

| Service A |
| --- |
| Divesh Srivastava |
| John Paul McDougal |
| KIA International |
| Euroaft Corp. |
| Hatronic Inc. |
| Comp. Sci. Dept. |
| … |

| Service B |
| --- |
| Eurodraft Corp. |
| Hatronic Corp. |
| Euroaft |
| Divesh Shrivastava |
| John P. McDougal |
| Dept. of Comp. Sci. |
| KIA |

=

∅
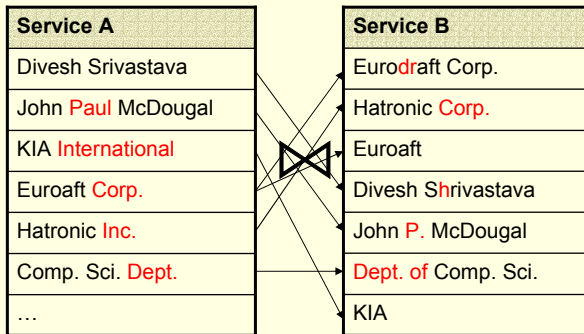
# Matching String Attributes

## Need for a similarity metric!

- Match entries with **typing mistakes**
  - Divesh Srivastava vs. Divesh Shrivastava

- Match entries with **abbreviations**
  - Euroaft Corporation vs. Euroaft Corp.

- Match entries with **different conventions**
  - Comp. Sci. Dept. vs. Dept. of Comp. Sci.

# Using String Edit Distance

**String Edit Distance:** Number of single character insertions, deletions, and modifications to transform one string to the other

| | | |
|---|---|---|
| Divesh Srivastava **-** Divesh Shrivastava | | → 1 |
| Dept. Comp. Sci. **-** Dept. of Comp. Sci. | | → 3 |
| Euroaft Corporation **-** Euroaft | | → 12 |

- **Good for** spelling errors, short word inserts/deletes
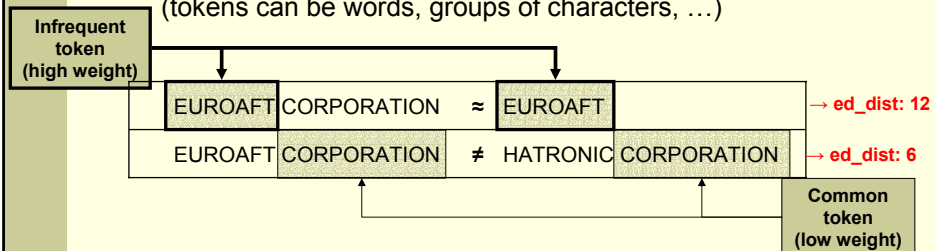- **Problems with** word order variations, long word inserts/deletes

---

# Using Cosine Similarity

Similar string pairs should share "infrequent" tokens (tokens can be words, groups of characters, …)

**Infrequent token (high weight)**

| EUROAFT | CORPORATION | ≈ | EUROAFT | → ed_dist: 12 |
|---|---|---|---|---|
| EUROAFT | CORPORATION | ≠ | HATRONIC CORPORATION | → ed_dist: 6 |

**Common token (low weight)**

$$\text{Similarity} = \sum_{\text{all tokens}} weight(token, t_1) * weight(token, t_2)$$

- **Good for** common token inserts/deletes, token order variations
- **Problems with** rare token inserts/deletes, rare misspellings

# Approximate String Joins

No native support for approximate string joins in RDBMSs

Two existing (straightforward) solutions:
- Join data outside of DBMS
- Join data via user-defined functions (UDFs) inside DBMS

# AS Joins Outside DBMS

1. Export data
2. Join outside of DBMS
3. Import the result

Main advantage:
> Can exploit specialized tools (e.g., address matching) and business rules, without restrictions from DBMS functionality

Disadvantages:
- Substantial amounts of data to be exported/imported
- Cannot be easily integrated with other DBMS processing steps

# AS Joins Using UDFs

1. Write a UDF to check if two strings match within distance K
2. Write an SQL statement that applies the UDF to the string pairs

```
SELECT R.sA, S.sA
FROM   R, S
WHERE  approx_string_match(R.sA, S.sA, K)
```

Main advantage:
    Easily integrated with other DBMS processing steps

Main disadvantage:
    Inefficient: UDF applied to entire cross-product of relations

# Our Approach: Use Q-grams

Intuition
- Similar strings have many common **substrings** (q-grams)
- Preprocess string data, generate auxiliary tables of substrings
- Perform "approximate string join", exploiting RDBMS capabilities of exact joins and aggregations

Advantages
- No modification of underlying RDBMS needed.
- Can leverage the RDBMS query optimizer.
- Much more efficient than the approach based on naive UDFs

# What is a Q-gram?

- **Q-gram**: A sequence of q characters of the original string

- Split each string into **all** overlapping q-grams
    - string with length L → **L + q - 1**  q-grams

- Example: q=3
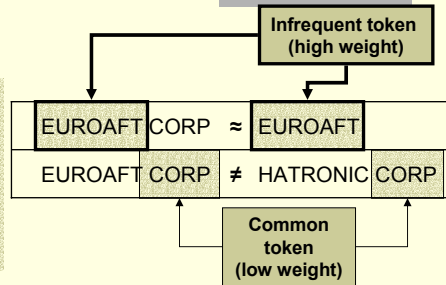    - srivastava = ##s, #sr, sri, riv, iva, vas, ast, sta, tav, ava, va$, a$$

# Q-grams and String Edit Distance

- Each edit distance operation affects at most **q** q-grams

- Two strings S1 and S2 with **string edit distance ≤ K** have at least **[max(S1.len, S2.len) + q - 1] – Kq** q-grams in common

- Example:
    - srivastava = ##s, #sr, sri, riv, iva, vas, ast, sta, tav, ava, va$, a$$
    - shrivastava = ##s, #sh, shr, hri, riv, iva, vas, ast, sta, tav, ava, va$, a$$

- Useful filter: eliminate all string pairs without "enough" common q-grams (no false dismissals!)

# Words and Cosine Similarity

Using **words** as tokens:

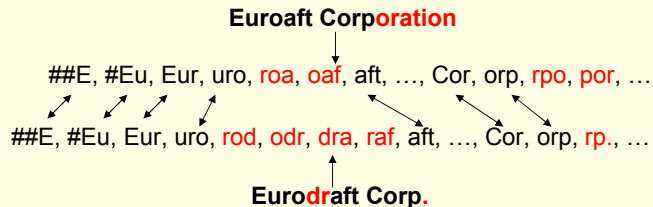- Split each entry into words
- Similar entries share infrequent words

| Infrequent token (high weight) |
| EUROAFT | CORP | ≈ | EUROAFT | |
| EUROAFT | CORP | ≠ | HATRONIC | CORP |

Common token (low weight)

- **Problems with** misspellings, abbreviations
  Euroaft Corporation ≠ Eurodraft Corp.

"WHIRL" – W.Cohen, SIGMOD'98

---

# Q-grams and Cosine Similarity

Use **q-grams** as tokens:

- Similar entries share many, infrequent q-grams

**Euroaft Corporation**

##E, #Eu, Eur, uro, roa, oaf, aft, …, Cor, orp, rpo, por, …

##E, #Eu, Eur, uro, rod, odr, dra, raf, aft, …, Cor, orp, rp., …

**Eurodraft Corp.**

- **Good for** common misspellings, abbreviations

# Matching Strings Efficiently in SQL

- Problem 1:
  Find all pairs of strings $t_1$, $t_2$ with **string edit distance ≤ K**

- Problem 2:
  Find all pairs of strings $t_1$, $t_2$ with **cosine similarity ≥ φ** where
  cosine similarity **= Σ** weight(token, $t_1$) * weight(token, $t_2$)
  token

- SQL-only solution desirable:
  - Scalability
  - Robustness
  - Ease of deployment

# Problem 1: Edit Distance in DBMS

- LENGTH FILTER: two strings S1 and S2 with **edit distance ≤ K** cannot differ in length by > K

- COUNT FILTER: two strings S1 and S2 with **edit distance ≤ K** have **≥ [max(S1.len, S2.len) + q - 1] – Kq** q-grams in common
  - Create auxiliary DBMS tables with tuples of the form:
    **<sid, qgram>,** join and aggregate these tables

- POSITION FILTER: corresponding q-grams of S1 and S2 cannot differ in their positions by more than **K**
  - Create auxiliary DBMS tables with tuples of the form:
    **<sid, qgram, pos>,** join and aggregate these tables

# Problem 1: The SQL Statement

```
SELECT    R1.sid, R2.sid
FROM      R1, R1Q, R2, R2Q
WHERE     R1.sid = R1Q.sid AND R2.sid = R2Q.sid AND
          R1Q.qgram = R2Q.qgram AND
          abs(R1Q.pos - R2Q.pos) <= k AND
          abs(LEN(R1.str) - LEN(R2.str)) <= k AND
          (LEN(R1.str)+q-1 > k*q OR LEN(R2.str)+q-1 > k*q)
GROUP BY  R1.sid, R2.sid, R1.str, R2.str
HAVING    COUNT(*) >= max(LEN(R1.str), LEN(R2.str))+q-1 - k*q AND
          edit_distance(R1.str, R2.str, k)


UNION ALL

SELECT    R1.sid, R2.sid
FROM      R1, R2
WHERE     LEN(R1.str)+q-1 <= k*q AND LEN(R2.str)+q-1 <= k*q AND
          abs(LEN(R1.str) - LEN(R2.str)) <= k AND
          edit_distance(R1.str, R2.str, k)
```
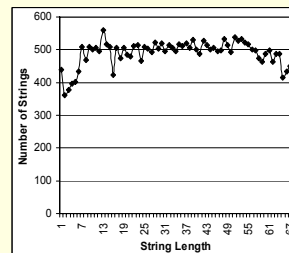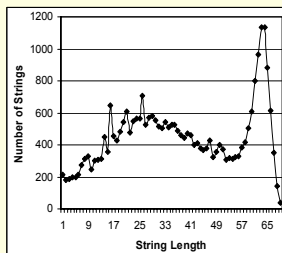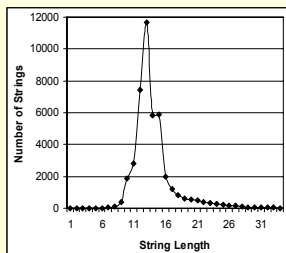
# Problem 1: Experimental Data

- Three customer data sets from AT&T Worldnet service
  - (a) set1 with about 40K strings
  - (b) set2 and (c) set3 with about 30K strings each

# Problem 1: DBMS Setup

- Used Oracle 8i (supports UDFs), on Sun 20 Enterprise Server

- Materialized the q-gram tables with entries <sid, qgram, pos> (less then 2 minutes per table)

- Tested configurations with and without indexes on the auxiliary q-gram tables (less than 5 minutes to generate each index)

The generation time for the auxiliary q-gram tables and indexes is small: even on-the-fly materialization is feasible

# Problem 1: RDBMS Query Plans

- Naive approach with UDFs: nested-loops joins (prohibitively slow even for small data sets)

- Q-gram approach: usually sort-merge joins

- In our prototype implementation, sort-merge joins is the fastest too

# Problem 1: Naïve UDFs vs. Q-grams



| | k=1 | k=2 | k=3 |
|---|---|---|---|
| ▪ Q1 (UDF only) | 1954 | 2028 | 2044 |
| ▫ Q4 (Filtering) | 48 | 68 | 91 |
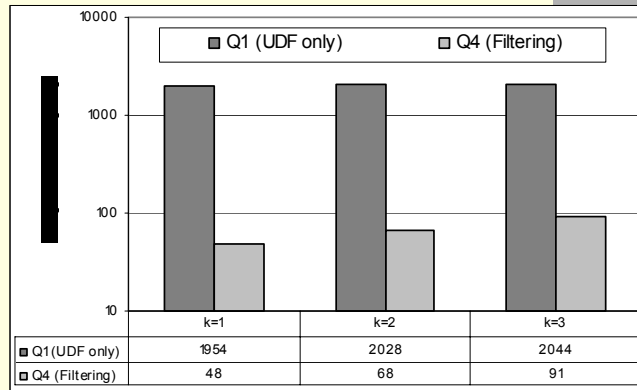
For a subset of set1, our technique was 20 to 30 times faster than the naïve use of UDFs

---

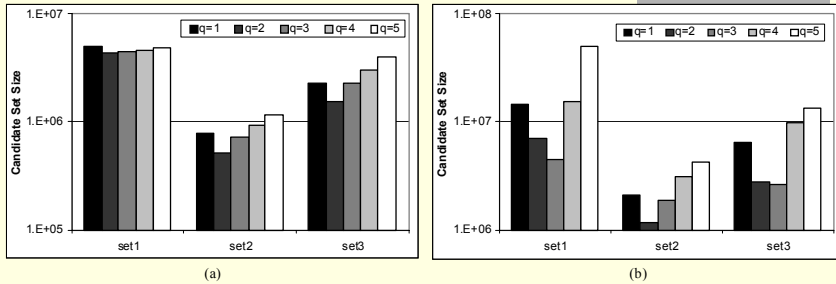# Problem 1: Effect of Filters
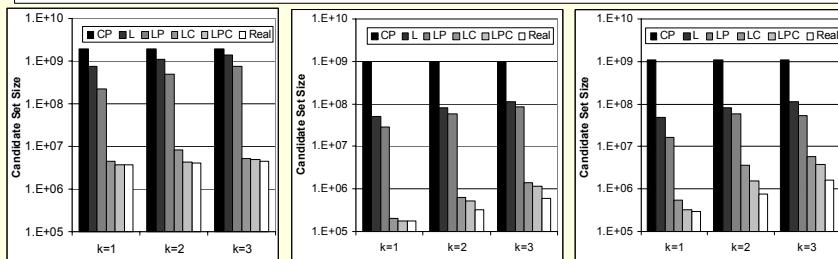


LENGTH FILTER: 40-70% reduction for set1 (small length deviation)

           90-98% reductions for set2, set3 (big length deviation)

+COUNT FILTER:  > 99% reduction

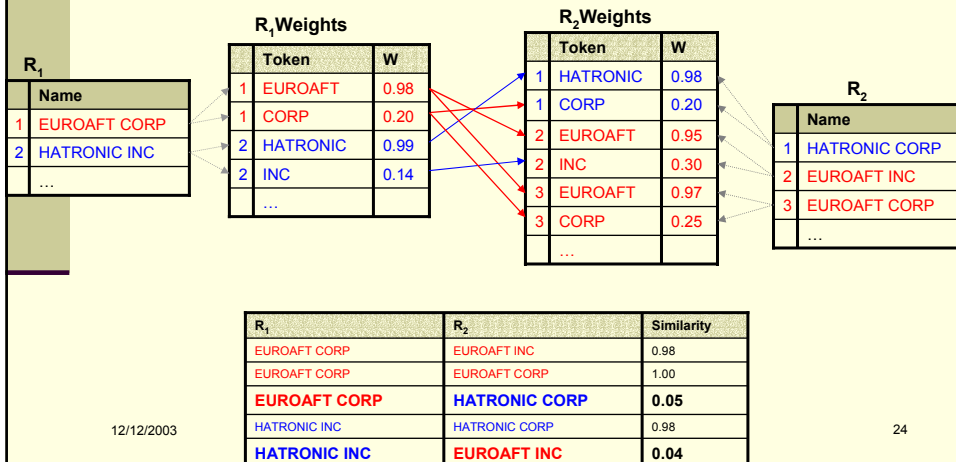+POSITION FILTER: ~ additional 50% reduction

# Problem 1: Effect of Q-gram Size



(a)

(b)

- For the given data sets, q=2 worked best
- q=2 is small enough to avoid, as much as possible, the space overhead for the auxiliary tables

---

# Problem 2: Cosine Similarity in DBMS

- Create **in SQL** relations $R_i$Weights (token weights from $R_i$)
- Compute similarity of **each tuple pair**



**$R_1$**

| | Name |
|---|---|
| 1 | EUROAFT CORP |
| 2 | HATRONIC INC |
| | … |

**$R_1$Weights**

| | Token | W |
|---|---|---|
| 1 | EUROAFT | 0.98 |
| 1 | CORP | 0.20 |
| 2 | HATRONIC | 0.99 |
| 2 | INC | 0.14 |
| | … | |

**$R_2$Weights**

| | Token | W |
|---|---|---|
| 1 | HATRONIC | 0.98 |
| 1 | CORP | 0.20 |
| 2 | EUROAFT | 0.95 |
| 2 | INC | 0.30 |
| 3 | EUROAFT | 0.97 |
| 3 | CORP | 0.25 |
| | … | |

**$R_2$**

| | Name |
|---|---|
| 1 | HATRONIC CORP |
| 2 | EUROAFT INC |
| 3 | EUROAFT CORP |
| | … |

| $R_1$ | $R_2$ | Similarity |
|---|---|---|
| EUROAFT CORP | EUROAFT INC | 0.98 |
| EUROAFT CORP | EUROAFT CORP | 1.00 |
| **EUROAFT CORP** | **HATRONIC CORP** | **0.05** |
| HATRONIC INC | HATRONIC CORP | 0.98 |
| **HATRONIC INC** | **EUROAFT INC** | **0.04** |

# Problem 2: Naïve SQL

**Computes similarity for many <span style="color:red">useless pairs</span>**

```
SELECT   r1w.tid AS tid1, r2w.tid AS tid2
FROM     R1Weights r1w, R2Weights r2w
WHERE    r1w.token = r2w.token
GROUP BY r1w.tid, r2w.tid
HAVING   SUM(r1w.weight*r2w.weight) ≥ φ
```

**Expensive operation!**

---

# Problem 2: Sampling Step

- Cosine similarity = $\Sigma$ weight(token, $t_1$) * weight(token, $t_2$)
- Products cannot be high when weight is small
- Can (safely) drop low weights from $R_2$Weights (adapted from [Cohen & Lewis, SODA97] for efficient execution in an RDBMS)
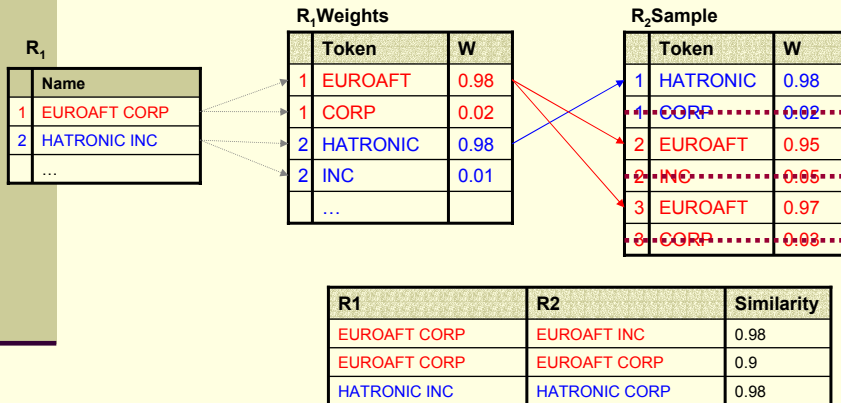
**$R_2$Weights**

|   | Token | W |
|---|---------|--------|
| 1 | EUROAFT | 0.9144 |
| 2 | HATRONIC | 0.8419 |
|   | … |  |
| 2 | CORP | 0.01247 |
| 1 | INC | 0.00504 |

using token weights, sample

→

20 times

**$R_2$Sample**

|   | Token | #TIMES SAMPLED |
|---|---------|----------------|
| 1 | EUROAFT | **18** (18/20=0.90) |
| 2 | HATRONIC | **17** (17/20=0.85) |

**Eliminates low similarity pairs**
*(e.g., "EUROAFT INC" with "HATRONIC INC")*

# Problem 2: Sampling-Based Joins

**R₁**

| | Name |
|---|---|
| 1 | EUROAFT CORP |
| 2 | HATRONIC INC |
| | ... |

**R₁Weights**

| | Token | W |
|---|---|---|
| 1 | EUROAFT | 0.98 |
| 1 | CORP | 0.02 |
| 2 | HATRONIC | 0.98 |
| 2 | INC | 0.01 |
| | ... | |

**R₂Sample**

| | Token | W |
|---|---|---|
| 1 | HATRONIC | 0.98 |
| 1 | CORP | 0.02 |
| 2 | EUROAFT | 0.95 |
| 2 | INC | 0.05 |
| 3 | EUROAFT | 0.97 |
| 3 | CORP | 0.03 |

| R1 | R2 | Similarity |
|---|---|---|
| EUROAFT CORP | EUROAFT INC | 0.98 |
| EUROAFT CORP | EUROAFT CORP | 0.9 |
| HATRONIC INC | HATRONIC CORP | 0.98 |

---

# Problem 2: Sampling-Based SQL

```
SELECT   r1w.tid AS tid1, r2s.tid AS tid2
FROM     R1Weights r1w, R2Sample r2s, R2sum r2sum
WHERE    r1w.token = r2s.token AND r1w.token = r2sum.token
GROUP BY r1w.tid, r2s.tid
HAVING   SUM(r1w.weight*r2sum.total*r2s.c) ≥ S*φ
```

### Fully implemented in pure SQL!

# Problem 2: Experimental Setup

- 40,000 entries from AT&T customer database, split into $R_1$ (26,000 entries) and $R_2$ (14,000 entries)
- Tokenizations:
  - Words
  - Q-grams, q=2 & q=3



- Methods compared:
  - Variations of sample-based joins
  - Baseline in SQL
  - WHIRL [SIGMOD98], adapted for handling q-grams

---
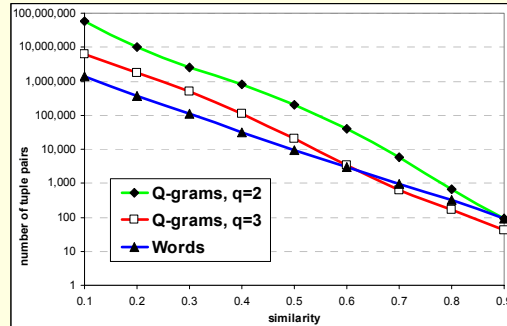
# Problem 2: Metrics

**Execute the (approximate) join for similarity ≥ φ**

- **Precision**: (measures accuracy)
  - Fraction of the pairs in the answer with real **similarity ≥ φ**

- **Recall**: (measures completeness)
  - Fraction of the pairs with real **similarity ≥ φ** that are also in the answer

- **Execution time**
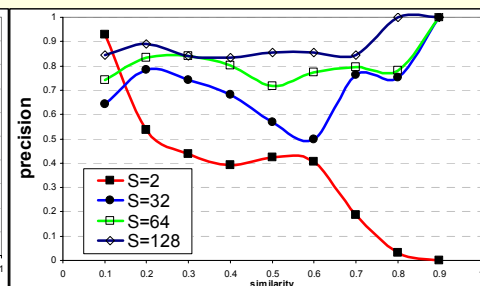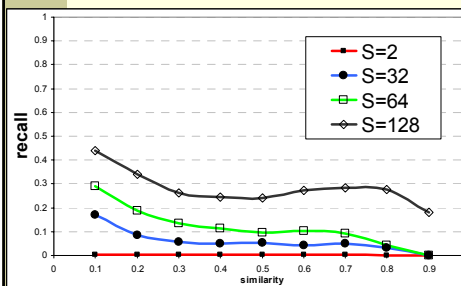
# Problem 2: Comparing with WHIRL
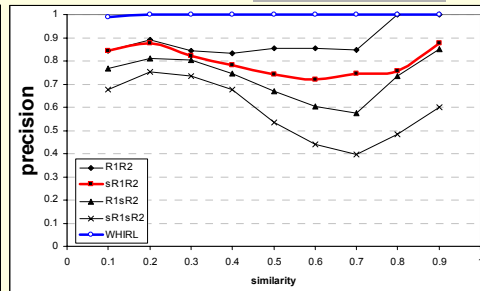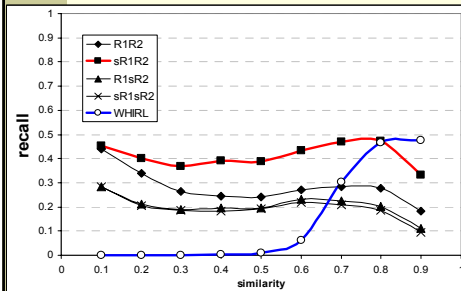


*3-grams, S=128*

- Sample-based Joins: Good recall across similarity thresholds. Retrieves many interesting matches (many good matches have similarity ~0.5)
- WHIRL: Low recall (memory problems) – Misses many good matches
- WHIRL: Perfect precision result of post-join step

---

# Problem 2: Changing Sample Size



*3-grams, S=128*
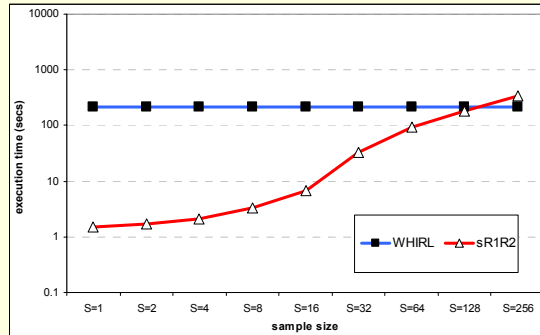
- Increased sample size → More accurate weight estimation → Better recall, precision
- Drawback: Increased execution time (more pairs found)

# Problem 2: Execution Time



*3-grams*

WHIRL and sample-based joins 'break-even' at S ~ 64, 128
Baseline in SQL >24hrs, never finished (out of disk space)

---

# Problem 2: Evaluation Summary

- Sample-based joins preferable when recall is important. WHIRL good for very high thresholds ("a few good matches")

- Cosine similarity with q-grams gives better results (semantically). Higher execution time compared to word-based cosine similarity

- Sample-based joins more adaptable and flexible:
  - …easier to tune
  - …more scalable
  - …more robust
  - …easy to deploy in any environment

# Related Work

- Fellegi & Sunter. A theory for record linkage. *JASA* 1969.
- Winkler. Matching and record linkage. *Business Survey Methods*. Wiley, 1995.

- Hernandez & Stolfo. The merge/purge problem for large databases. SIGMOD 1995.
- Galhardas et al. Declarative data cleaning: Language, model, … VLDB 2001.
- Sarawagi & Bhamidipaty. Interactive deduplication using active learning. KDD 2002.
- Ananthakrishna et al. Eliminating fuzzy duplicates in data warehouses. VLDB 2002.
- Dasu & Johnson: *Exploratory data mining and data cleaning*, Wiley, 2003

- Tejada, Knoblock & Minton. Learning domain-independent string … KDD 2002.

- Navarro. A guided tour to approximate string matching. *ACM Comp. Surveys*, 2001.

---

# Conclusions

- We introduced techniques for mapping approximate string joins into "vanilla" SQL expressions, each with its own advantages:
  - String edit distance: no false positives
  - Cosine similarity: tradeoff between recall and efficiency

- Our techniques do not require modifying the underlying RDBMS
  - Significantly outperform existing approaches

- Other applications?

# Acknowledgements

- Joint work with:
  - Luis Gravano (Columbia University)
  - Panagiotis G. Ipeirotis (Columbia University)
  - H. V. Jagadish (University of Michigan)
  - Nick Koudas (AT&T Labs-Research)
  - S. Muthukrishnan (Rutgers University, AT&T Labs-Research)

- Based on papers in VLDB'01 and WWW'03, and poster in ICDE'03