Similarity Join Algorithms: An Introduction

Wei Wang
University of New South Wales
Australia
http://www.cse.unsw.edu.au/~weiw

Roadmap

A. Motivation
B. Problem Definition and Classification
C. Similarity Join Algorithms
D. Epilogue

Objectives

- Classify existing approaches along based on several perspectives
- Explain several useful ideas in solving the problem

Computers are dumb

```
C:\Users\weiw>python
ActivePython 2.5.1.1 (ActiveState Software Inc.) based on ...
>>> 1.23 / 2.0
0.61499999999999999
```

- Numerical errors

```
double x;
...
if (fabs(x – 0.1) < EPSILON) {
    ...
}
```

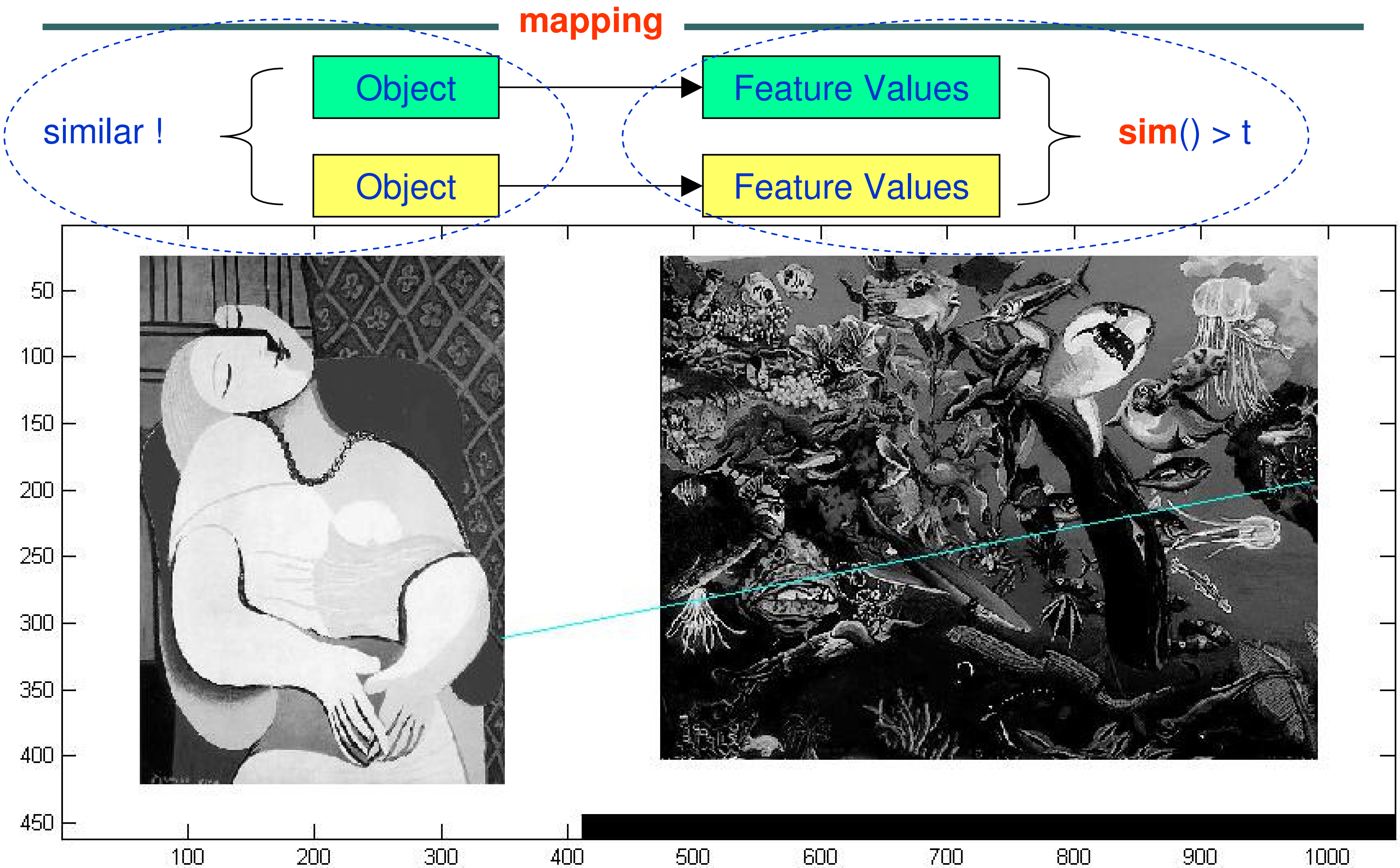
Humans are incomprehensible

- Typo
  - *Why everybdoy can undrstand this?*
- Lack of consistency

炜煒

- Lack of precision
  - *a photo and its digitally-modified version are bit-wise different!*

Notion of Similarities /1



Notion of Similarities /2

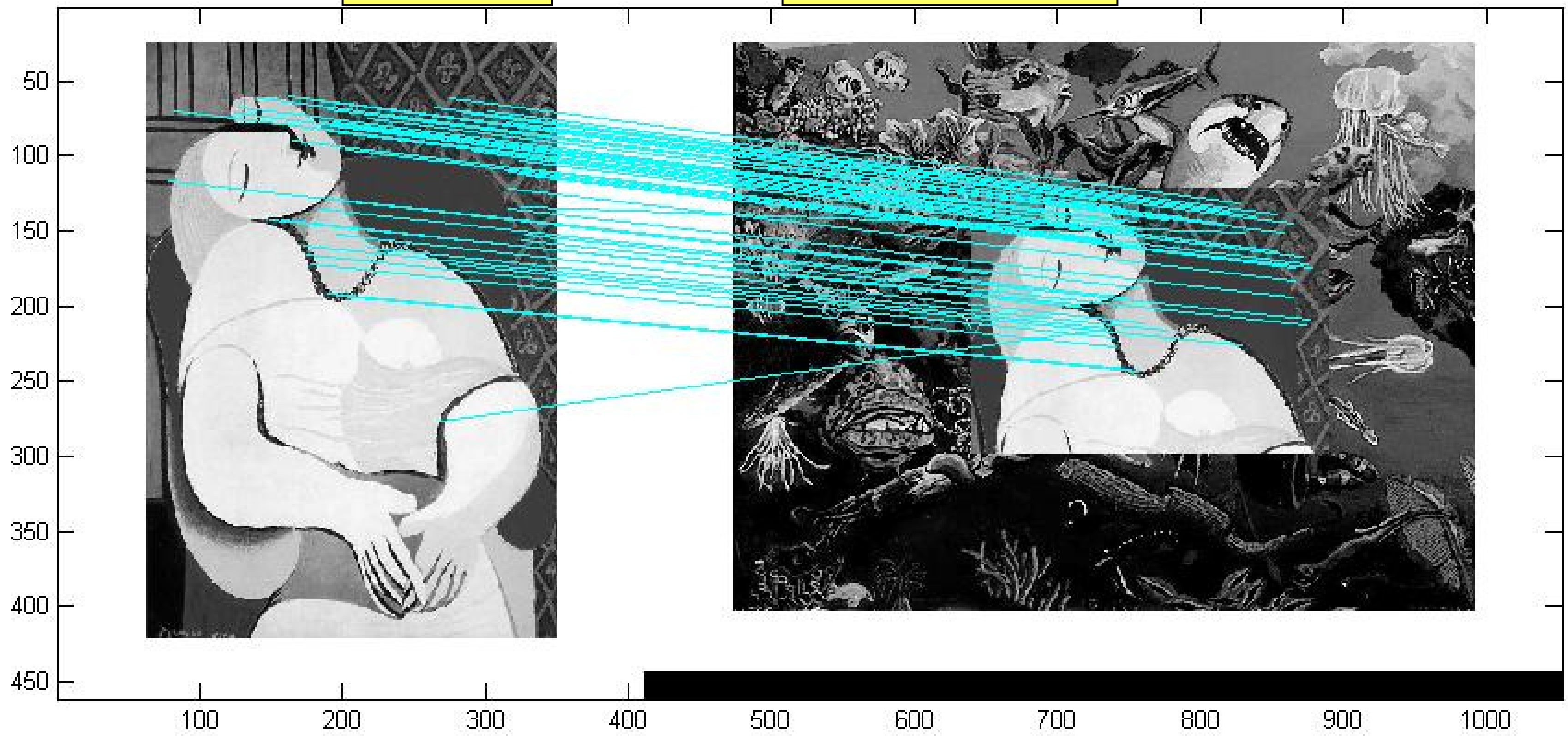
SIFT (+ PCA)

similar !

Object → Feature Values

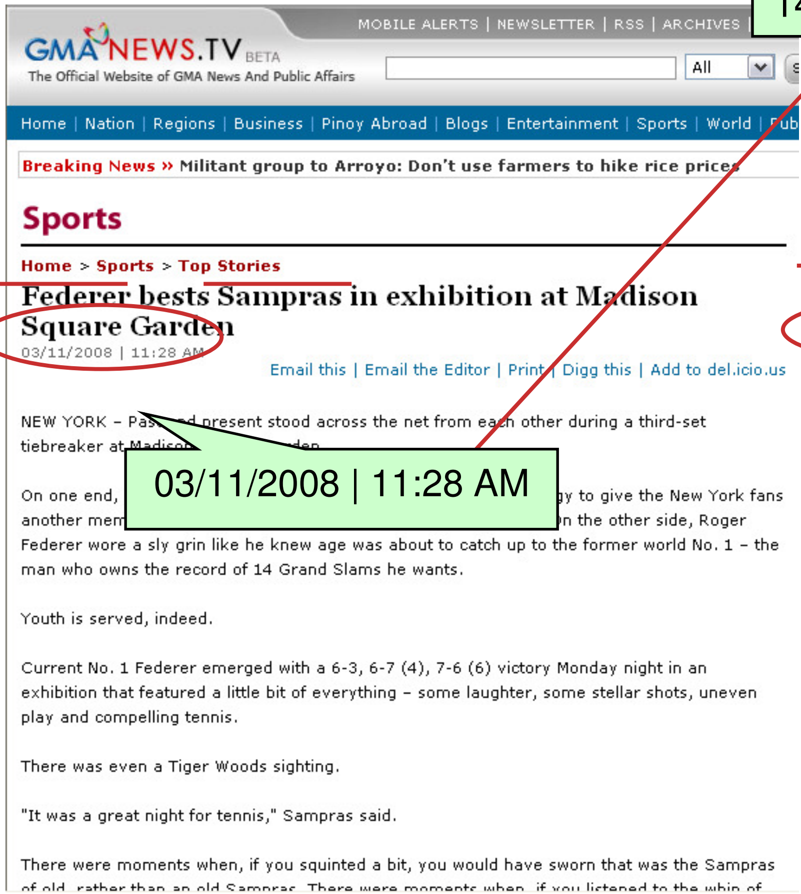
Object → Feature Values

$L_2() <= t$ & RANSAC



App: Deduplicatio

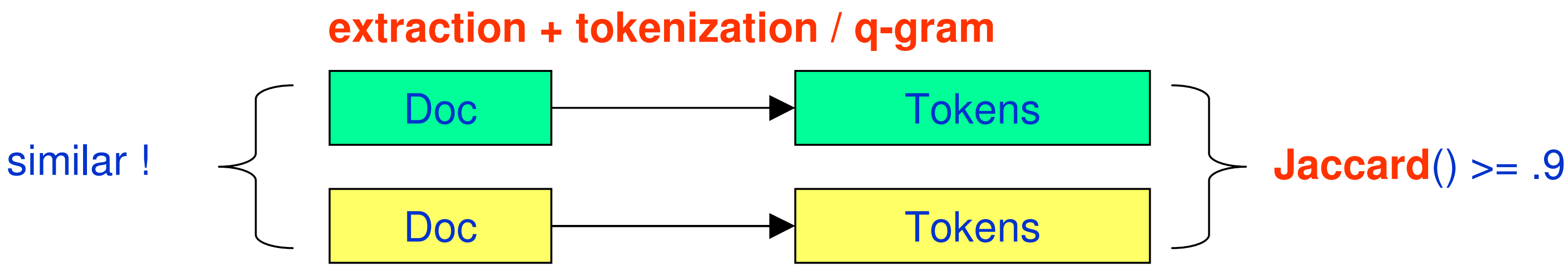
On one end, a winded Pete Sampras tried to summon enough energy to give the New York fans another memorable win to talk about it on the subway ride home. On the other side, Roger Federer wore a sly grin like he knew age was about to catch up to the former world No. 1 - the man who owns the record of 14 Grand Slams he wants.



03/11/2008 | 11:28 AM

By JAY COHEN, AP Sports Writer
Mar 11, 4:23 am EDT

App: Deduplication /2



- Identify spams / plagiarism / copyright protection / replicate Web collections
  - dejavu for MEDLINE database
  - www.rentacoder.com

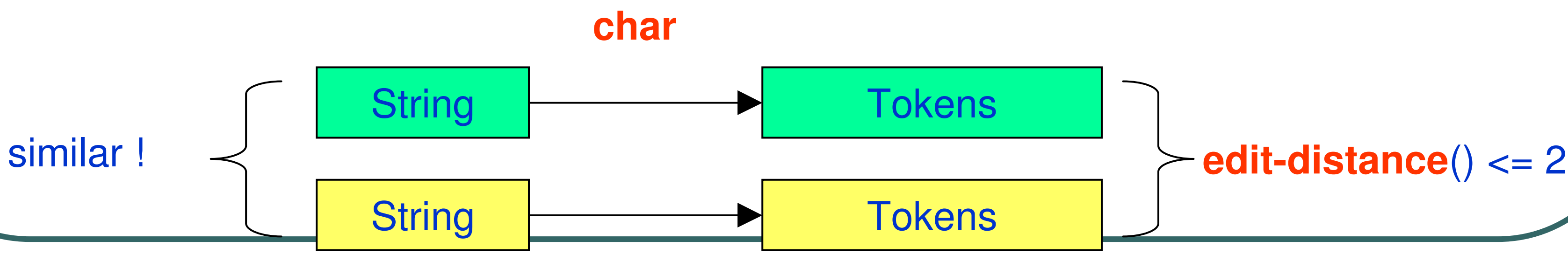
App: Data Integration / Record Linkage

- Merge databases
  - TEXAS

```
burton_ty_r_3412_provine_road_mc_kinney_tx7507008510310808
14070000000155109000020000006402_
burton_ty_r_3412_provine_rd_mc_kinney_tx75070085103108081
40700000001551090010200000006402_
```

  - DBLP

```
evanthia_papadopoulou_l
evanthia_papadopoulou_i
```



Other Applications

- Collaborative filtering
- Bioinformatics
- File/Document management systems
- Match-making services
  - Job recruitment
  - Dating

Roadmap

A. Motivation
B. Problem Definition and Classification
C. Approximate Similarity Join Algorithms
D. Epilogue

Problem Definition

- Input
  - two sets of objects: R and S
  - a similarity function: sim(r, s)
  - a threshold: t
- Output
  - all pairs of objects $r \in R$, $s \in S$, such that

    sim(r, s) $\geq$ t
- Variations
  - dist(r, s) $\leq$ d

E.g., $\cos(D_i, D_j) \geq 0.9$ for near duplicate document/Web page detection.

E.g., edit-dist(s_i, s_j) $\leq$ 2 to match customers' names.

t is usually close to 1
d is usually close to 0

Similarity/Distance Functions

- L_p distance

$$L_p(x,y) = \left(\sum_i |x_i - y_i|^p \right)^{1/p}$$

- Hamming distance

$$H(x,y) = |(x-y) \cup (y-x)|$$

- set_contains?, set_intersects?

$$contains(x,y) = \begin{cases} 1 & ,x \subseteq y \\ 0 & ,x \not\subseteq y \end{cases}$$

- Overlap and Jaccard

$$overlap(x,y) = |x \cap y| \qquad J(x,y) = \frac{|x \cap y|}{|x \cup y|}$$

- Cosine similarity
- Edit distance

$$cosine(x,y) = \frac{\vec{x} \cdot \vec{y}}{\|x\| \cdot \|y\|}$$

Classification

- We look at the *ideas & techniques* used in previous work

	Euclidean	Metric	Others
Exact	Ore / MSJ / GESS	D-index	PSJ, Probe-Count-Opt, SSJoin, All-Pairs, PPJoin+, Ed-Join, Hamming distance join, PartEnum
Approx.	LSH		Shingling, simhash, I-match, SpotSigs, blocking, canopy clustering

Scope

- Connection to many other well-known problems
  - kNN/range search and spatial databases
  - approximate string matching
  - top-k query processing
  - dimensionality reduction (signature-based schemes)
- By no means exhaustive
  - SIGMOD06 tutorial by Koudas, Sarawgi & Srivastava
  - SAC07 tutorial by Zezula, Dohnal & Amato
  - Survey papers
- We focus on “*similarity join*” “*algorithms*”

Roadmap

A. Motivation
B. Problem Definition and Classification
C. Similarity Join Algorithms
   1. Exact algorithm
      - Euclidean
      - Metric
      - Others (set & string)
   2. Approximate algorithm
D. Epilogue

First Glance into the Problem

- Simple variation
  - find exact duplicate → sim(x, y) = 1
  - Use hashing (e.g., SHA1, Rabin's fingerprinting)
- Naïve Algorithm
  - Simple nested loop algorithm
    - Compare all $O(n^2)$ pairs
- Optimization opportunities → ***Be Happy !***
  - ***Be lazy***: only consider promising pairs
  - ***Be aggressive***: pruning-and-refinement paradigm
  - ***Don't be fussy***: Resort to approximate solutions

Challenges

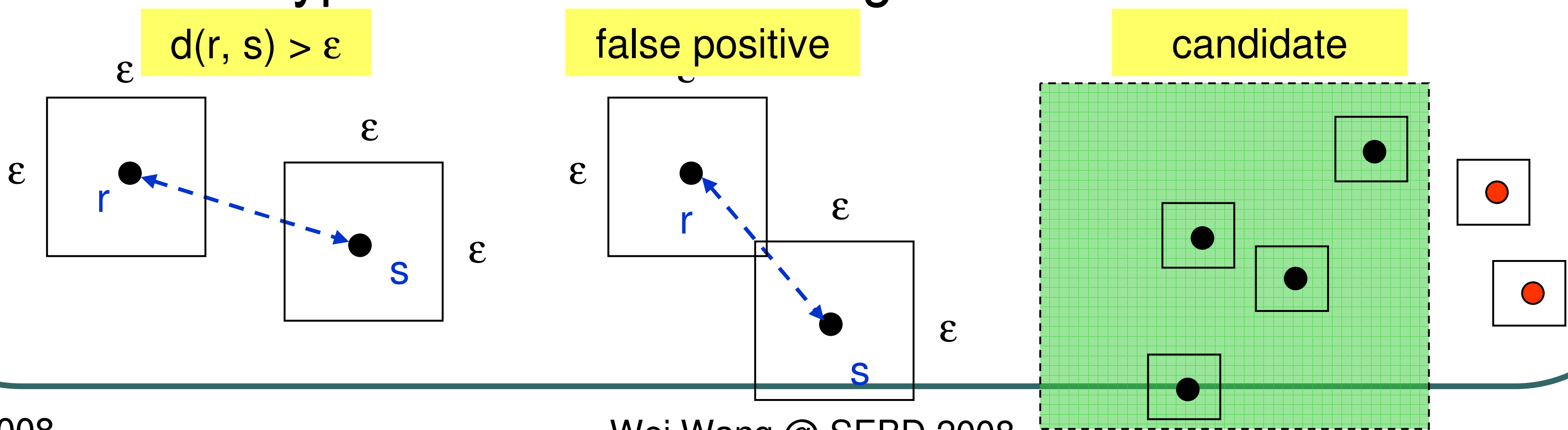
- High dimensionality
  - Curse of dimensionality
  - Sparsity
- Large datasets
- Hard similarity functions
  - expensive to evaluate
  - hard to index
  - do not have nice properties (e.g., transitivity, metric)

C. Similarity Join

1. Exact algorithm
   - Euclidean
   - Metric
   - Others (set & string)
2. Approximate algorithm

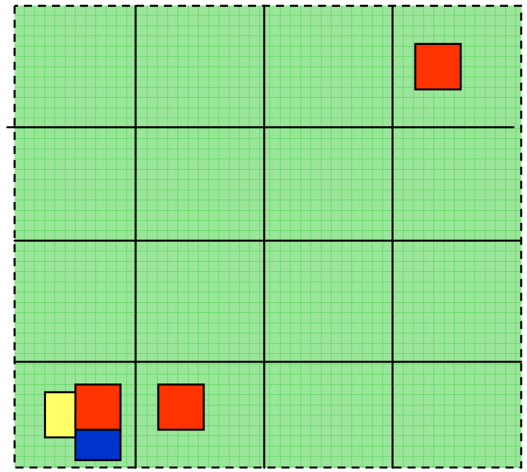
Multidimensional Similarity Join

- Focus on points in a *high dimensional* Euclidean space with L_p distance functions
  - $\{\langle r, s\rangle \mid r \in R, s \in S, L_p(r, s) \le \varepsilon\}$
- We pick Ore/MSJ/GESS as a representative method
  [Orenstein, SSD91] [Koudas & Sevcik, TKDE00] [Dittrich & Seeger, KDD01]
- Utilize hypercube-based filtering



Replication

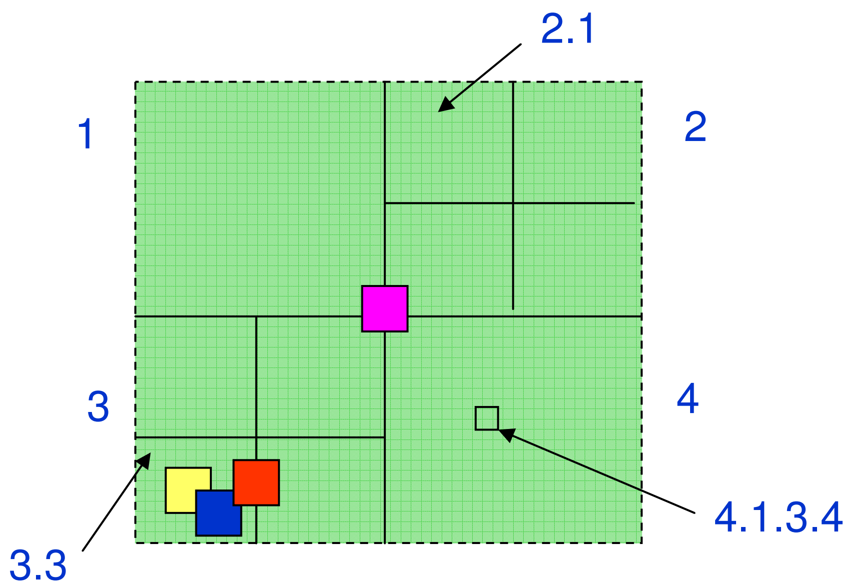
- Only consider the finest partitions
- Use replication if a hypercube intersects multiple partitions
- To find overlapping hypercubes, only consider partitions <x, y>, s.t.,
  - x = y
- Problems:
  - Too much replication
  - Need deduplication
- Other methods
  - ε-kdb-tree [Shim et al, ICDE 97] avoids replication but accesses neighboring partitions on-the-fly



Recursive Space Partitioning

- "Hash" hypercubes into their smallest enclosing partitions (or "buckets")
- To find overlapping hypercubes, only consider partitions <x, y>, s.t.,
  - x = y
  - or x is a prefix of y

Partition	Cubes
ϕ	
3	
3.3	



Merge Join with a Stack

- Sort partitions based on their labels
- Perform merge join with a stack
- Generate candidate pairs when popping elements from the stack

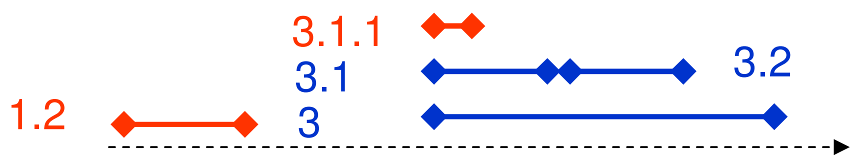
R	
3	{a}
3.1	{b}
3.2	{c}

$\bowtie$

S	
1.2	{d}
3.1.1	{e}
3.2	{f}

Correct as if r overlaps s, r and s has a containment relationship

stack:



Other Approaches

- LSS algorithm that is GPU's parallel sort-and-search capability [Lieberman et al, ICDE08]

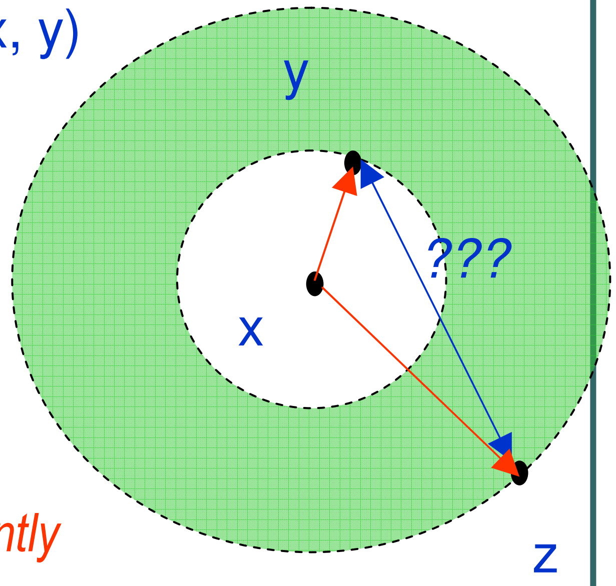
C. Similarity Join

1. Exact algorithm
   - Euclidean
   - Metric
   - Others (set & string)
2. Approximate algorithm

Similarity Join in Metric Space

- Three approaches experimentally studied in [Dohnal et al, ECIR 03]
  - Partition-based
    - Partition on $d(p, x_i)$
  - Filtering-based
    - Multiple pivots + triangle inequality filtering
  - Index-based
    - D-Index(ρ)
- Other approaches
  - [Paredes and Reyes, SISAP 08] indexes both joining sets *jointly*
  - [Jin et al, DASFAA 03] uses StringMap to derive approximate answers

$$d(y, z) \geq d(x, z) - d(x, y)$$



C. Similarity Join

1. Exact algorithm
   - Euclidean
   - Metric
   - Others (set & string)
2. Approximate algorithm

Similarity Join for Sets & Strings

- Similarity between *sets*
  - Binary similarity functions
    - Contains, intersects
  - Numerical similarity functions
    - Overlap, Jaccard, dice, cosine
- Similarity between *strings*
  - Treat strings as sets
  - Jaccard (on q-grams), edit distance

Set Containment Join /1

- Problem:
  - find { (r, s) | r ∈ R, s ∈ S, r ⊆ s }
- PSJ Algorithm [Ramasamy et al, VLDB00]
  - Generate candidates
    - len+sig(r) → hash(*random-elem*(r))
    - len+sig(s) → hash(s[1]), hash(s[2]), …
    - Join only corresponding partitions
      - with (length, signature) optimizations
  - Verification
    - Test if r ⊆ s

Set Containment Join /2

- Other methods
  - signature hash join [Helmer & Moerkotte, VLDB97]
  - index-nested-loop-join is faster, even building an in-memory index on-the-fly [Mamoulis, SIGMOD03]

Set Similarity Join

- Problem
  - find $\{(r, s) \mid r \in R, s \in S, \text{overlap}(r, s) \geq t\}$
  - A fundamental "operator"
    - can handle other similarity functions (Jaccard, cosine, Hamming, dice, edit distance, …) via transformation
- Probe-Count-Opt Algorithm [Sarawagi & Kirpal, SIGMOD04]
  - index-nested-loop-join style
  - for each tuple, invoke an optimized version of *list merge with threshold* algorithm

Probe-Count /1

- Upper bounding the overlap

-- overlap constraint: t = 3

-- current record = {a, b, c, d, e}

1	1	2	2	3	3	…	9	9	9	9	…

a	1	3	5	7	9	11	13	15
b	2	4	6	8	9	12	14	
c	1	5	7	19				
d	3	4	9	14				
e	2	8	9	11				
f	1	9	15	27				

$\}$ t-1 (a, b)

Cand-Gen

Candidates = I(a) ∪ I(b) ∪ I(c) ∪ I(d) ∪ I(e)

⇩

Candidates = I(c) ∪ I(d) ∪ I(e)

= { 1, 2, 3, 4, 5, 7, 8, 9*2, 11, … }

Verification

Verify 1: binary_search(I(b), 1) = false

➔ overlap(cur, 1) ≤ 2

Verify 2: …

…

Probe Count /2

- Other optimizations
  - Sorting by increasing record size
  - Clustering
  - External memory version
- Other methods
  - ScanCount, MergeSkip, Divide-Skip [Li et al, ICDE08]
- Comment on Probe-Count-Opt

  e.g., t = .9 * |S|

  ✓ Only the *rarest* |S| - (t-1) tokens are used to generate candidates

  × *Verification may be quite expensive*

  × *Unnecessary candidates generated (and verified)*

Prefix Filtering-based Similarity Joins

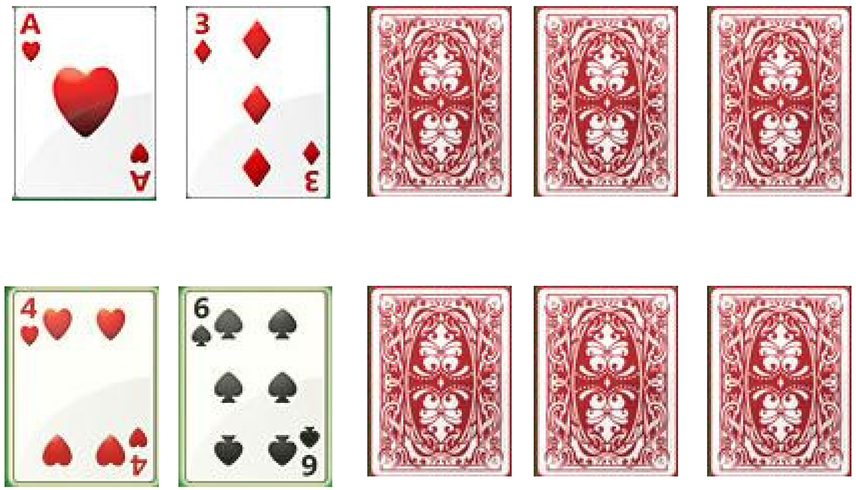
- SSJoin [Chaudhuri et al, ICDE06]
  - Formalize the *prefix-filtering* principle and use it in a *symmetric* way
  - Access original record for verification
- All-Pairs [Bayardo et al, WWW07]
  - Use prefix-filtering in an *asymmetric* way
- PPJoin+ [Xiao et al, WWW08]
  - Employs prefix-filtering, position filtering and suffix filtering

Prefix Filtering /1

- Establish an upper bound of the overlap between two sets based on *part of* them

sorted

Player 1



Player 2

sorted

if t=4,
overlap(player1, player2) < t
or
upperbound(overlap(p1,p2))=t-1

What's the maximum possible number of cards held by both players (denomination not considered) ?

Prefix Filtering /2

prefix-len = |U| - (t-1) for overlap similarity function

- Formally
  - $\text{Prefix}_t(U) \cap \text{Prefix}_t{=}(V) = \phi \rightarrow \text{overlap}(U, V) < t$
  - i.e., (U, V) can be safely pruned
  - Global ordering important
- Algorithm (on top of an RDBMS)
  - Compute prefix(S) for each record S
  - Candidates = { pairs of records that share at least one token in their prefixes }
  - Verify(Candidates)

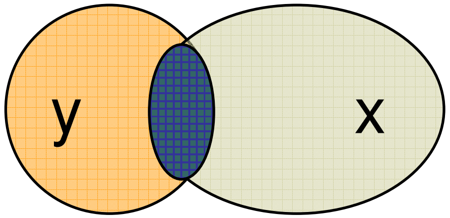
All-Pairs [Bayardo et al, WWW07]

- All-Pairs improves SSJoin
  - stand-alone implementation
  - tight transformation between similarity/distance functions
  - hash table instead heap
  - *indexing & probing prefixes*

  - also tackles *weighted* cosine similarity join

Relationships among Similarity/Distance Functions

- **Jaccard similarity**
  - $J(x, y) = | x \cap y | / | x \cup y|$
  - $J(x, y) \geq t \iff O(x,y) \geq \frac{t}{1+t} \cdot (|x| + |y|)$
  - $J(x, y) \geq t \implies |y| \geq t \cdot |x|$ (wlog. if $|y| \leq |x|$)



- Cosine similarity
  - similar transformations can be obtained.
  - $\cos(x, y) \geq t \iff O(x,y) \geq t\sqrt{|x| \cdot |y|}$
  - $J(x, y) \geq t \implies |y| \geq t^2 \cdot |x|$ (wlog. if $|y| \leq |x|$)
- Edit distance

Verify(x, {y1, y2, …})

```
for each y_i
  if overlap(x, y_i) ≥ t
      output(<x, y_i>)
end
```

All-Pairs

- for each $S_j \in S$ in increasing size *// nested loop*
  - Candidates = ϕ
  - prefix-len = calc_*probing*_prefix_len()
  - for i=1 to prefix-len *// go thru probing prefix*
    - w = $S_j[i]$
    - for each $S_k \in$ Inverted-list(w) & len-filter *// prefix(S_k) and prefix(S_j) // intersects*
      - Candidates = Candidates $\cup S_k$
    - If i < calc_*indexing*_prefix_len()
      - Inverted-list(w) = Inverted-list(w) $\cup S_j$ *// index the current token*
  - Verify(S_j, Candidates)

Prefix Lengths

- Jaccard similarity [Xiao et al, WWW08]
  - J(x, y) ≥ t $\Leftrightarrow$ O(x, y) ≥ (t/(1+t)) * (|x| + |y|)
  - *indexing*-prefix-len = $|x| - \left\lceil \frac{2t}{1+t} |x| \right\rceil + 1$
  - *probing*-prefix-len = $|x| - \lceil t |x| \rceil + 1$
- Cosine similarity [Bayardo et al, WWW07]
  - cos(x, y) ≥ t $\Leftrightarrow$ O(x, y) ≥ t * $(|x|*|y|)^{1/2}$
  - *indexing*-prefix-len = $|x| - \lceil t |x| \rceil + 1$
  - *probing*-prefix-len = $|x| - \left| t^2 |x| \right| + 1$

RID	Name
1	Database System Concepts
2	Database Concepts Techniques
3	Database System Programming Concepts Oracle Linux
4	Database Programming Concepts Illustrated
5	System Programming Concepts Techniques Oracle Linux

len
3
3
6
4
6

Order: Illustrated, Linux, Oracle, Techniques, Programming, System, Database, Concepts

token	df	Order
Database	4	7
System	3	6
Concepts	5	8
Techniques	2	4
Programming	3	5
Oracle	2	3
Linux	2	2
Illustrated	1	1

RID	Name	len	pl	il
1	System Database Concepts	3	1	1
2	Techniques Database Concepts	3	1	1
4	Illustrated Programming Database Concepts	4	1	1
3	Linux Oracle Programming System Database Concepts	6	2	1
5	Linux Oracle Techniques Programming System Concepts	6	2	1

length filtering does not help in this toy example

Jaccard, t=0.8

cur RID = 1 2 3 4 5

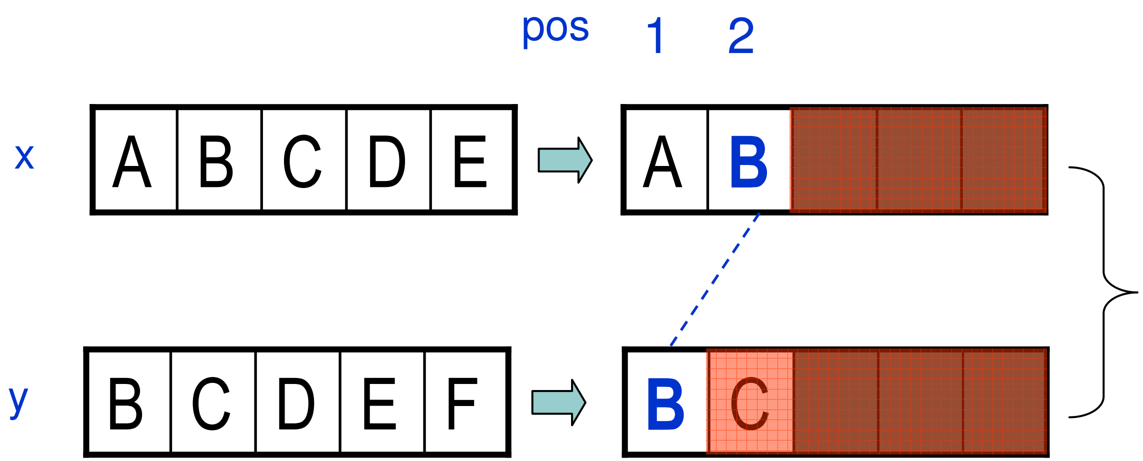
token	Inverted list
System	{1}
Techniques	{2}
Illustrated	{3}
Linux	{4, 5}

PPJoin+ [Xiao et al, WWW08]

- PPJoin+ improves All-Pairs
  - Optimized for Jaccard/cosine similarity constraints
  - less candidates generated
  - less full-scale verifications
- Idea: *fully* exploit the global ordering
  - Record the position of the tokens in the prefix ➔ ppjoin
  - Probe the tokens in the suffixes ➔ ppjoin+

How Positional Information Helps/1

- Derive an upper bound of the overlap based on position information in the prefixes



All-Pairs

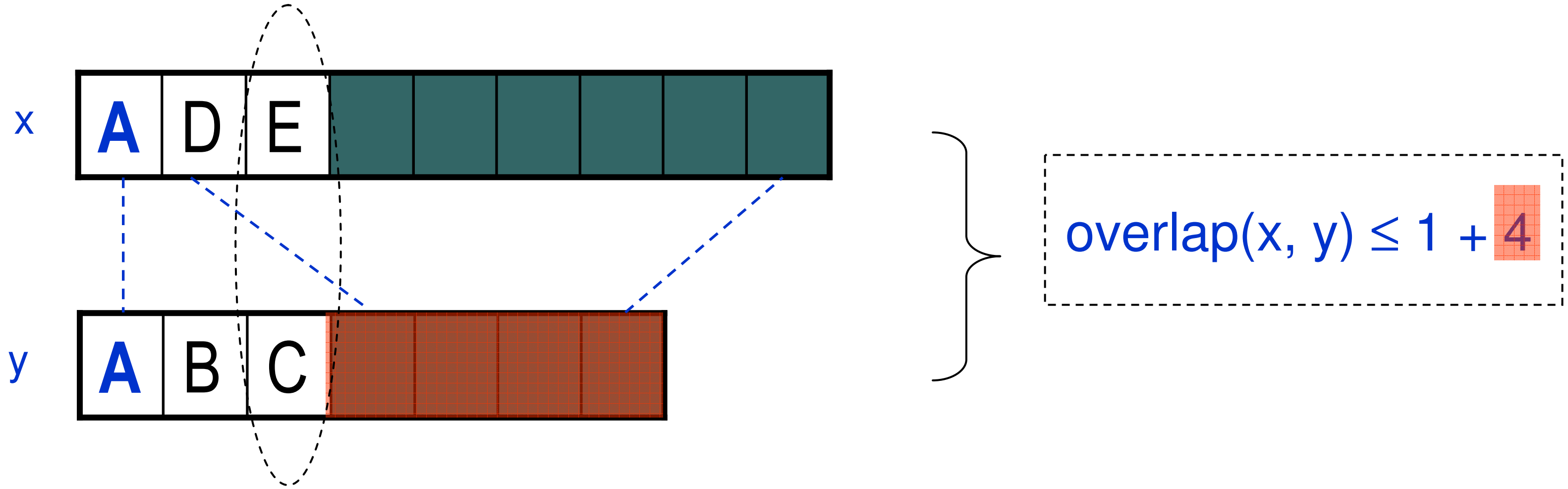
prefix(x) ∩ prefix(y) != φ
➔ <x, y> is a candidate pair

PPJoin

overlap(x, y) ≤ 1 + min{ (5-2), (5-1) } = 4 ≤ α = 5
➔ <x, y> is NOT a candidate pair

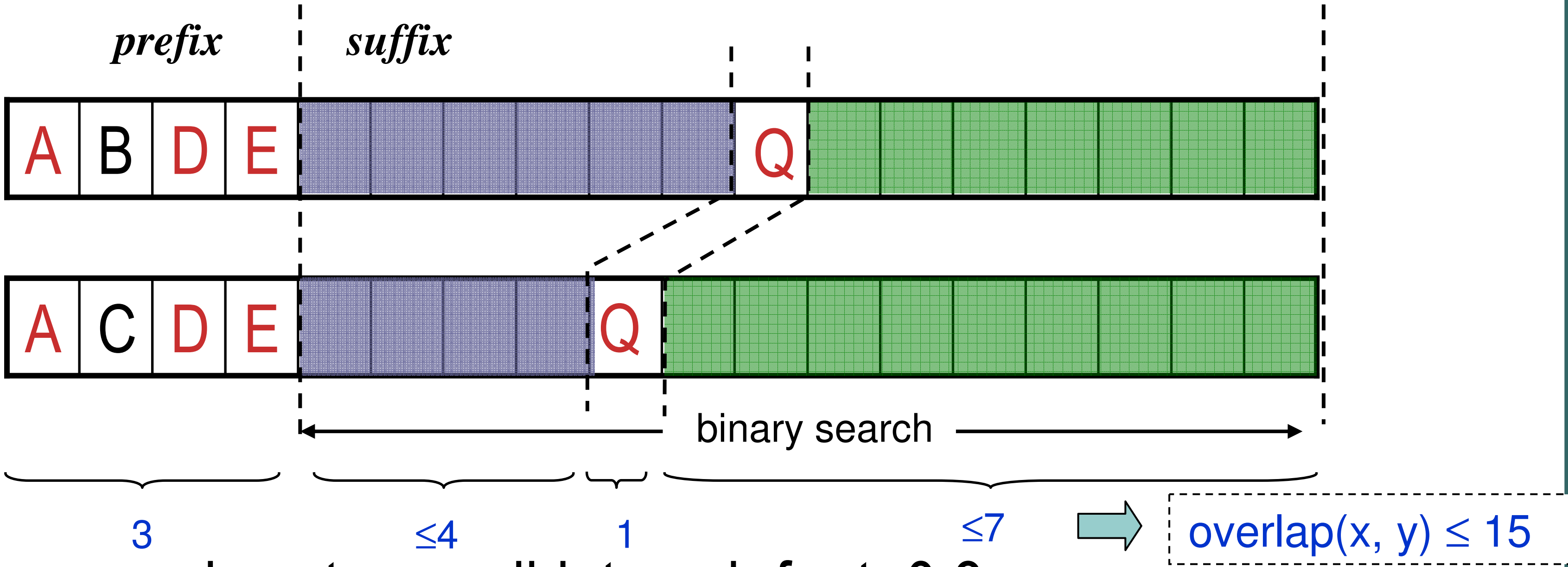
How Positional Information Helps/2

- Also useful in verification



ppjoin+ /1

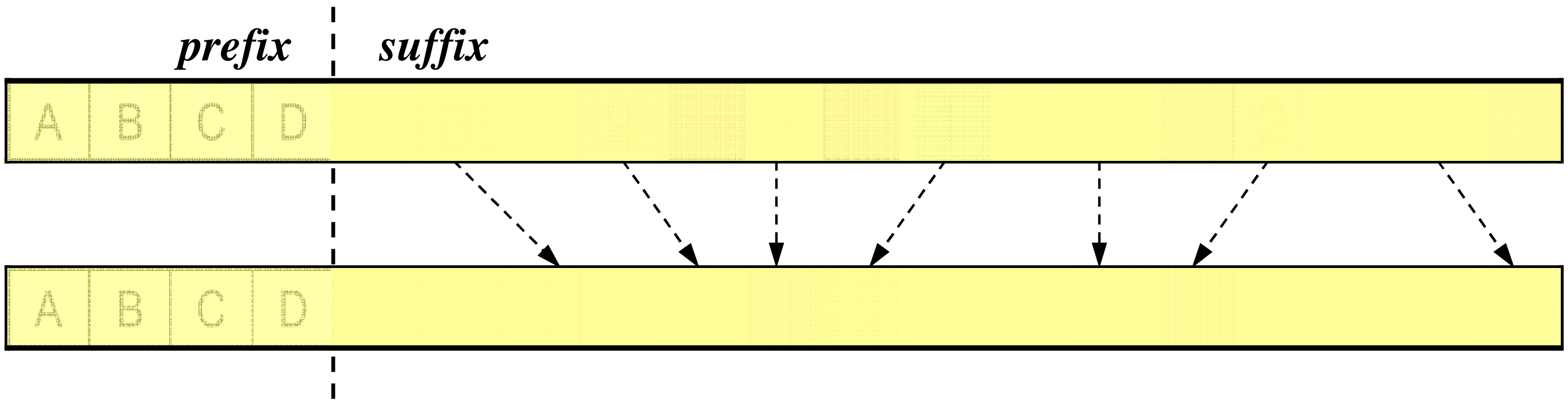
- *Can position information be used to the suffixes?*



- <x,y> is not a candidate pair for t=0.8
  - Overlap(x, y) must be ≥ 16

ppjoin+ /2

- Apply multiple probes in a divide-and-conquer manner
  - stop conditions: either reach MAX_DEPTH or current candidate pair is pruned



$\text{ubound}_{dep=1} = 4 + 6 + 1 + 7 = 18$

$\text{ubound}_{dep=2} = 4 + 3 + 1 + 1 + 1 + 3 + 1 + 3 = 17$

$\text{ubound}_{dep=3} = 4 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 = 15$

Edit Similarity Join

- Edit distance
  - Widely used text dissimilarity measure
    - Models human errors (e.g., typos)
  - Expensive to evaluate
    - O(len^2) using standard dynamic programming
- Similarity join with an edit distance threshold
  - i.e., find (r, s) s.t. ed(r, s) ≤ d

q-gram-based Method

- q-gram-based filtering
[Gravano et al, VLDB01]

count filtering

  - if ed(r, s) ≤ d → at least LB(r,s) common q-grams between them
  - LB(r, s) = max(|r|, |s|) + q + 1 – d*q

length filtering

  - | |r| - |s| | ≤ d

position filtering

  - positions of the matching q-grams should be within d

- Implementation via SQL & UDF
  - q=2 achieves best performance

```
      itali
{ ##i
   #it
    ita
     tal
      ali  }
       li$   d*q q-grams
        i$$ } destroyed
```

Implication: *Edit similarity join can be processed using other similarity join algorithm*

Ed-Join [Xiao et al, VLDB08]

- Ed-Join improves the previous method
  - Location-based *mismatch* filtering
    - Prefix filtering with minimum prefix length (for edit distance)
  - Content-based *mismatch* filtering
  - Interesting experimental results
- Idea
  - *mismatching q-grams* also provide useful information

Location-based Mismatch Filtering

- Prefix length = q*d + 1 ➔ *Minimum* prefix length l ∈ [d+1, q*d+1]
  - (r, s) is a candidate pair **only if** their *minimum prefixes* intersects

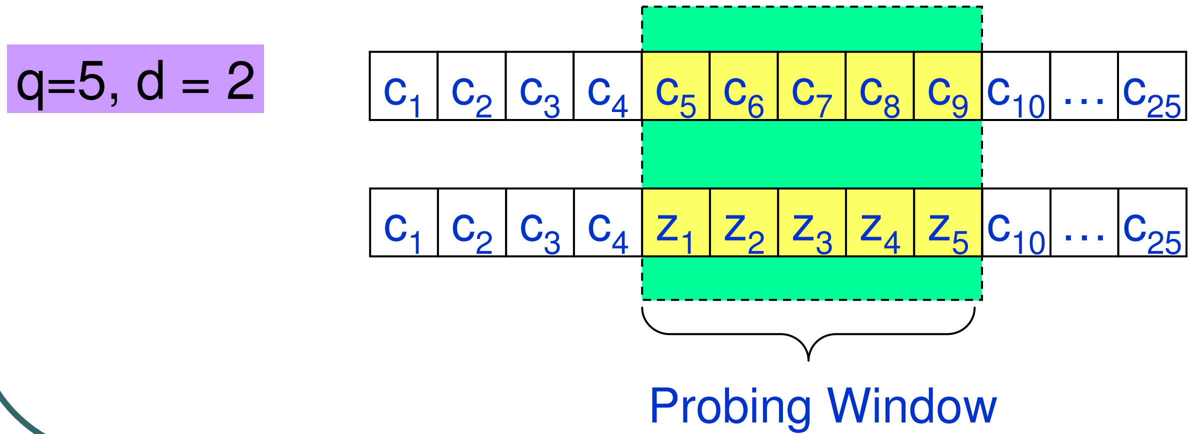
q=2, d = 1

abaabab	(aa,3)	(ab,1)	(ab,4)	(ab,6)	(ba,2)	(ba,5)	min-prefix
xxyabab	(xx,1)	(xy,2)	(ab,4)				min-prefix

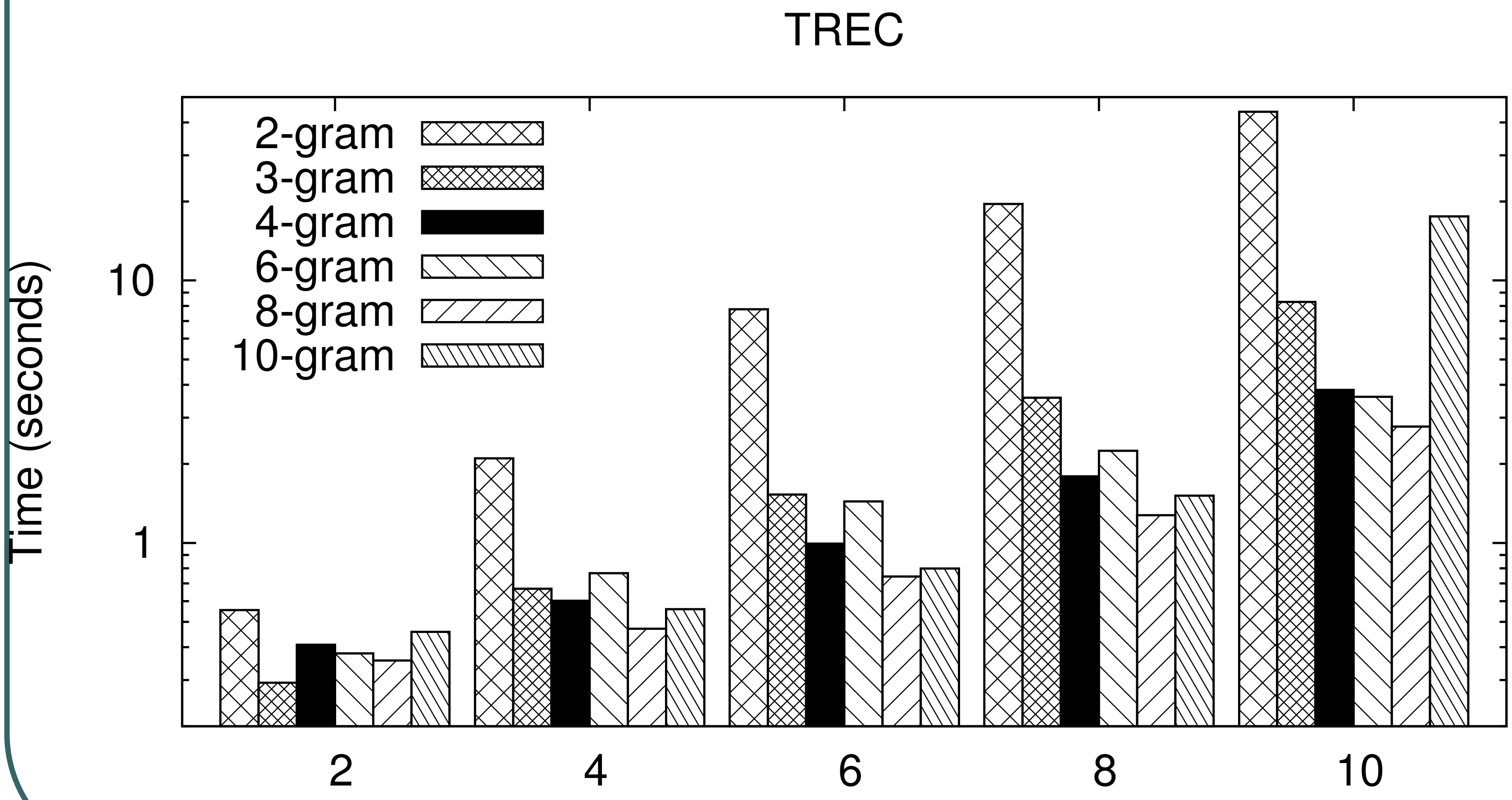
  - ➔ less candidates
- *Count filtering* is a special case of *location-based mismatch filtering*

Content-based Mismatch Filtering

- Effective for *burst errors* ← worst case for count filtering
  - "We use Sybase" → "We use Oracle"
- L_1 distance within any *probing window* ≤ 2*d



Optimal q-gram Length



q-grams longer than 2 is usually (much) better !

Other Approaches

- Variants of q-gram
  - VGRAM [Li et al, VLDB07], proposes variable-length q-grams
  - Gapped q-gram [Burkhardt & Kärkkäinen, CPM02], only applicable to d=1
- Neighborhood generation
  - FastSS [Bocek et al, ETH TR 07] use deletions only and achieve $O(d * \Sigma^k * \log(n\Sigma^k))$ *similarity query* time and $O(n\Sigma^k)$ space.
- + *divide and conquer* based on *pigeon hole principle*
  - Hamming Distance Join [Manku et al, WWW07]
  - PartEnum [Arasu et al, VLDB06]

Partitioning-based Approaches

- *Enumeration + Divide and conquer*
  - Hamming Distance Join [Manku et al, WWW07]
  - PartEnum [Arasu et al, VLDB06]
  - both works for *Hamming distance* threshold, but other constraints can be easily transformed to Hamming distance constraint, e.g.,

$$J(x, y) \geq t \Longleftrightarrow H(x, y) \leq \frac{1-t}{1+t} \cdot (|x| + |y|)$$

Hamming Distance Join [Manku et al, WWW07]

- Background
  - N docs mapped to sketches of f-***bit***s each (using simhash [Charikar, STOC02])
  - given a new document, generate its sketch q
  - need to return all sketches that has Hamming distance at most t from q, i.e., $Hamming(x, y) \leq t$
- No "good" theoretical solutions
- Naïve solutions
  - Query expansion OR Data replication

too many queries *too many copies*

this proposal

Hamming Distance Query [Manku et al, WWW07]

- if v is an answer, v and q differ by at most t bits
  - but these t bits can be anywhere within the [1 .. f]
  - solution: partition

f=6, t=2

q	1	1	0	1	0	0
v_1	1	1	0	1	1	1

- *Duplicate data 3 times (or index)*
- *$E(|C_i|) = N / 2^2$*
- *Design of parameters important*

$\binom{15}{2}$

Problem:
- #Duplication = C(n, t)
- |Cand_i| = N / $2^{(n-t)*c}$
- Cannot deal with large t

form n=3 partitions

q	1	1	0	1	0	0

How many partitions are preserved?

$\binom{3}{1}$

1	1	?	?	?	?	⇨ C_1 = {a, … }
?	?	0	1	?	?	⇨ C_2 = {x, … }
?	?	?	?	0	0	⇨ C_3 = {m, … }

Elements in C_i need further verification

PartEnum [Arasu et al, VLDB06]

- ***Part* + *Part* + *Enum***

f=6, t=3

form 2 partitions

v | 1 | 1 | 0 || 1 | 0 | 0 |

form 3 2^{nd}-level partitions

| 1 | 1 | 0 | ⇨ { 1?? , ?10, ??0 }

| 1 | 0 | 0 | ⇨ { 1?? , ?0?, ??0 }

At least one partition has error $\lfloor t/2 \rfloor = 1$ ← Pigeon hole principle

Part (n1=2 partitions)

Enum (n2=3 partitions)

- *Each record generate* $n1 \binom{n2}{\lfloor k/n1 \rfloor}$ *signatures*
- *Hamming(u, v)* $\leq k$ ➔ *sigs(u)* $\cap$ *sigs(v)* <> ϕ

t = 10
- ENUM with n=12 ➔ 66 sigs / record
- PartEnum with n1=3, n2=4 ➔ 12 sigs / record

Approximate Similariy Join Algorithms

- Sketch-based methods (for metric space)
  - LSH
  - Shingling
  - Randomized Projection

> - Theoretical Guarantee on the Approximation
> - Still hard to perform the join on the sketches

- Heuristic methods
  - Blocking, Canopy
  - I-Match [Chowdhury et al, TOIS 02, Kolcz et al, SIGKDD 04]
  - SpotSigs

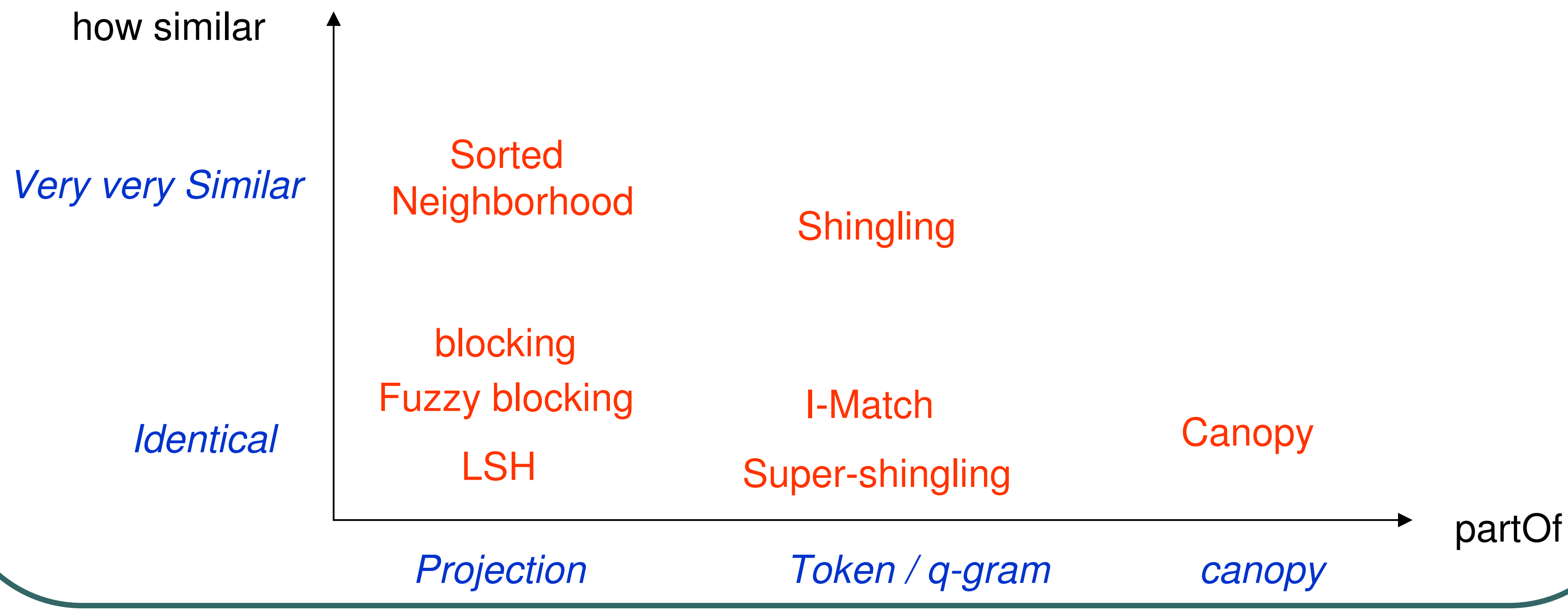
> - Works well for specific types of applications/datasets

C. Similarity Join

1. Exact algorithm
   - Euclidean
   - Metric
   - Others (set & string)
2. Approximate algorithm

The Idea

- X and Y are very similar ➔ partOf(X) is "very very similar" to partOf(Y)



Locality Sensitive Hashing

- LSH solves nearest neighbor problem approximately [Indyk & Motwani, STOC98] [Gionis et al, VLDB99] [Indyk FOCS00] … [Andoni & Indyk, FOCS06]
  - Widely used, e.g., multimedia database & computer vision
- Idea:
  - encourage collision of h(x) and h(y) when $x \approx y$
  - contrast this with traditional & cryptology hash functions

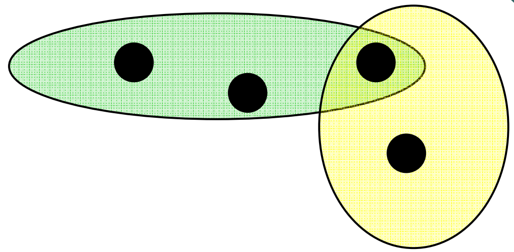
Definition

- LSH: a family H is called (R, cR, p_1, p_2)-sensitive if for any two points $x, y \in \Re^d$
  - if $d(x, y) \leq R \rightarrow Pr_H[h(x) = h(y)] \geq p_1$
  - if $d(x, y) \geq cR \rightarrow Pr_H[h(x) = h(y)] \leq p_2$

$c > 1$
$p_1 > p_2$

LSH Cookbook

- Known LSH families
  - $\Re^d$, Hamming distance
    - $h_k(x) = x_k$, i.e., random projection on one dimension
  - $\Re^d$, L_1 distance
  - $\Re^d$, L_p distance
    - $h_{\mathbf{r},b} = \lfloor (\mathbf{r} \cdot \mathbf{x} + b) / w \rfloor$, $\mathbf{r}[i]$ is sampled from Gaussian distribution
    - p-stable distribution for $p \in [0, 2]$
  - Jaccard: *min-hashing*
  - arccos: *simhash*
  - L_2 distance on a unit hypersphere [Terasawa & Tanaka, WADS07]

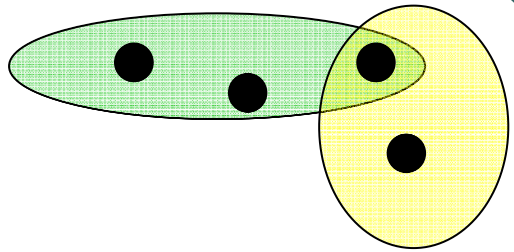


Shingling

- Doc D ➔ *set* of Shingles (aka. q-grams)
  - $Sim(D_i, D_j) = Jaccard(Shingles(D_i), Shingles(D_j))$
- Consider the universe $U = | R \cup S |$
  - Random (wrt U) sample ***one*** element from R and S
  - $P[sample(R) = sample(S)] = | R \cap S | / | R \cup S | = Jaccard$
- However, we don't know U beforehand
  - Min-hashing
    - randomly (wrt I) permutate $e_i \in R$
    - select the first element after permutation

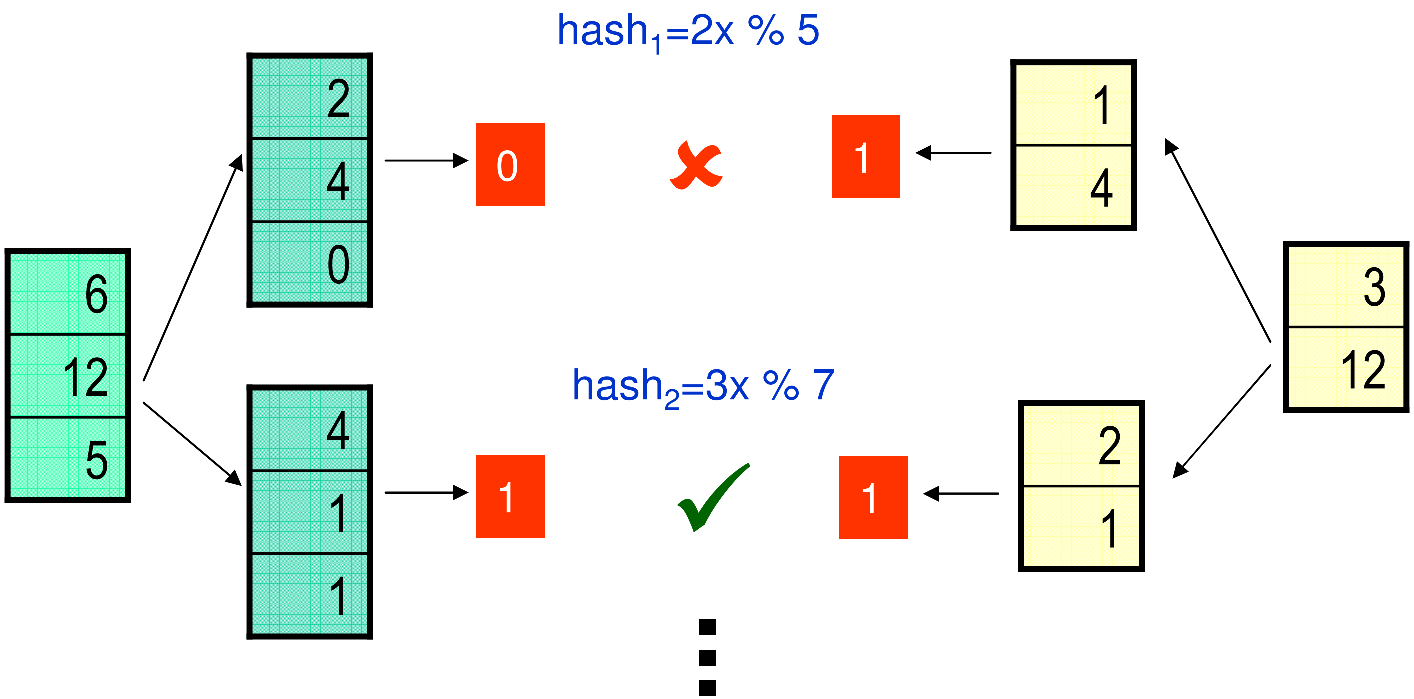
e_i ➔ $hash(e_i)$

$sig(R) = min_i \{ hash(e_i) \}$



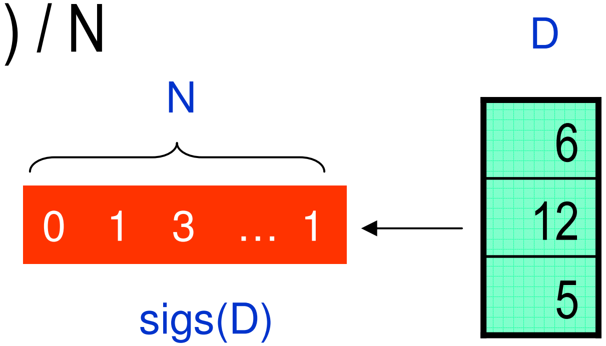
Shingling Example

- Jaccard(R, S) $\approx$ COUNT(h(R) = h(S)) / N



Joining the Signatures

- ~~Doc D → set of Shingles (aka. q-grams)~~
  - ~~Sim(D_i, D_j) = Jaccard(Shingle*s*(D_i), Shingle*s*(D_j))~~
- Doc D → set of signatures (of shingles)
  - Sim(D_i, D_j) = Overlap(sig*s*(D_i), sig*s*(D_j)) / N



- Still expensive for exact join
  - Remove frequent shingles [Heintze 1996]
  - Retain only every 25th shingle [Broder et al, WWW97]
  - with *both* optimizations, 10 days for 30M docs
  - Super-shingling, with overlap threshold = 1

SimHash

- Generalization of LSH to other similarity measures [Charikar, STOC 02]
  - θ(**x**, **y**): related to cosine
  - $h_{\mathbf{u}}(\mathbf{x}) = sign(\mathbf{u} \cdot \mathbf{x})$, where **u** is a random unit vector
  - then $\Pr[h_{\mathbf{u}}(\mathbf{x}) = h_{\mathbf{u}}(\mathbf{y})] = 1 - \theta(\mathbf{x}, \mathbf{y}) / \pi$

Practical Implementation

- Near duplicate Web page detection from google [Henzinger, SIGIR06] [Manku et al, WWW07]
  - Document D ➔ set of tokens with idf weighting ➔ form a set of "features" v(D)
  - Each feature is randomly projected to *f*-dimensional binary vector of [-1,1]
  - Sum up the weighted projections of all features in v(D) ➔ **r**(D)
  - a *f*-bit signature sig(D) ← sign(**r**(D))
- Results (in comparison with Shingling)
  - Fairly accurate and stable
  - Does not capture order among tokens

Approximate Join Algorithm Without Quality Guarantee

- Application areas:
  - Record linkage, data cleaning
  - Clustering
- Algorithms:
  - Standard blocking
  - Sorted neighborhood
  - Fuzzy blocking
  - Canopy clustering

Blocking

- Standard blocking [Jaro, JASS89]
- Idea: similar records usually have *identical* feature values
- Algorithm:
  - GROUP BY the blocking key (e.g., lastname[1..4])
  - pair-wise comparison within each group
- Limitations
  - Strong assumption (e.g., no typo in lastname[1..4])
  - Recall depends on the choice of the blocking key

Sorted Neighborhood
[Hernandez & Stolfo, SIGMOD95]

ed(x.fname,y.fname)<3 &&
geo-dist(x.addr, y.addr) → x=y

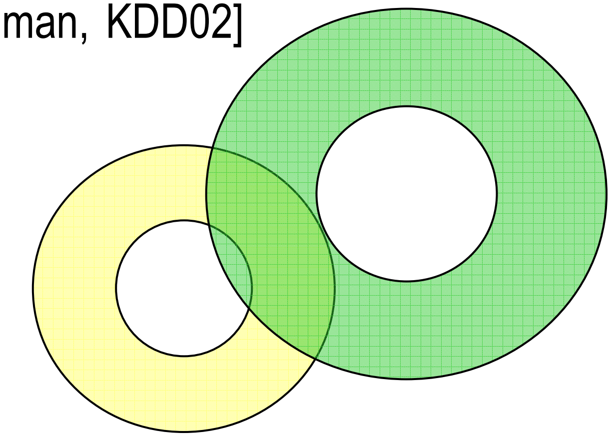
- Application:
  - merging records from multiple sources, using *complex* similarity functions
- Idea: similar records usually have *similar* feature values
- Algorithm:
  - create a key for every record (e.g., lastname[1..4])
  - sort data wrt the key
  - pair-wise comparison within a sliding window of size w
- Moral: Multi-pass + transitive closure > single-pass (large w)
- Limitations: only allow limited errors on the key

Fuzzy Blocking

- Bigram Indexing [Christen & Churches, Febrl, 2003]
- Allow small errors in the key by
  - requiring only a fraction of bigrams are preserved
  - insert the record into multiple blocks
- E.g., key value = “abcde”, and we require 70% bigrams preserved
  - Generate all 4 possible combinations, insert into corresponding blocks
  - e.g., {ab, bc, cd}, {ab, bc, de}, {ab, cd, de}, {bc, cd, de}

Canopy Clustering

- Canopy Clustering as a solution to tackle *hard* clustering problems [McCallumzy et al, KDD00] [Cohen & Richman, KDD02]
  - millions of points
  - many thousands of dimensions
  - many thousands of clusters



- Idea
  - Create overlapping canopies (i.e., special subsets)
  - Perform clustering but do not consider (x, y) if they never appear in one canopy

I-Match Algorithm

1. Doc ➔ Bag of tokens ➔ Sorted set of unique tokens ➔ Prune tokens wrt idf values ➔ SHA digest
   - "Hello World and Hello Web" ➔ … ➔ [and, Hello, Web, World] ➔ [Hello, Web, World] ➔ 0x685b…..
2. [d_1, SHA_1], [d_2, SHA_2], …
   - collision on SHA digest values ➔ near duplicate document

---

- 18K Web docs ➔ 83 sec (I-Match) vs ~590 sec (Shingling)
- It is shown that pruning token s.t. nidf(token) < 0.1 results in most accurate results for near-duplicate detection
  - effectively, *ignoring frequently occurring tokens*

SpotSigs [Jonathan et al, SIGIR07]

- Frequently occurring tokens are useful
  - Serve as anchors
  - Closely related to *document fingerprinting* methods
- SpotSigs
  - Choose set of <antecedent, spot dist>
    - e.g., <"are", 2>, <"to", 3>
  - Sig(till_here) = { "are" → "serve", "to" → "methods", "to" → "till_here"}
  - sim(X → Y) = | sig(X) ∩ sig(X) | / | sig(X) |

Roadmap

A. Motivation
B. Problem Definition and Scope
C. Similarity Join Algorithms
D. Epilogue
   1. Recurring ideas
   2. Performance comparison
   3. Open issues

Recurring Ideas /1

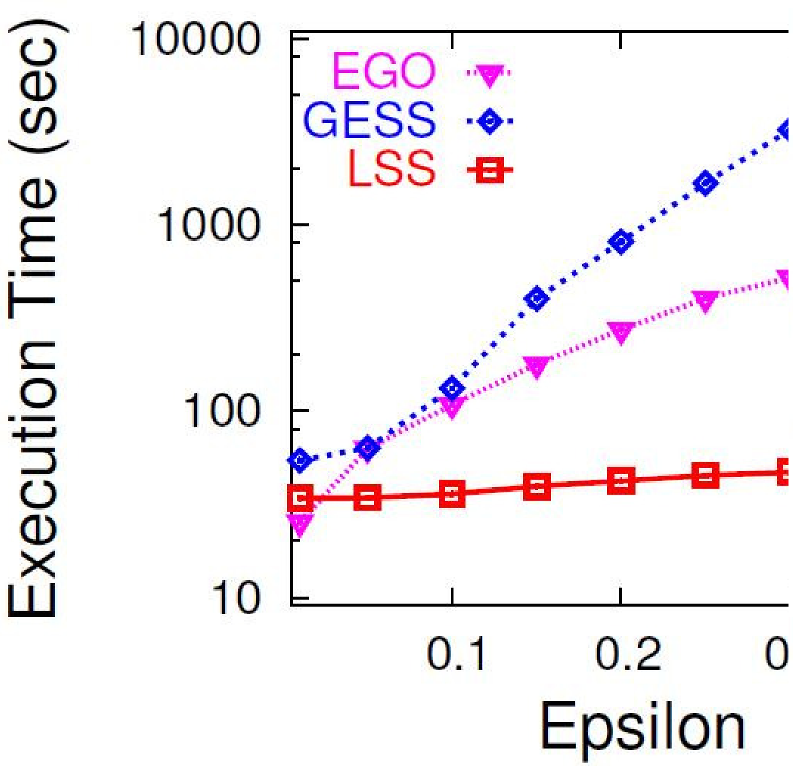
- Similar objects should be also similar in some feature space
  - MBRs in R-tree
  - M-tree
  - Randomized projection
  - Canopy (distance wrt a pivot)
- Replication
  - Replication in spatial join
  - Neighborhood generation
  - Hamming sim join, PartEnum

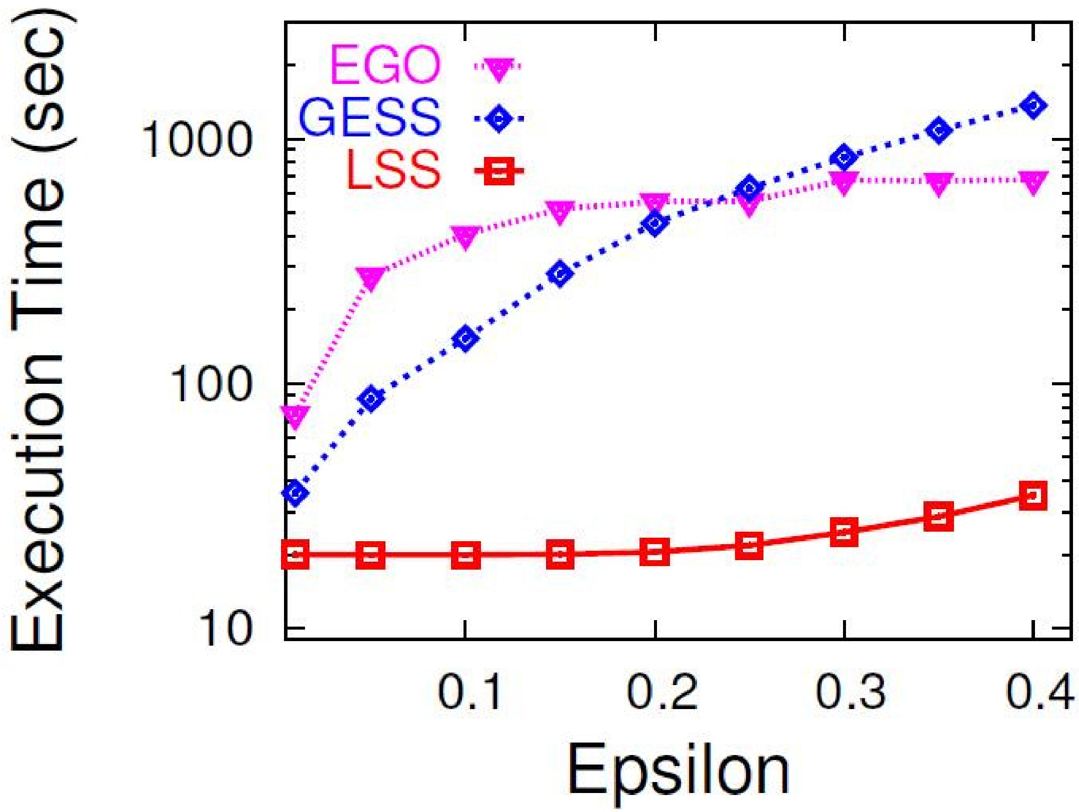
Recurring Ideas /2

- Index
  - Set containment join
  - All-Pairs, PPJoin+, Ed-Join
- Pruning
  - Derive lower/upper-bounding techniques to prune candidates as early as possible
- Partitioning
  - Length partition in All-Pairs, PartEnum
  - Reduce approximate distance threshold by Pigeon-hole principle

Performance: Vector Space

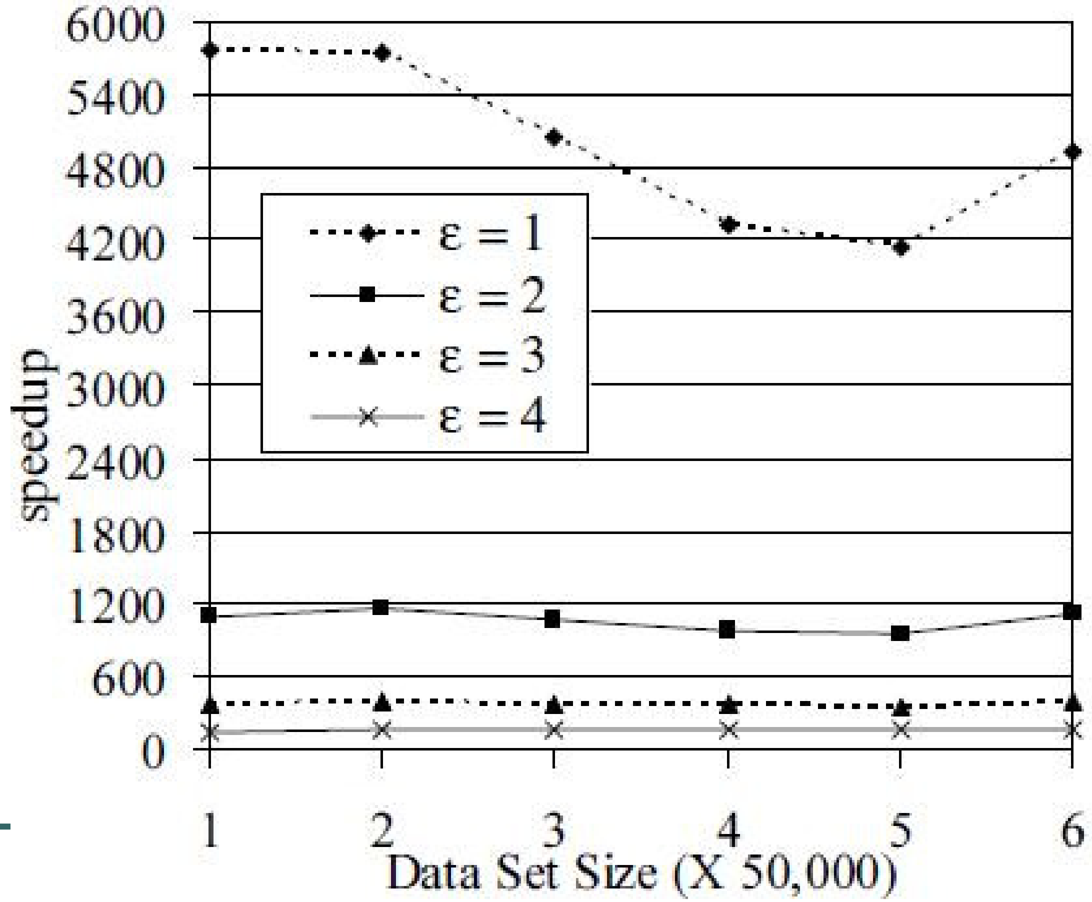
- Corel: ColorHistogram & LayoutHistogram (68K points, 32d)





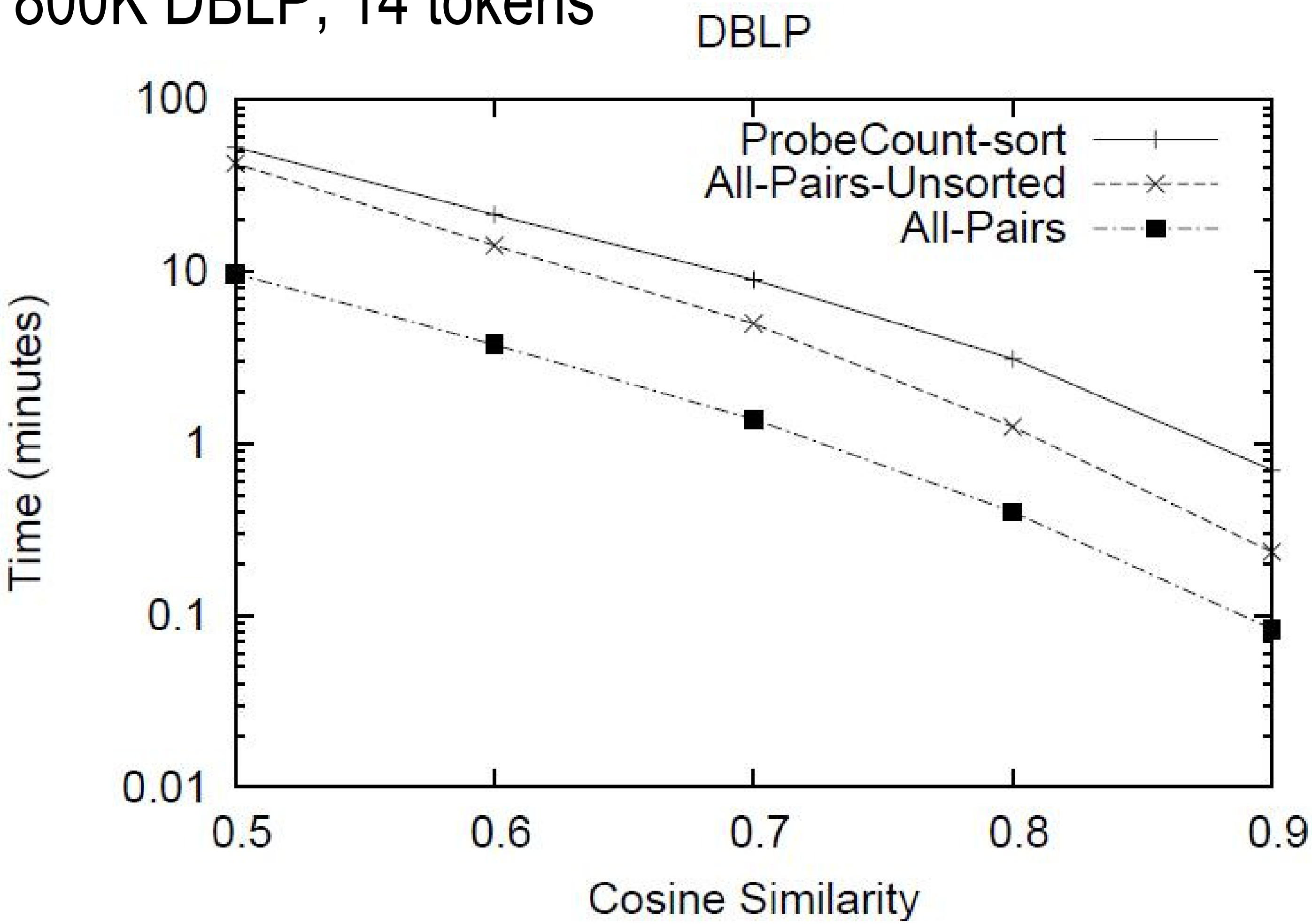
Performance: Metric Space

- Sentences, edit distance. Measures speedups



Performance: Set Similarity Join /1

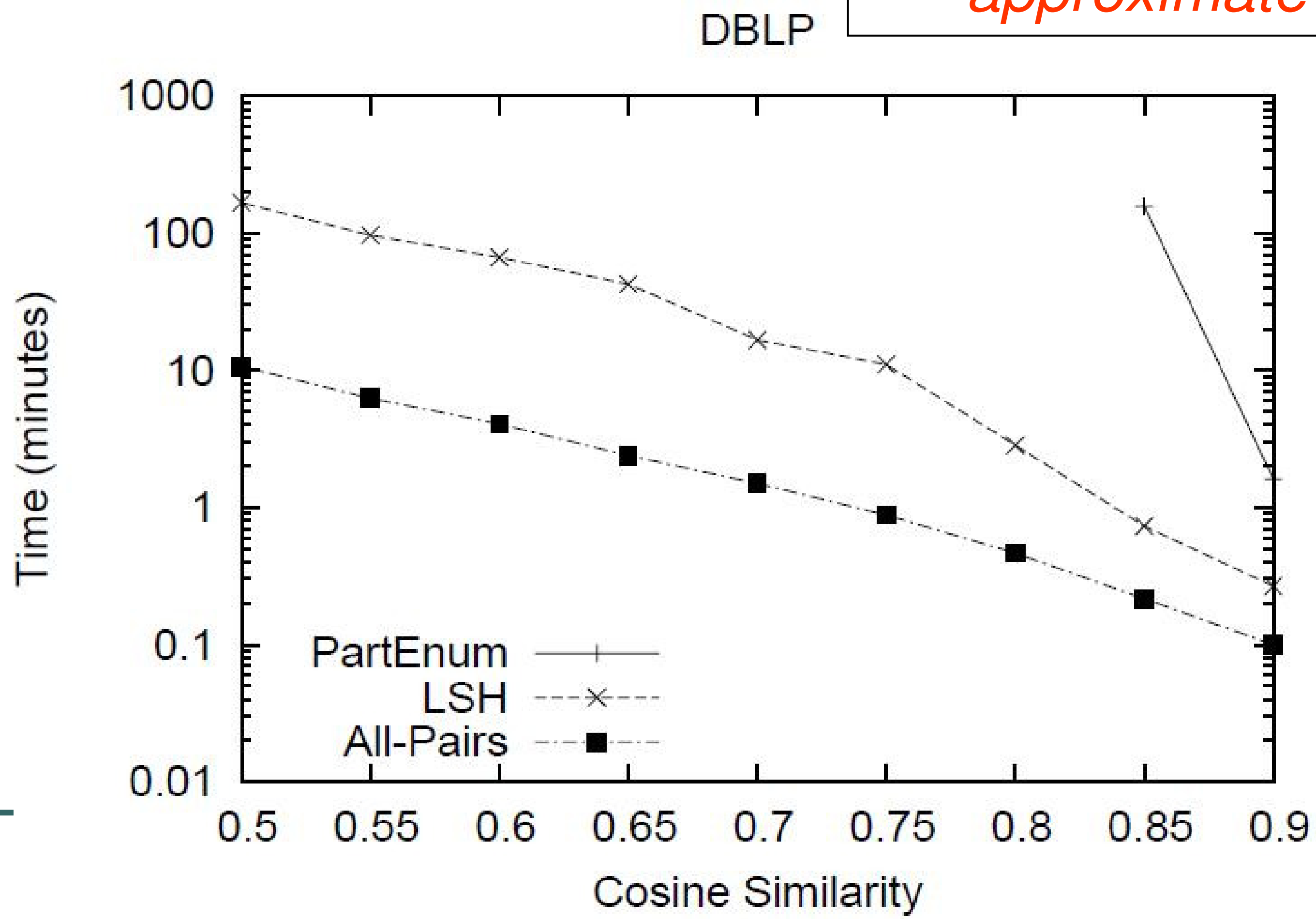
- 800K DBLP, 14 tokens



Performance: Set Similarity Join /2

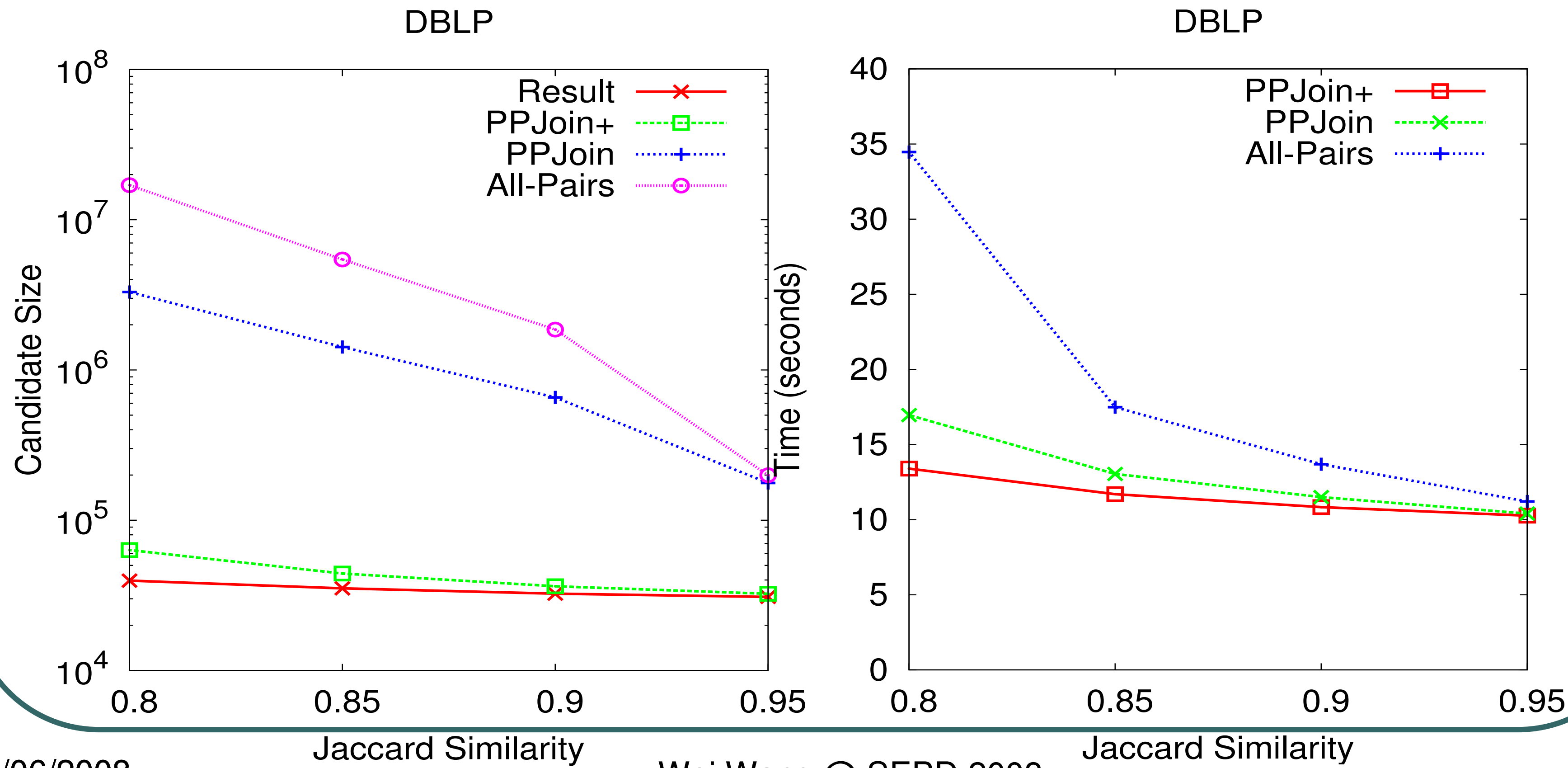
Exact sim join algorithm is even faster than an approximate one !

- 800K DBLP, 14 tokens



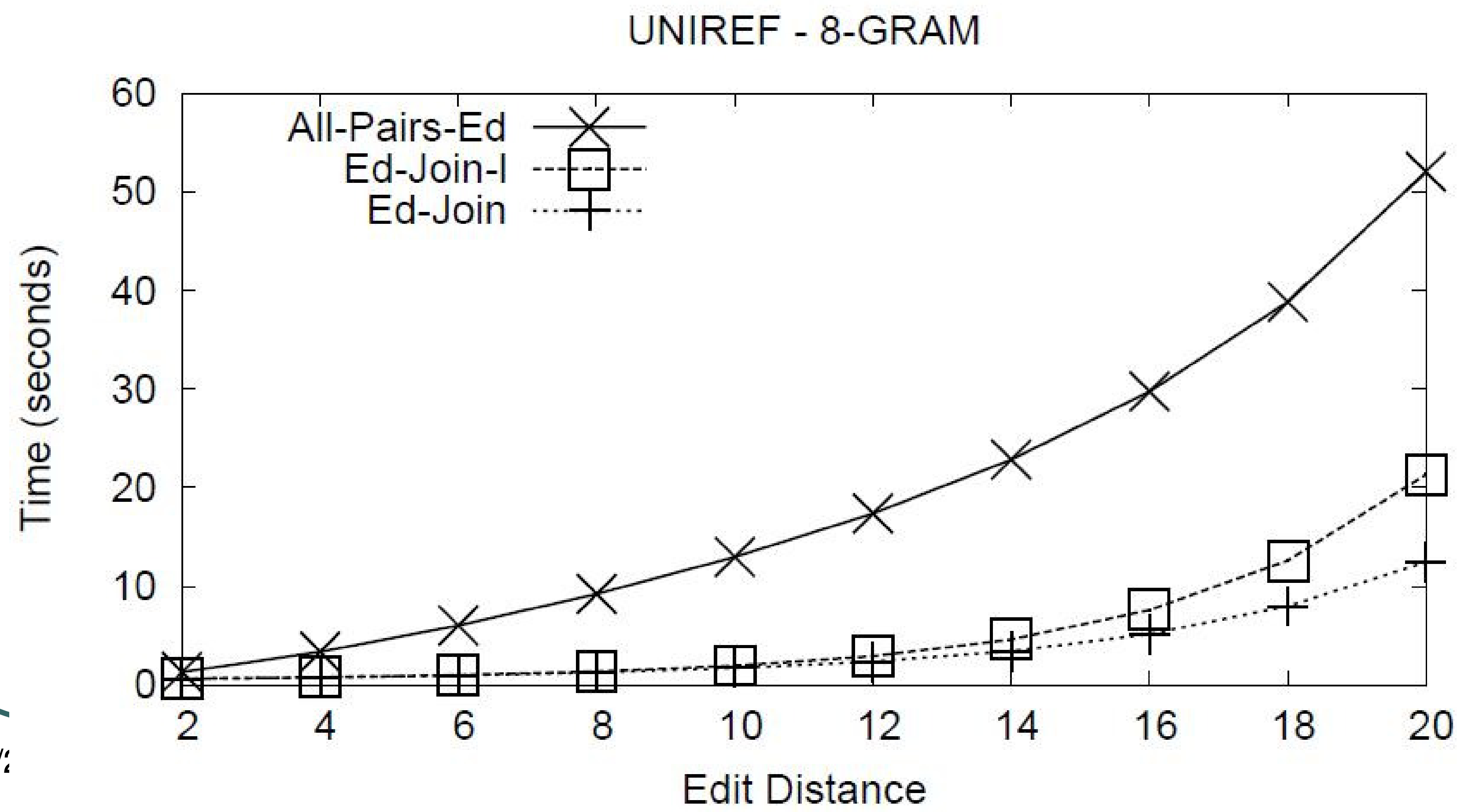
Performance: Set Similarity Join /3

- 873K DBLP, 14 tokens



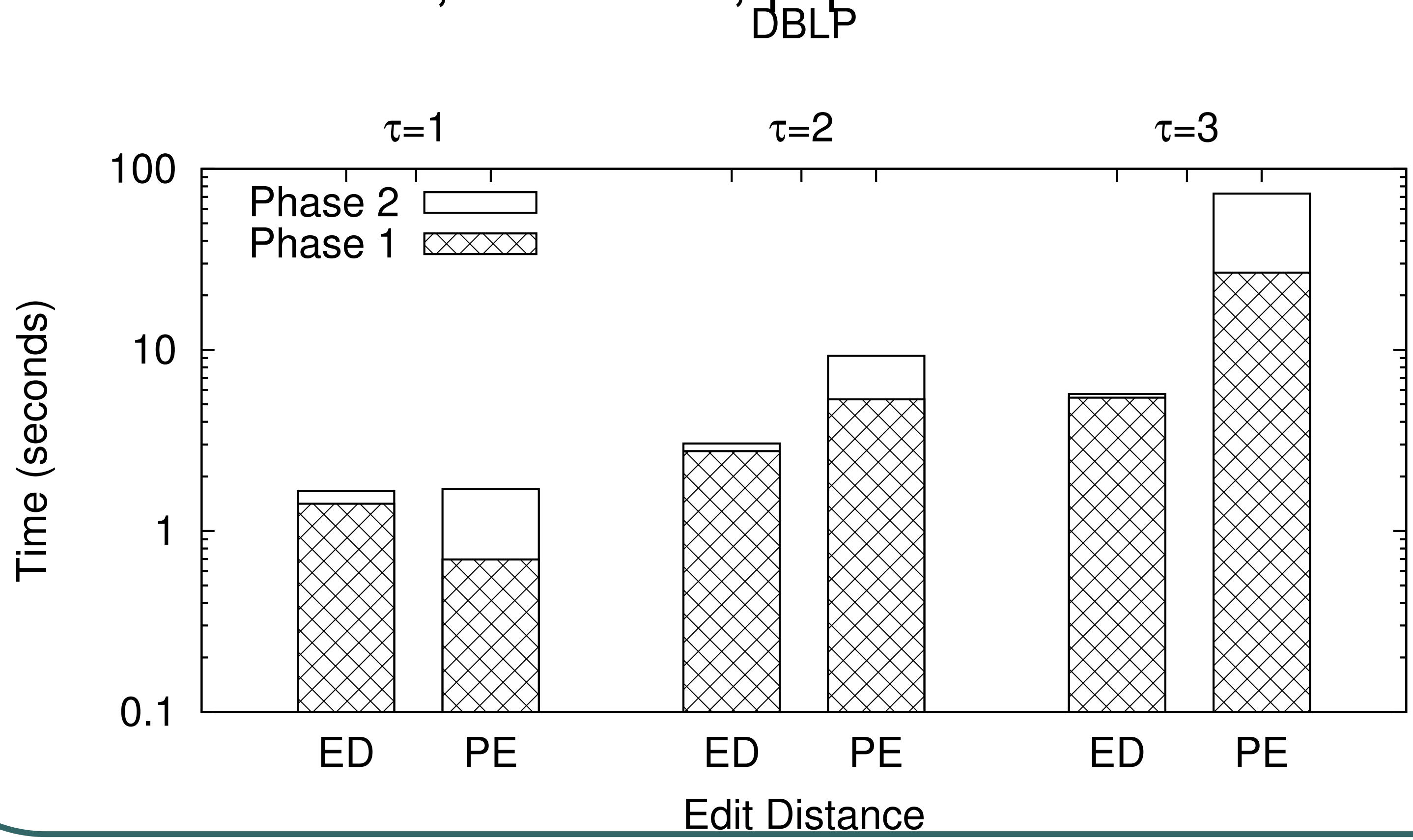
Performance: Edit Similarity Join /1

- 366K UNIREF protein sequences, 465 chars, |Σ|=25



Performance: Edit Similarity Join /2

- 863K DBLP, 105 chars, $|\Sigma|$=93



Open Issues /1

- Further optimization on performances
  - Index for similarity functions (e.g., cosine)
  - Better pruning techniques
  - Optimize for the specific similarity/distance function

Open Issues /2

- To the base of the iceberg
  - *Color histogram intersection*, *earth moving distance* in multimedia databases

$$\sum_i \min(x[i], y[i]) \;/\; \min(\sum_i x[i], \sum_j y[j])$$

  - *Dynamic time warping* in speech recognition and time-series databases

$$D(i,j) = d(i,j) + \min(D(i-1,j), D(i,j-1), D(i-1,j-1))$$

  - *Similarity functions for data integration / record linkage*

$$d_j(x,y) = \frac{1}{3}\left(\frac{m}{|x|} + \frac{m}{|y|} + \frac{m-t}{m}\right) \qquad d_{jw}(x,y) = d_j(x,y) + l \cdot p \cdot (1 - d_j(x,y))$$

Open Issues /2

- To the base of the iceberg
  - *Similarity functions for protein sequences*
    - Smith & Waterman local alignment vs. BLAST
  - *Tree edit distance* (Similarity between XML or Web documents)
    - [Yang et al, SIGMOD05]
  - *Graph distance* (isomorphism, maximal common subgraph, …)

Open Issues /3

- Think out of the square
  - Black-box style similarity function
    - e.g., from the output of a classifier [Chandel et al, SIGMOD07]
    - e.g., IR relevance model that depends on many parameters
    - e.g., similarity function that depends on external parameters
  - Query optimization problem
    - e.g., combination of multiple similarity functions [Chaudhuri et al, VLDB07]

Objectives Revisited

- Classify existing approaches along based on several perspectives
  - Euclidean space / metric space / other
  - Exact / approximate
- Explain several useful ideas in solving the problem
  - Partitioning
  - Lower/upper bounding
  - Similarity function specific filtering
  - Synopsis / signature

Q & A

Wei Wang: http://www.cse.unsw.edu.au/~weiw
Slides: http://www.cse.unsw.edu.au/~weiw/project/simjoin.html