String Similarity Join

Dong Deng

Real World Data is Dirty

- Misspellings of the query "*britney spears*" on Google

488941	britney spears	29	britent spears	9	brinttany spears
40134	brittany spears	29	brittnany spears	9	britanay spears
36315	brittney spears	29	britttany spears	9	britinany spears
24342	britany spears	29	btiney spears	9	britn spears
7331	britny spears	26	birttney spears	9	britnew spears
6633	briteny spears	26	breitney spears	9	britneyn spear
2696	britteny spears	26	brinity spears	9	britrney spear
1807	briney spears	26	britenay spears	9	brtiny spears
1635	brittny spears	26	britneyt spears	9	brtittney spears
1479	brintey spears	26	brittan spears	9	brtny spears
1479	britanny spears	26	brittne spears	9	brytny spears
1338	britiny spears	26	btittany spears	9	rbitney spears
1211	britnet spears	24	beitney spears	8	birtiny spears
1096	britiney spears	24	birteny spears	8	bithney spears
991	britaney spears	24	brightney spears	8	brattany spears
991	britnay spears	24	brintiny spears	8	breitny spears
811	brithney spears	24	britanty spears	8	breteny spears
811	brtiney spears	24	britenny spears	8	brightny spears
664	birtney spears	24	britini spears	8	brintay spears
664	brintney spears	24	britnwy spears	8	brinttey spears
664	briteney spears	24	brittni spears	8	briotney spears

http://marc.merlins.org/linux/talks/google/britney.html

Set Similarity Functions

- Overlap Size

$$overlap(x, y) = |x \cap y|$$

- Jaccard Similarity

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

x **= {A,B,C,D,E}** ***y*** **= {B,C,D,E,F}**
overlap(x, y) = 4 **J(x, y) = 4/6 = 0.67**

String Similarity Function

- Edit Distance ED(r, s): the minimum number of edit operations (insertion/deletion/substitution) needed to transform r to s.
- For example: ED(*hilton,* huston) = 2

h*i*lton

↓ *substitute i with u*

hu*l*ton

↓ *substitute l with s*

huston

- Edit Similarity: $\mathrm{EDS}(\mathrm{r},\mathrm{s}) = 1 - \frac{ED(r,s)}{\max(|r|,|s|)}$

Calculating Edit Distance

Calculating ED(r,s)

Let r_n and s_m be the last characters in r and s.

a) Match r_n and s_m

```
ED(r,s)= 0 + ED(r[1,n-1],s[1,m-1])
```

b) Substitute r_n with s_m

```
ED(r,s)= 1 + ED(r[1,n-1],s[1,m-1])
```

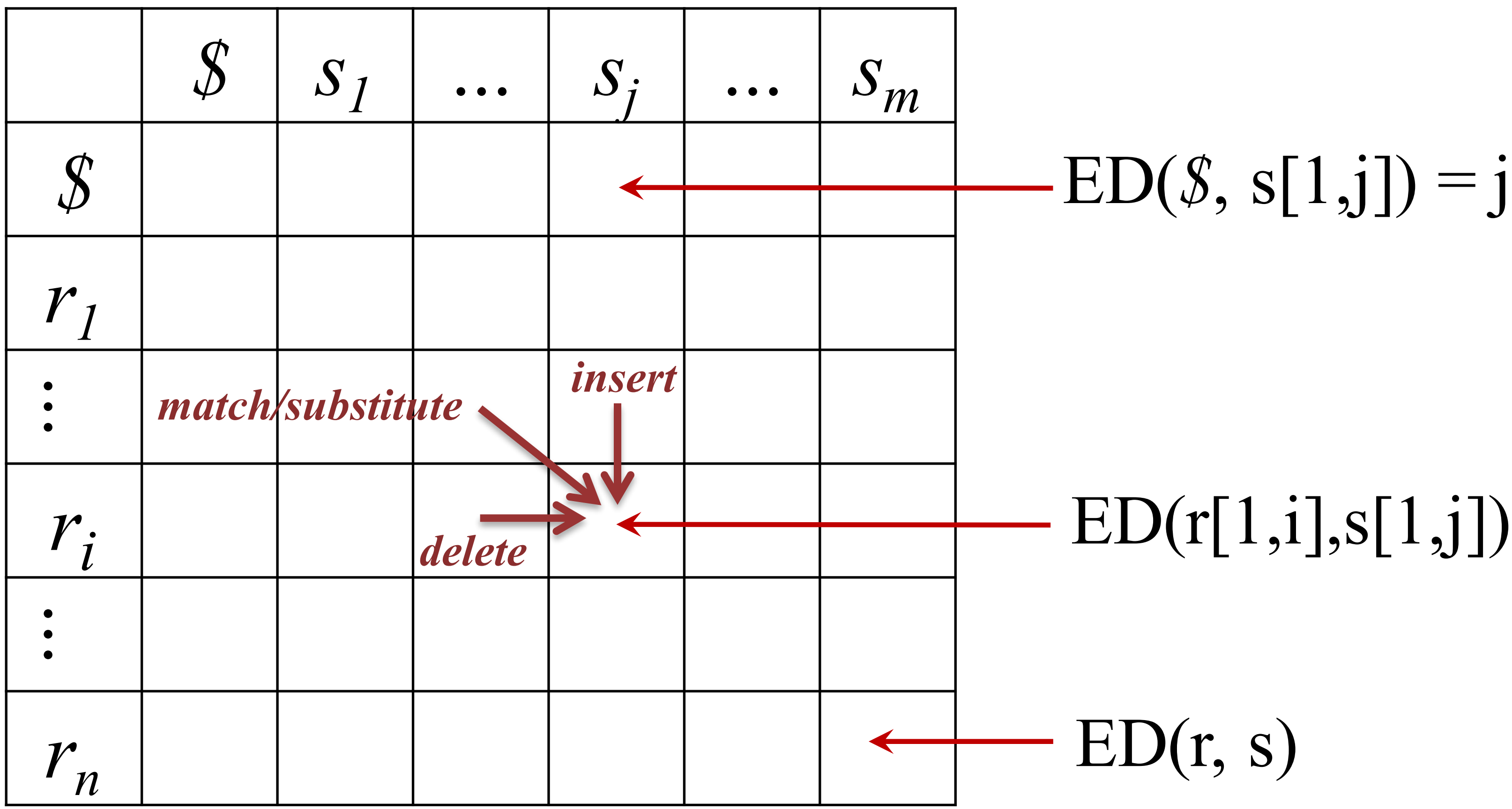
c) Delete r_n

```
ED(r,s)= 1 + ED(r[1,n-1],s[1,m])
```

d) Insert s_m

```
ED(r,s)= 1 + ED(r[1,n],s[1,m-1])
```

Calculating Edit Distance



Calculating Edit Distance

	$	b	r	i	t	n	e	y
$	0	1	2	3	4	5	6	7
b	1							
t	2							
i	3							
n	4							
e	5							
y	6							

Calculating Edit Distance

	\$	b	r	i	t	n	e	y
\$	0	1	2	3	4	5	6	7
b	1	0						
t	2							
i	3							
n	4							
e	5							
y	6							

Calculating Edit Distance

	$	b	r	i	t	n	e	y
$	0	1	2	3	4	5	6	7
b	1	0	1					
t	2							
i	3							
n	4							
e	5							
y	6							

Calculating Edit Distance

	$	b	r	i	t	n	e	y
$	0	1	2	3	4	5	6	7
b	1	0	1	2				
t	2							
i	3							
n	4							
e	5							
y	6							

Calculating Edit Distance

	$	b	r	i	t	n	e	y
$	0	1	2	3	4	5	6	7
b	1	0	1	2	3	4	5	6
t	2							
i	3							
n	4							
e	5							
y	6							

Calculating Edit Distance

	$	b	r	i	t	n	e	y
$	0	1	2	3	4	5	6	7
b	1	0	1	2	3	4	5	6
t	2	1	1	2	3	4	5	6
i	3							
n	4							
e	5							
y	6							

Calculating Edit Distance

	$	b	r	i	t	n	e	y
$	0	1	2	3	4	5	6	7
b	1	0	1	2	3	4	5	6
t	2	1	1	2	3	4	5	6
i	3	2	2	1	2	3	4	5
n	4	3	3	2	2	2	3	4
e	5	4	4	3	3	3	2	3
y	6	5	5	4	4	4	3	2

Calculating Edit Distance

	$	b	r	i	t	n	e	y
$	0	1	2	3	4	5	6	7
b	1	0	1	2	3	4	5	6
t	2	1	1	2	3	4	5	6
i	3	2	2	1	2	3	4	5
n	4	3	3	2	2	2	3	4
e	5	4	4	3	3	3	2	3
y	6	5	5	4	4	4	3	**2**

Calculating Edit Distance

	$	b	r	i	t	n	e	y
$	0	1	2	3	4	5	6	7
b	1	0	1	2	3	4	5	6
t	2	1	*substitute* 1	2	3	4	5	6
i	3	2	2	1	*delete* 2	3	4	5
n	4	3	3	2	2	2	3	4
e	5	4	4	3	3	3	2	3
y	6	5	5	4	4	4	3	2

From s=‘*btiney*’ to r=‘*britney*’

Match s[1]=‘b’ → *b*

Substitute s[2]=‘r’ with ‘t’→*bt*

Match s[3] = ‘i’ → *bti*

Delete s[4]=‘t’→ *bti*

Match s[5]=‘n’ → *btin*

Match s[6]=‘e’ → *btine*

Match s[7]=‘y’ → *btiney* = r

What's the difference?

Functions (normalization)	**Edit Distance** **Edit Similarity**	**Overlap Size** **Jaccard Similarity**
Input	Sequences	Sets
Example Representations	DNA, String, Time Series	Image, Document, Vector, Friend List

String Similarity Join

- Input:
  - A collection of strings S
  - A threshold τ
- Output:
  - All string pairs $(s, r) \in S \times S \; such \; that \; ED(s, r) \leq \tau$

String Similarity Join

- Give threshold $\tau = 3$

ID	Strings
s_1	vankatesh
s_2	avataresha
s_3	kaushic chaduri
s_4	kaushik chakrab
s_5	kaushuk chadhui
s_6	caushik chakrabar

ED(s_1 ,s_2)=5 ED(s_1 ,s_3)=13 ED(s_1 ,s_4)=12 ED(s_1 ,s_5)=12
ED(s_1 ,s_6)=14 ED(s_2 ,s_3)=12 ED(s_2 ,s_4)=12 ED(s_2 ,s_5)=12
ED(s_2 ,s_6)=14 ED(s_3 ,s_4)=5 ED(s_3 ,s_5)=4 ED(s_3 ,s_6)=8
ED(s_4 ,s_5)=4 **ED(s_4 ,s_6)=3** ED(s_5 ,s_6)=8

Data Cleaning & Integration

Relation with Duplicates

ID	name	ZIP	Income
P1	Green	51519	30k
P2	Green	51518	32k
P3	Peter	30528	40k
P4	Peter	30528	40k
P5	Gree	51519	55k
P6	Chuck	51519	30k

Challenges

$O(n^2)$ *pairs of strings.*

1 million strings result in 1 trillion pairs !

$O(|r||s|)$ *time to calculate ED(r, s)*

Filter-and-Refine Framework

- Basic idea
  - **Filter** a large number of dissimilar string pairs
  - **Verify** the remaining potentially similar pairs
- Good Filter Condition
  - Efficient to check
  - Effective for pruning dissimilar pairs

Length Filter

- ED(r, s): The minimum number of edit operations (insertion/deletion/substitution) needed to transform r to s.

- *Property:* $\mathrm{ED}(\boldsymbol{r}, \boldsymbol{s}) \geq \big||\boldsymbol{r}|-|\boldsymbol{s}|\big|$
  - it needs at least $\big||r|-|s|\big|$ deletions to just make r and s have the same length

Applying the Length Filter

- Give threshold $\tau = 3$
- Pruning Condition:

$$\big| |s_i| - |s_j| \big| > 3$$

ID	Strings	Length
s_1	vankatesh	9
s_2	avataresha	10
s_3	kaushic chaduri	15
s_4	kaushik chakrab	15
s_5	kaushuk chadhui	15
s_6	caushik chakrabar	17

ED(s_1 ,s_2)=5 ~~ED(s_1 ,s_3)=13~~ ~~ED(s_1 ,s_4)=12~~ ~~ED(s_1 ,s_5)=12~~
~~ED(s_1 ,s_6)=14~~ ~~ED(s_2 ,s_3)=12~~ ~~ED(s_2 ,s_4)=12~~ ~~ED(s_2 ,s_5)=12~~
~~ED(s_2 ,s_6)=14~~ ED(s_3 ,s_4)=5 ED(s_3 ,s_5)=4 ED(s_3 ,s_6)=8
ED(s_4 ,s_5)=4 **ED(s_4 ,s_6)=3** ED(s_5 ,s_6)=8

Partitioning Filter

- Give threshold $\tau = 1$

*hi*lton

1

huston

hi dose not appear in huston and needs at least 1 edit

Partitioning Filter

- Give threshold $\tau = 1$

hi*lton*

1

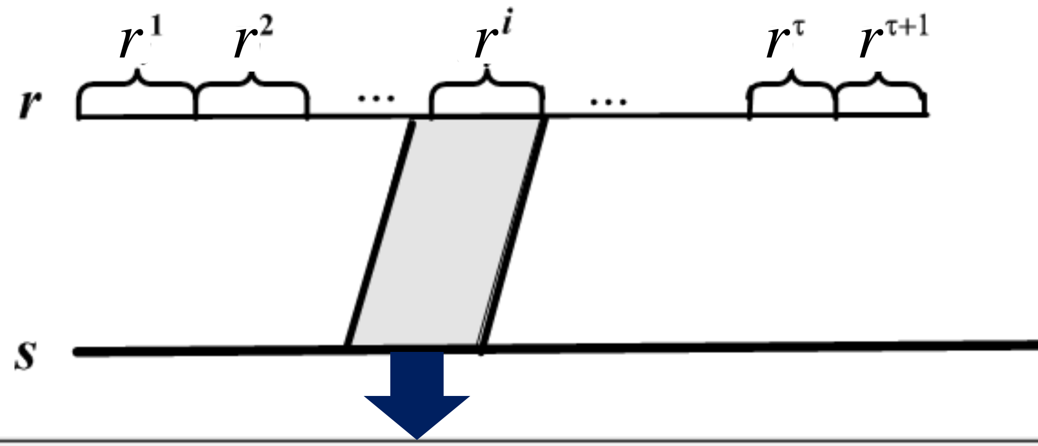
huston

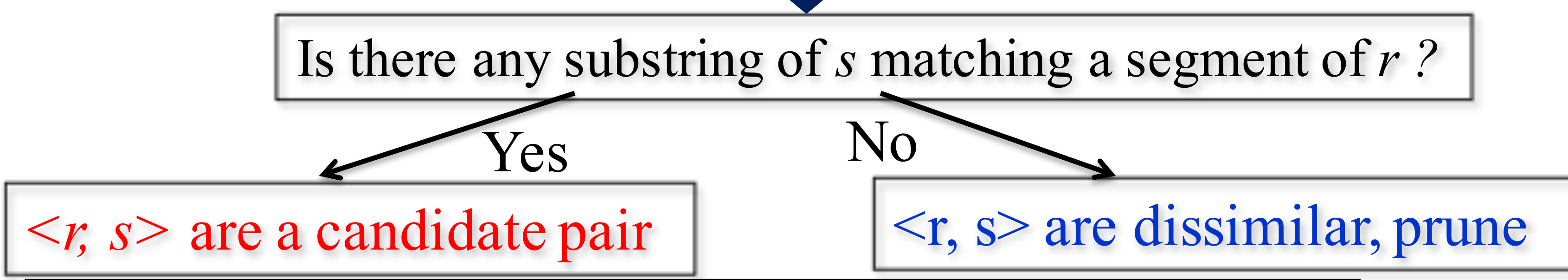
minimum # of edit operations is 2. Prune!

Partitioning Filter

- Threshold τ
- String r
- String s

split r to τ +1 disjoint segments





How to Partition?

- Give threshold $\tau = 1$

hil *ton*

Match

hus ton

Candidate!

Partition Scheme

- Even Partition Scheme
  - Given $\tau = 3$, "*avataresha*" → {"*av*", "*at*", "*are*", "*sha*"}

- Other Schemes
  - Select good partition strategies.
  - Adaptive partition scheme [Deng et al. 2012a].

Challenge

there are $|s|^2$ *substrings in s*

how to reduce the number of substrings
to compare with for each segment?

Length-based Method

- For each segment, only compare to the substrings with the same length as the segment.

r = "vankatesh" $\tau = 3$ s = "avataresha"

r^1	va	av va at ta ar re es sh ha
r^2	nk	av va at ta ar re es sh ha
r^3	at	av va at ta ar re es sh ha
r^4	esh	ava vat ata tar are res esh sha

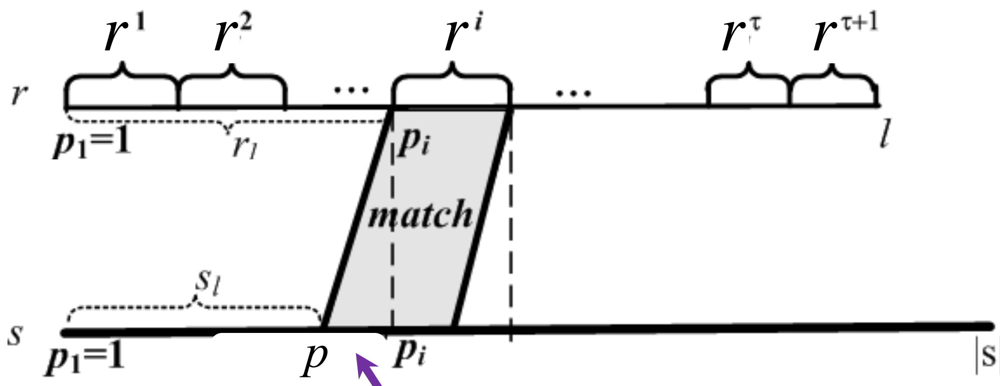
Length-based Method

- For any strings r, s, and τ, the number of comparisons:

$$(\tau+1)(|s|+1)-|r|$$

- For $r = $ "$\texttt{vankatesh}$" and $s = $ "$\texttt{avataresha}$" , the number is 35

Shift-based Method



Just transform $\boldsymbol{r_l}$ *to* $\boldsymbol{s_l}$ *needs at least* $\big||s_l|-|r_l|\big|$ *edit operations*

Pruning Condition: $\big||s_l|-|r_l|\big| > \tau \longrightarrow |p - p_i| > \tau$

- For each segment $\boldsymbol{r^i}$ with the start position $\boldsymbol{p_i}$, only compares to the substrings with start positions $\boldsymbol{p}$ in $[\boldsymbol{p_i} - \boldsymbol{\tau}, \boldsymbol{p_i} + \boldsymbol{\tau}]$

Shift-based Method

- For each segment r^i with the start position p_i, only compares to the substrings with start positions p in $[p_i - \tau, p_i + \tau]$

$r =$ "vankatesh" $\tau = 3$ $s =$ "avataresha"

r^1	va	←	av va at ta
r^2	nk	←	av va at ta ar re
r^3	at	←	va at ta ar re es sh
r^4	esh	←	tar are res esh sha

Shift-based Method

- For any strings r, s, and τ, the number of comparisons:

$$(\tau + 1)(2\tau + 1)$$

- For $r = $ “$\texttt{vankatesh}$” and $s = $ “$\texttt{avataresha}$”, the number is 22.

Position-aware Method

r = “vankatesh” $\Longrightarrow$ {va, nk, at, esh}

r_l r_r

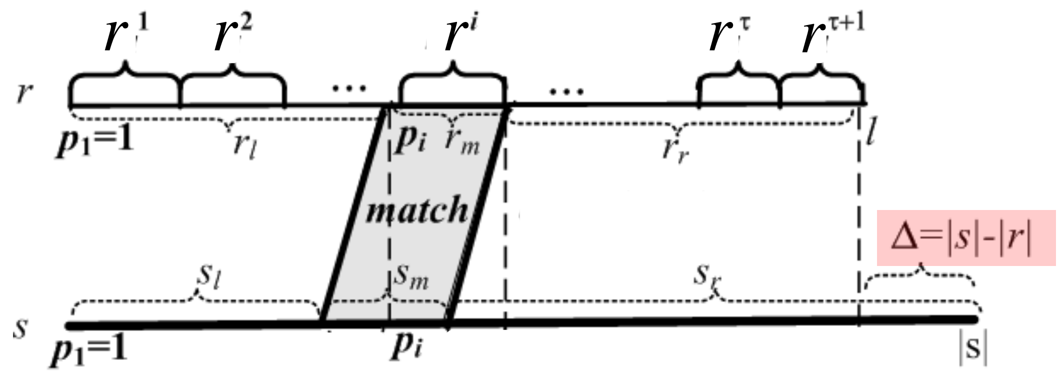
s = “avataresha”

s_l s_r

Transform $\boldsymbol{r_l}$ *to* $\boldsymbol{s_l}$*, match “at”, and transform* $\boldsymbol{r_r}$ *to* $\boldsymbol{s_r}$

$$\big||s_l|-|r_l|\big| + \big||s_r|-|r_r|\big| = 2 + 3 > \tau = 3$$

Position-aware Method



Transform $\boldsymbol{r_l}$ *to* $\boldsymbol{s_l}$*, match* $\boldsymbol{r_m}$ *and* $\boldsymbol{s_m}$ *then transform* $\boldsymbol{r_r}$ *to* $\boldsymbol{s_r}$

Pruning Condition: $||s_l|-|r_l|| + ||s_r|-|r_r|| > \tau$

- For each segment r^i with the start position p_i, only compare to the substrings with start position in $[P_i - \lfloor\frac{\tau-\Delta}{2}\rfloor, P_i + \lfloor\frac{\tau+\Delta}{2}\rfloor]$ where $\Delta = |s| - |r|$

Position-aware Method

- For each segment r^i with the start position p_i, only compare to the substrings with start position in $[P_i - \lfloor \frac{\tau-\Delta}{2} \rfloor, P_i + \lfloor \frac{\tau+\Delta}{2} \rfloor]$ where $\Delta = |s| - |r|$

$\tau = 3$ $r =$ "vankatesh" $s =$ "avataresha"

r^1 va ← av va at

r^2 nk ← va at ta ar

r^3 at ← ta ar re es

r^4 esh ← res esh sha

Position-aware Method

- For any strings r, s, and τ, the number of comparisons:

$$(\tau + 1)^2$$

- For $r = $ "$\texttt{vankatesh}$" and $s = $ "$\texttt{avataresha}$", the number is 14.

Multi-match-aware Method

-- Left-side Perspective

r = "vankatesh" $\Longrightarrow$ {va, nk, at, esh}

$\tau = 3$

r_l = ""

3 unvisited segments

1 1 1

+

$||s_l|-|r_l||=1$

$=4 > \tau$

s = "avataresha"

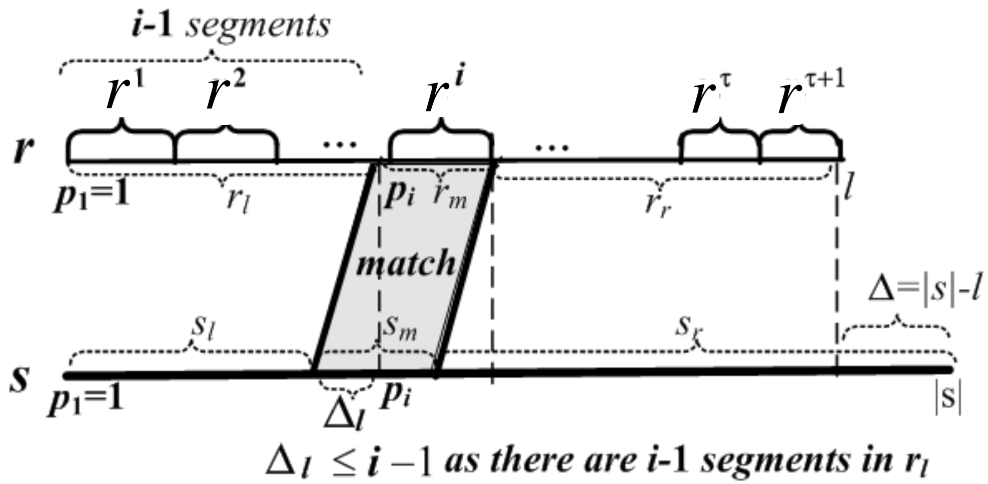
s_l = "a"

If none of the 3 unvisited segments having matching substring in s, then r and s cannot be similar

Thus we can safely skip the current matching segment and look for the next

Multi-match-aware Method

-- Left-side Perspective



Pruning Condition: $\big||s_l|-|r_l|\big|$ + (# of unvisited segments) > τ

- For each segment r^i with the start position p_i, only compare to the substrings with start position in $[P_i-(i-1), P_i+(i-1)]$

Multi-match-aware Method

-- *Left-side Perspective*

- For each segment r^i with the start position p_i , only compare to the substrings with start position in $[\boldsymbol{P_i - (i-1), P_i + (i-1)}]$

$\tau = 3$ $r =$ "vankatesh" $s =$ "avataresha"

r^1 va $\leftarrow$ av

r^2 nk $\leftarrow$ va at ta

r^3 at $\leftarrow$ at ta ar re es

r^4 esh $\leftarrow$ tar are res esh sha

Multi-match-aware Method

-- Left-side Perspective

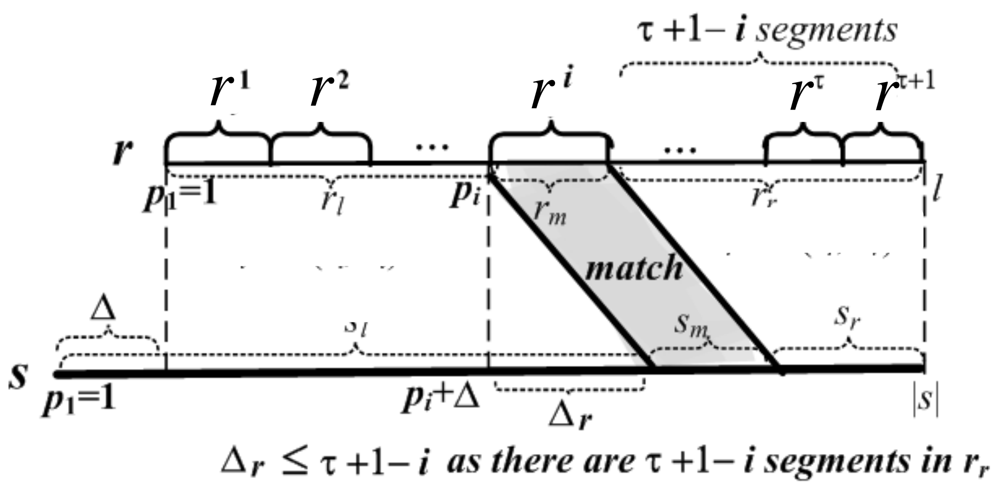
- For any strings r, s, and τ, the number of comparisons:

$$\tau^2 + 2\tau$$

- For $r =$ "$\mathtt{vankatesh}$" and $s =$ "$\mathtt{avataresha}$", the number is 14.

Multi-match-aware Method

-- Right-side Perspective



Pruning Condition: (# unvisited segments)$+\left| |s_r| - |r_r| \right| > \tau$

- For each segment r^i with the start position p_i, only compare to the substrings with start position in $[P_i + \Delta - (\tau + 1 - i), P_i + \Delta + (\tau + 1 - i)]$

Multi-match-aware Method

- Interestingly, we can apply the multi-match-aware method from left- and right-side perspectives simultaneously.

- For each segment $\boldsymbol{r}^{i}$ with start position $\boldsymbol{p}_{i}$, only compare to the substrings with start position in

$$[\boldsymbol{max}(\boldsymbol{P}_i - (\boldsymbol{i}-\mathbf{1}), \boldsymbol{P}_i + \boldsymbol{\Delta} - (\boldsymbol{\tau} + \mathbf{1} - \boldsymbol{i})), \boldsymbol{min}(\boldsymbol{P}_i + (\boldsymbol{i}-\mathbf{1}), \boldsymbol{P}_i + \boldsymbol{\Delta} + (\boldsymbol{\tau} + \mathbf{1} - \boldsymbol{i}))]$$

Multi-match-aware Method

- For each segment r^i with start position p_i, only compare to the substrings with start position in $[max(P_i-(i-1), P_i+\Delta-(\tau+1-i)),$ $min(P_i+(i-1), P_i+\Delta+(\tau+1-i))]$

$\tau = 3 \quad r = \text{“vankatesh”} \quad s = \text{“avataresha”}$

L_9^1 va $\longleftarrow$ av

L_9^2 nk $\longleftarrow$ va at ta

L_9^3 at $\longleftarrow$ ar re es

L_9^4 esh $\longleftarrow$ sha

Multi-match-aware Method

- For any strings r, s, and τ, the number of comparisons:

$$\left\lfloor \frac{\tau^2 - \Delta^2}{2} \right\rfloor + \tau + 1$$

- For $r =$ "$\texttt{vankatesh}$" and $s =$ "$\texttt{avataresha}$", the number is 8.

Theoretical Results

- The number of comparisons by the multi-match-aware method is **minimum** while guarantees completeness

- For any s, r and τ,

$$W_{multi\text{-}match}(s, r, \tau) \subseteq W_{position}(s, r, \tau) \subseteq W_{shift}(s, r, \tau) \subseteq W_{length}(s, r, \tau)$$

Applying Partitioning Filter

Applying Partitioning Filter

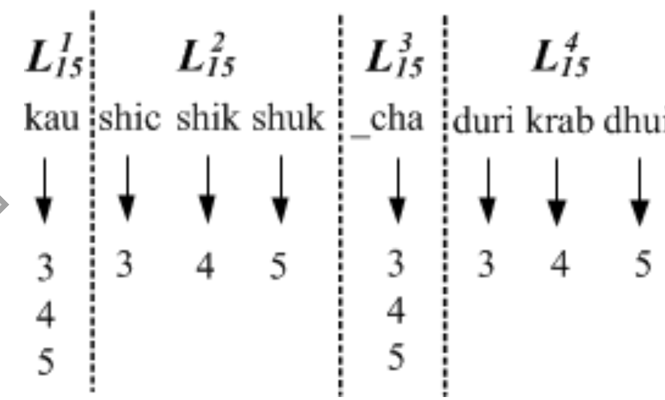
- 1. Group all the strings by length: **S**

ID	Strings	Length	
s_1	vankatesh	9	$\boldsymbol{S}_9$
s_2	avataresha	10	$\boldsymbol{S}_{10}$
s_3	kaushic chaduri	15	
s_4	kaushik chakrab	15	$\boldsymbol{S}_{15}$
s_5	kaushuk chadhui	15	
s_6	caushik chakrabar	17	$\boldsymbol{S}_{17}$

Applying Partitioning Filter

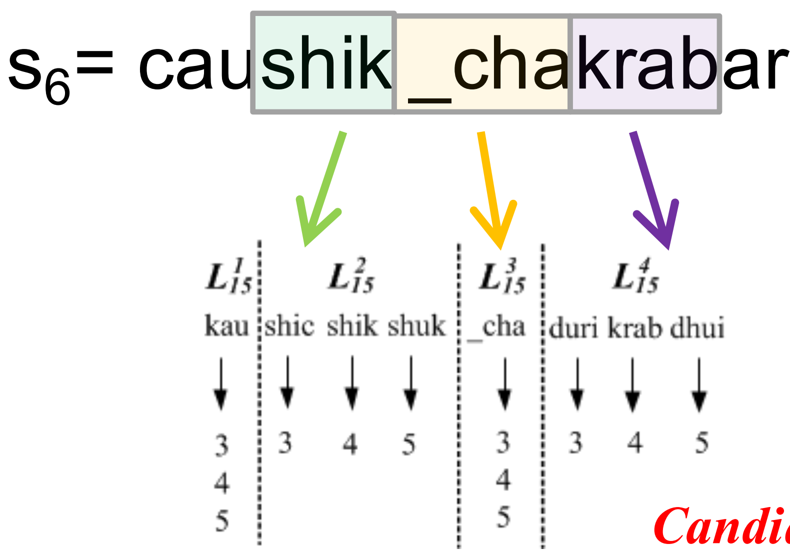
- 2. For each group S_l, partition its strings into $\tau + 1$ segments and build $\tau + 1$ inverted indexes L_l^i

S_{15}	1	2	3	4
s_3=	kau	shic	_cha	duri
s_4=	kau	shik	_cha	krab
s_5=	kau	shuk	_cha	dhui



Applying Partitioning Filter

- 3. For each string ***s*** and index $\boldsymbol{L_l^i}$, select substrings from ***s*** based on the partitioning filter to get candidates:



$$[\boldsymbol{max}(\boldsymbol{P_i} - (\boldsymbol{i}-\mathbf{1}), \boldsymbol{P_i} + \boldsymbol{\Delta} - (\boldsymbol{\tau} + \mathbf{1} - \boldsymbol{i})),$$
$$\boldsymbol{min}(\boldsymbol{P_i} + (\boldsymbol{i}-\mathbf{1}), \boldsymbol{P_i} + \boldsymbol{\Delta} + (\boldsymbol{\tau} + \mathbf{1} - \boldsymbol{i}))]$$

the segments in $\boldsymbol{L_l^i}$ *have the same start positions* p_i *and same string lengths* l

***Candidates*:** $<s_3, s_6>$ $<s_4, s_6>$ $<s_5, s_6>$

Applying Partitioning Filter

- 4. Verify the candidates

***Candidates*:** $<s_3, s_6>$ $<s_4, s_6>$ $<s_5, s_6>$

$$ED(s_3, s_6) > 3$$
$$ED(s_4, s_6) = 3$$
$$ED(s_5, s_6) > 3$$

Improving Verification

Calculating Edit Distance

	$	b	r	i	t	n	e	y
$	0	1	2	3	4	5	6	7
b	1	0	1	2	3	4	5	6
t	2	1	1	2	3	4	5	6
i	3	2	2	1	2	3	4	5
n	4	3	3	2	2	2	3	4
e	5	4	4	3	3	3	2	3
y	6	5	5	4	4	4	3	**2**

Verification

ED(r[1,i],s[1,j]) $\geq$ |i - j|

	$	**b**	**r**	**i**	**t**	**n**	**e**	**y**
$	0	1	2	3	4	5	6	7
b	1	0	1	2	3	4	5	6
t	2	1	1	2	3	4	5	6
i	3	2	2	1	2	3	4	5
n	4	3	3	2	2	2	3	4
e	5	4	4	3	3	3	2	3
y	6	5	5	4	4	4	3	2

Verification

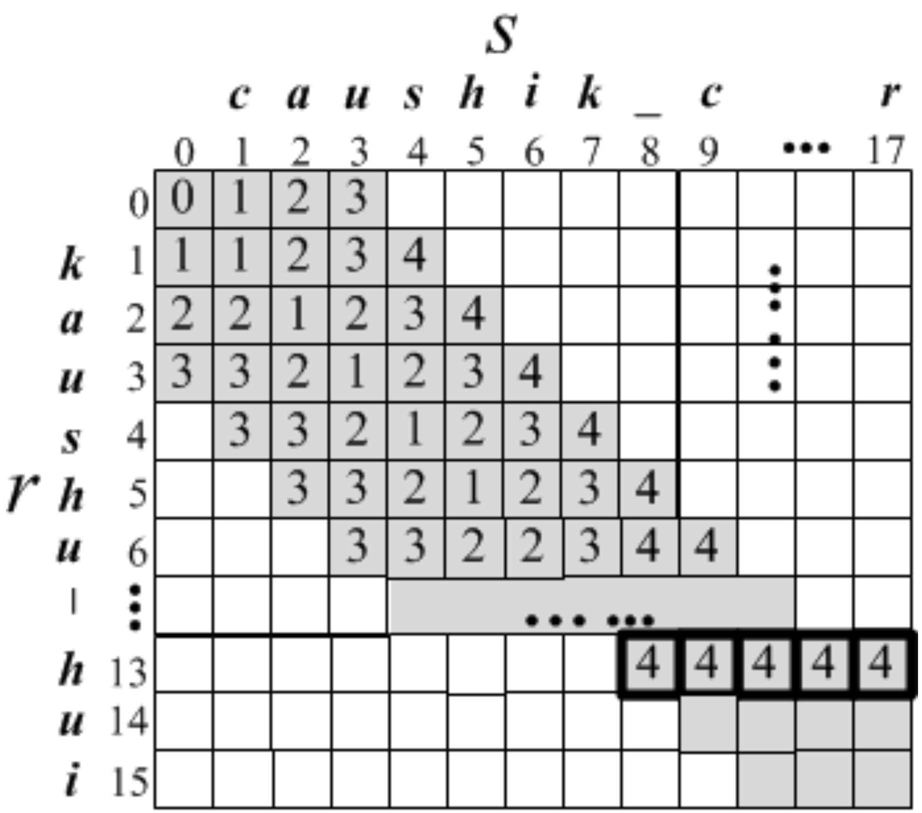
	$	b	r	i	t	n	e	y
$	0	1	2					
b	1	0	1	2				
t	2	1	1	2	3			
i		2	2	1	2	3		
n			3	2	2	2	3	
e				3	3	3	2	3
y					4	4	3	2

Verification

only need to calculate a band of width 2τ+1

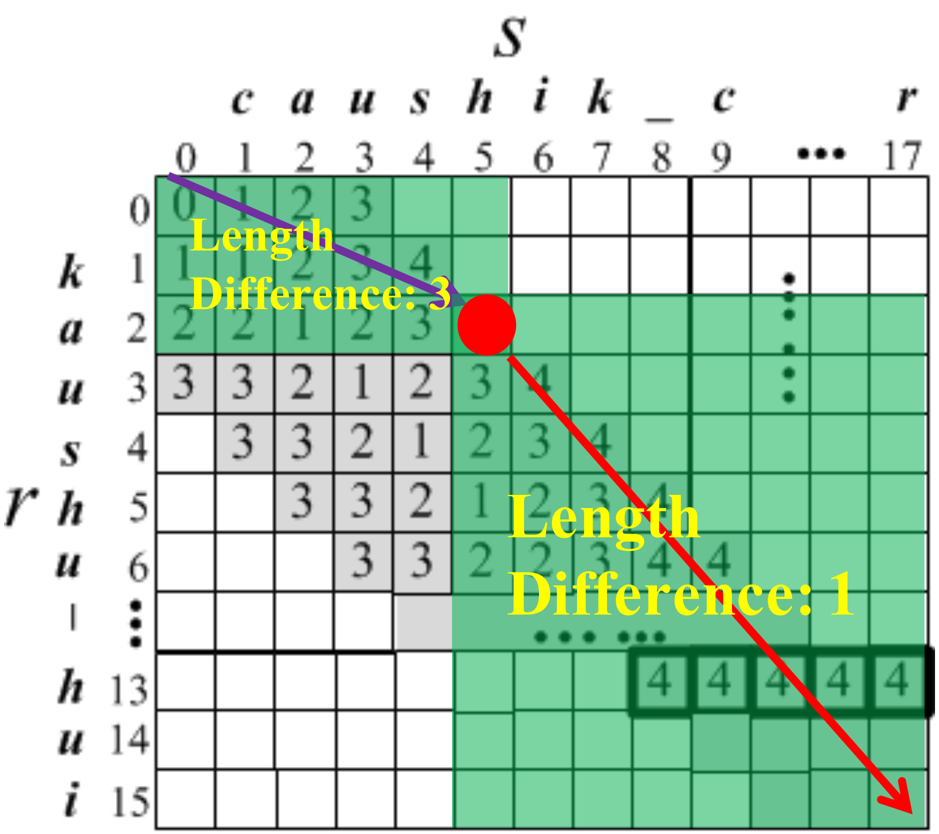
	$	b	r	i	t	n	e	y
$	0	1	2					
b	1	0	1	2				
t	2	1	1	2	3			
i		2	2	1	2	3		
n			3	2	2	2	3	
e				3	3	3	2	3
y					4	4	3	2

Early Termination



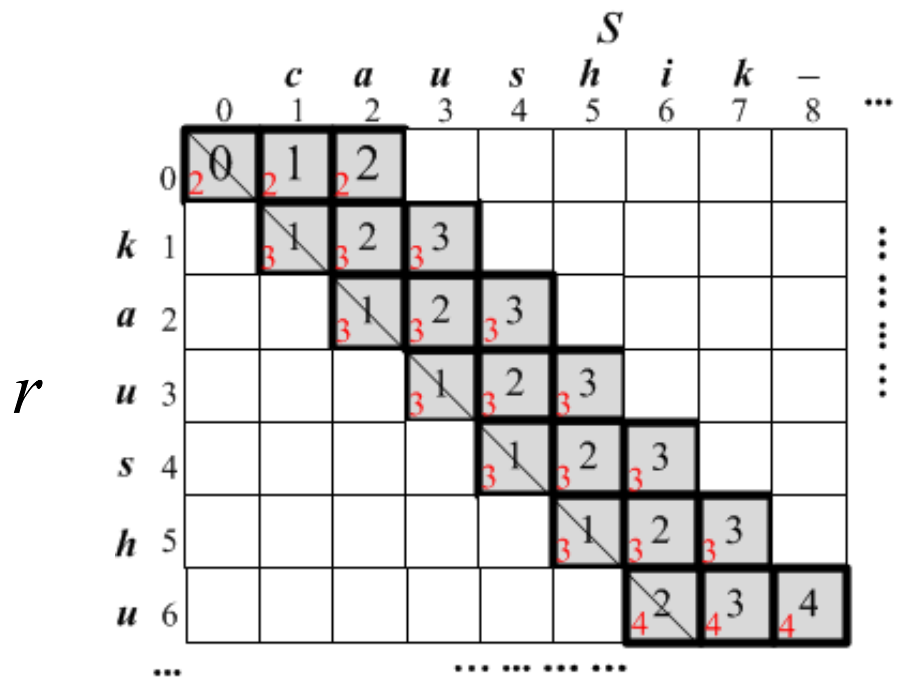
all derived cells must have values larger than τ.

Length-aware Verification



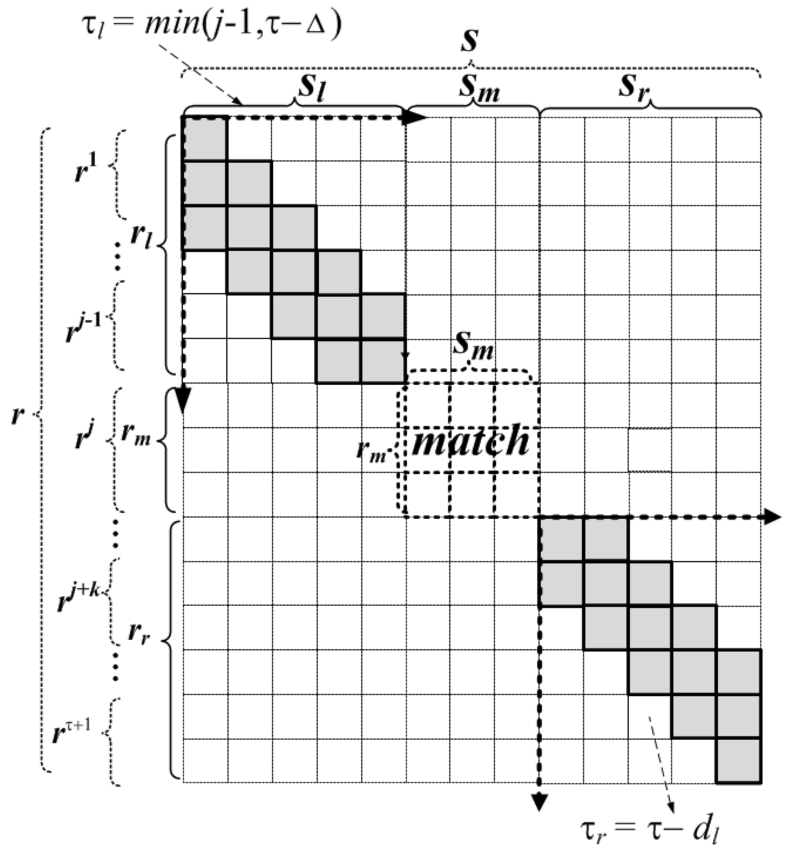
total length difference is 4 > τ. Thus no need to calculate M[2][5].

Length-aware Verification



only need to calculate a band of width τ

Extension-base Verification



Extension-base Verification

- We can verify a candidate pair using tighter thresholds:
  - For the left parts we can set $\tau_l = i - 1$.
  - For the right parts we can set $\tau_r = \tau + 1 - i$.

$$\textit{band widths are } \tau_l < \tau \textit{ and } \tau_r < \tau$$

Takeaways

(1) The partitioning filter for Edit Distance

(2) The multi-match-aware method

(3) The extension-based verification

References

- Pass-Join: A Partition based Method for Similarity Joins. G. Li, D. Deng, J. Wang, J. Feng. VLDB 2012.
- A Pivotal Prefix Based Filtering Algorithm for String Similarity Search. D. Deng, G. Li, J. Feng. SIGMOD 2014.
- Ed-Join: An Efficient Algorithm for Similarity Joins with Edit Distance Constraints. C. Xiao, W. Wang, X. Lin. VLDB 2008