Pass-Join: A Partition-based Method for Similarity Joins

Guoliang Li
Dong Deng
Jiannan Wang
Jianhua Feng

演讲人：余海洋
What to do?

Find all similar string pairs from the one set (two sets is the same) which pair’s edit distance is less than or equal to $\tau$. 
What is edit distance?

Edit distance is the minimum number of single-character edit operations (i.e., insertion, deletion, and substitution)
What is edit distance?

\[ S_1 = \{“abcdefg”}\]  

\[ S_2 = \{“abc\textcolor{red}{h}defg”}\] or \{“abcdef\textcolor{red}{h}”\}
What is it used for?

- Database
- IR (Information Retrieval)
- ...
How to do?

A naïve solution.

1. Find all string pairs from the two set.
2. Verify the string pairs if they are similar.

Time complexity is $O(O(1) \times O(2))$
How to do?

- **Pass-Join**

  ✓ Find all candidate string pairs from the two set.
  ✓ Reduce the time which is used to verify the string pairs if they are similar.
How to reduce string pairs?

- **Length Filter**

  - If one pair is similar and their length difference must be less than or equal to Tao. So we

  - Sort the strings

<table>
<thead>
<tr>
<th>Table 1: A set of strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Strings</td>
</tr>
<tr>
<td>Strings</td>
</tr>
<tr>
<td>avataresha</td>
</tr>
<tr>
<td>caushik chakrabar</td>
</tr>
<tr>
<td>kaushic chaduri</td>
</tr>
<tr>
<td>kaushik chakrab</td>
</tr>
<tr>
<td>kaushuk chadhui</td>
</tr>
<tr>
<td>vankatesh</td>
</tr>
<tr>
<td>(b) Sorted strings</td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>s1</td>
</tr>
<tr>
<td>s2</td>
</tr>
<tr>
<td>s3</td>
</tr>
<tr>
<td>s4</td>
</tr>
<tr>
<td>s5</td>
</tr>
<tr>
<td>s6</td>
</tr>
</tbody>
</table>
How to reduce string pairs?

Partition Scheme

✓ Given a string $s$, we partition it into $Tao+1$ disjoint segments.

✓ The shorter a segment of $r$ is, the higher probability the segment appears in other strings.

✓ Let $k = \left\lfloor \frac{|s|}{Tao+1} \right\rfloor \ast (Tao+1)$, and the first $Tao+1-k$ ones have length $\left\lfloor \frac{|s|}{Tao+1} \right\rfloor$, else ones have length $\left\lfloor \frac{|s|}{Tao+1} \right\rfloor + 1$

✓ For example, when $Tao = 3$, $S_1$’s ($S_1 = “vankatesh”$) partition is \{“va”, “nk”, “at”, “esh”\}
How to reduce string pairs?

Partition Scheme

Let $S_l$ denote the set of strings with length $l$ and $S^i_l$ denote the set of the $i$-th segment of strings in $S_l$. We build an inverted index for each $S^i_l$, denoted by $L^i_l$. Given an $i$-th segment $w$, let $L^i(w)$ denote the inverted list of segment $w$, i.e., the set of strings whose $i$-th segments are $w$. 
How to reduce string pairs?

 Partition-based Framework

\textbf{Algorithm 1: Pass-Join (S, \tau)}

\begin{itemize}
  \item \textbf{Input:} S: A collection of strings
  \item \tau: A given edit-distance threshold
  \item \textbf{Output:} A = \{ (s \in S, r \in S) | ED(s, r) \leq \tau \}
\end{itemize}

1 \textbf{begin}
2 \hspace{1em} Sort S first by string length and second in alphabetical order;
3 \hspace{1em} \textbf{for} s \in S \textbf{do}
4 \hspace{2em} \textbf{for} L_i^l (|s| - \tau \leq l \leq |s|, 1 \leq i \leq \tau + 1) \textbf{do}
5 \hspace{3em} W(s, L_i^l) = \text{SubstringSelection}(s, L_i^l);
6 \hspace{3em} \textbf{for} w \in W(s, L_i^l) \textbf{do}
7 \hspace{4em} \textbf{if} w \text{ is in } L_i^l \text{ then}
8 \hspace{5em} \text{Verification}(s, L_i^l(w), \tau);
9 \hspace{1em} \text{Partition } s \text{ and add its segments into } L_i^{|s|};
10 \textbf{end}

\textbf{Function SubstringSelection}(s, L_i^l)

\begin{itemize}
  \item \textbf{Input:} s: A string; L_i^l: Inverted index
  \item \textbf{Output:} W(s, L_i^l): Selected substrings
\end{itemize}

1 \textbf{begin}
2 \hspace{1em} W(s, L_i^l) = \{ w | w \text{ is a substring of } s \};
3 \textbf{end}

\textbf{Function Verification}(s, L_i^l(w), \tau)

\begin{itemize}
  \item \textbf{Input:} s: A string; L_i^l(w): Inverted list; \tau: Threshold
  \item \textbf{Output:} A = \{ (s \in S, r \in S) | ED(s, r) \leq \tau \}
\end{itemize}

1 \textbf{begin}
2 \hspace{1em} \textbf{for} r \in L_i^l(w) \textbf{do}
3 \hspace{2em} \textbf{if} ED(s, r) \leq \tau \text{ then } A \leftarrow (s, r);
4 \textbf{end}

\textbf{Figure 3:} Pass-Join algorithm
How to reduce string pairs?

Partition-based Framework

Figure 1: An example of our partition-based framework
Partition-based Framework

- Improving Substring Selection
  - Length-based Method
  - Shift-based Method
  - Position-aware Substring Selection
  - Muti-match-aware Substring Selection
Partition-based Framework

- Length-based Method
Partition-based Framework

- **Length-based Method**

**Length-based Method**: As segments in $\mathcal{L}_i$ have the same length, denoted by $l_i$, the length-based method selects all substrings of $s$ with length $l_i$, denoted by $\mathcal{W}_\ell(s, \mathcal{L}_i^i)$. Let $\mathcal{W}_\ell(s, l) = \bigcup_{i=1}^{\tau+1} \mathcal{W}_\ell(s, \mathcal{L}_i^i)$. The length-based method satisfies completeness, as it selects all substrings with length $l_i$. The size of $\mathcal{W}_\ell(s, \mathcal{L}_i^i)$ is $|\mathcal{W}_\ell(s, \mathcal{L}_i^i)| = |s| - l_i + 1$, and the number of selected substrings is $|\mathcal{W}_\ell(s, l)| = (\tau + 1)(|s| + 1) - l$. 

2012/6/15  http://datamining.xmu.edu.cn  15/11
Partition-based Framework

- Shift-based Method

![Diagram](image-url)
Partition-based Framework

Shift-based Method: However the length-based method does not consider the positions of segments. To address this problem, Wang et al. [22] proposed a shift-based method to address the entity identification problem. We can extend their method to support our problem as follows. As segments in $\mathcal{L}_i$ have the same length, they have the same start position, denoted by $p_i$, where $p_1 = 1$ and $p_i = p_1 + \sum_{k=1}^{i-1} l_k$ for $i > 1$. The shift-based method selects $s$'s substrings with start positions in $[p_i - \tau, p_i + \tau]$ and with length $l_i$, denoted by $\mathcal{W}_f(s, \mathcal{L}_i)$. Let $\mathcal{W}_f(s, l) = \bigcup_{i=1}^{\tau+1} \mathcal{W}_f(s, \mathcal{L}_i)$. The size of $\mathcal{W}_f(s, \mathcal{L}_i)$ is $|\mathcal{W}_f(s, \mathcal{L}_i)| = 2\tau + 1$. The number of selected substrings is $|\mathcal{W}_f(s, l)| = (\tau + 1)(2\tau + 1)$.
Partition-based Framework

➢ Position-aware Substring Selection

\[
(a) \text{Minimal Position } p_{min} = \max(1, \ p_i - \left\lfloor \frac{\tau - \Delta}{2} \right\rfloor)
\]

\[
\Delta_l \leq \left\lfloor \frac{\tau - \Delta}{2} \right\rfloor \text{ as } \tau \geq d_l + d_r \geq \Delta_l + (\Delta_l + \Delta)
\]
Partition-based Framework

- Position-aware Substring Selection

**Minimal Start Position:** Suppose the start position of $s_m$, denoted by $p$, is not larger than $p_i$. Let $\Delta = |s| - |r|$ and $\Delta_l = p_i - p$. We have $d_l = ED(r_l, s_l) \geq \Delta_l$ and $d_r = ED(r_r, s_r) \geq \Delta_l + \Delta$, as illustrated in Figure 4(a). If $s$ is similar to $r$ (or any string in $L_i(r_m)$), we have

$$\Delta_l + (\Delta_l + \Delta) \leq d_l + d_r \leq \tau.$$  

That is $\Delta_l \leq \lfloor \frac{\tau - \Delta}{2} \rfloor$ and $p = p_i - \Delta_l \geq p_i - \lfloor \frac{\tau - \Delta}{2} \rfloor$. Thus $p_{min} \geq p_i - \lfloor \frac{\tau - \Delta}{2} \rfloor$. As $p_{min} \geq 1$, $p_{min} = \max(1, p_i - \lfloor \frac{\tau - \Delta}{2} \rfloor)$. 

\[(a)\text{ Minimal Position } p_{min} = \max(1, p_i - \lfloor \frac{\tau - \Delta}{2} \rfloor) \]

\[\Delta_l \leq \lfloor \frac{\tau - \Delta}{2} \rfloor \text{ as } \tau \geq d_l + d_r \geq \Delta_l + (\Delta_l + \Delta)\]
Partition-based Framework

- Position-aware Substring Selection

\[ p_{\text{max}} = \min(|s| - l_i + 1, p_i + \frac{\tau + \Delta}{2}) \]
\[ \Delta_r \leq \left\lfloor \frac{\tau + \Delta}{2} \right\rfloor \text{ as } \tau \geq d_l + d_r \geq \Delta_r + (\Delta_r - \Delta) \]

Let \( S_i^1 \), \( S_i^2 \), ..., \( S_i^\tau \), \( S_i^{\tau+1} \) represent segments of the partition-based framework.
Partition-based Framework

Position-aware Substring Selection

Maximal Start Position: Suppose the start position of $s_m$, $p_i$, is larger than $p_i$. Let $\triangle = |s| - |r|$ and $\triangle_r = p - p_i$. We have $d_l = ED(r_l, s_l) \geq \triangle_r$ and $d_r = ED(r_r, s_r) \geq |\triangle_r - \triangle|$ as illustrated in Figure 4(b). If $\triangle_r \leq \triangle$, $d_r \geq \triangle - \triangle_r$. Thus $\triangle = \triangle_r + (\triangle - \triangle_r) \leq d_l + d_r \leq \tau$, and in this case, the maximal value of $\triangle_r$ is $\triangle$; otherwise if $\triangle_r > \triangle$, $d_r \geq \triangle_r - \triangle$. If $s$ is similar to $r$ (or any string in $L_i^t(r_m)$), we have

$$\triangle_r + (\triangle_r - \triangle) \leq d_l + d_r \leq \tau.$$  

That is $\triangle_r \leq \left\lceil \frac{\tau + \triangle}{2} \right\rceil$, and $p = p_i + \triangle_r \leq p_i + \left\lceil \frac{\tau + \triangle}{2} \right\rceil$. Thus $p_{\text{max}} \leq p_i + \left\lceil \frac{\tau + \triangle}{2} \right\rceil$. As the segment length is $l$, based on the boundary, we have $p_{\text{max}} \leq |s| - l_i + 1$. Thus $p_{\text{max}} = \min(|s| - l_i + 1, p_i + \left\lceil \frac{\tau + \triangle}{2} \right\rceil)$.

Maximal Position $p_{\text{max}} = \min(|s| - l_i + 1, p_i + \left\lceil \frac{\tau + \triangle}{2} \right\rceil)$

$\triangle_r \leq \left\lceil \frac{\tau + \triangle}{2} \right\rceil$ as $\tau \geq d_l + d_r \geq \triangle - \Delta_r$.
Partition-based Framework

- Muti-match-aware Substring Selection
  
  ✓ If we know that s must have a substring after Sm which will match one segment, we can discard substring Sm.
Partition-based Framework

- Muti-match-aware Substring Selection

\[
\overline{l_i} = \max(1, p_i - (i - 1)) \quad T_i = \min(|s| - l_i + 1, p_i + (i - 1))
\]
Partition-based Framework

Muti-match-aware Substring Selection

Suppose $s$ has a substring $s_m$ with start position $p$ matching a segment $r_m \in \mathcal{L}_i$. We still consider the three parts of the two strings: $s_l, s_m, s_r$ and $r_l, r_m, r_r$ as illustrated in Figure 5. Let $\Delta_l = |p_i - p|$. $d_l = ED(r_l, s_l) \geq \Delta_l$. As there are $i - 1$ segments in $s_l$, if each segment only has 1 error when transforming $r_l$ to $s_l$, we have $\Delta_l \leq i - 1$. If $\Delta_l \geq i$, $d_l = ED(r_l, s_l) \geq \Delta_l \geq i$, $d_r = ED(r_r, s_r) \leq \tau - d_l \leq \tau - i$ (if $s$ is similar to $r$). As $r_r$ contains $\tau + 1 - i$ segments, $s_r$ must contain a substring matching a segment in $r_r$ based on the pigeon-hole principle, which can be proved similar to Lemma 1. In this way, we can discard $s_m$, since for any string $r \in \mathcal{L}_i(r_m)$, $s$ must have a substring that matches a segment in the right part $r_r$, and thus we can identify strings similar to $s$ using the next matching segment. In summary, if $\Delta_l = |p - p_i| \leq i - 1$, we keep the substring with start position $p$ for $\mathcal{L}_i$. That is the minimal start position is $\bot_i = \max(1, p_i - (i - 1))$ and the maximal start position is $\top_i = \min(|s| - l_i + 1, p_i + (i - 1))$. 

(a) Multi-match from the left-side perspective

\[ \bot_i = \max(1, p_i - (i - 1)) \quad \top_i = \min(|s| - l_i + 1, p_i + (i - 1)) \]
Partition-based Framework

- Muti-match-aware Substring Selection

\[ \downarrow_{i}^{r} = \max(1, p_{i} + \Delta - (\tau + 1 - i)) \]

\[ \Uparrow_{i}^{r} = \min(|s| - l_{i} + 1, p_{i} + \Delta + (\tau + 1 - i)) \]

\[ \Delta_{r} \leq \tau + 1 - i \text{ as there are } \tau + 1 - i \text{ segments in } r_{r} \]
Partition-based Framework

- Muti-match-aware Substring Selection

The above observation is made from the left-side perspective. Similarly, we can use the same idea from the right-side perspective. As there are $\tau + 1 - i$ segments on the right part $r_r$, there are at most $\tau + 1 - i$ edit errors on $r_r$. If we transform $r$ to $s$ from the right-side perspective, position $p_i$ on $r$ should be aligned with position $p_i + \Delta$ on $s$ as shown in Figure 5(b). Suppose the position $p$ on $s$ matching position $p_i$ on $r$. Let $\Delta_r = |p - (p_i + \Delta)|$. We have $d_r = ED(s_r, r_r) \geq \Delta_r$. As there are $\tau + 1 - i$ segments on the right part $r_r$, we have $\Delta_r \leq \tau + 1 - i$. Thus the minimal start position for $L_i^r$ is $\downarrow_i^r = \max(1, p_i + \Delta - (\tau + 1 - i))$ and the maximal start position is $\Uparrow_i^r = \min(|s| - l_i + 1, p_i + \Delta + (\tau + 1 - i))$. 

(b) Muti-match from the right-side perspective

$$\downarrow_i^r = \max(1, p_i + \Delta - (\tau + 1 - i)) \quad \Uparrow_i^r = \min(|s| - l_i + 1, p_i + \Delta + (\tau + 1 - i))$$
Partition-based Framework

- Muti-match-aware Substring Selection

More interestingly, we can use the two techniques simultaneously. That is for $\mathcal{L}_i$, we only select the substrings with the start positions between $\underline{i} = \max(\underline{i}^l, \underline{i}^r)$ and $\overline{i} = \min(\overline{i}^l, \overline{i}^r)$ and with length $l_i$, denoted by $\mathcal{W}_m(s, \mathcal{L}_i)$. Let $\mathcal{W}_m(s, l) = \bigcup_{i=1}^{\tau+1} \mathcal{W}_m(s, \mathcal{L}_i)$. The number of selected substrings is $|\mathcal{W}_m(s, l)| = \left\lfloor \frac{\tau^2 - \Delta^2}{2} \right\rfloor + \tau + 1$ as stated in Lemma 2.
Partition-based Framework

- Muti-match-aware Substring Selection

```
Algorithm 2: SUBSTRINGSELECTION(s, \( \mathbb{L}_i^i \))

Input: s: A string; \( \mathbb{L}_i^i \): Inverted index
Output: \( \mathcal{W}(s, \mathbb{L}_i^i) \): Selected substrings
1 begin
2 for \( p \in [\bot_i, \top_i] \) do
3 Add the substring of \( s \) with start position \( p \) and
4 with length \( l_i \) (\( s[p, l_i] \)) into \( \mathcal{W}(s, \mathbb{L}_i^i) \);
4 end

Figure 6: SubstringSelection algorithm
```
Partition-based Framework

⇒ Improving The Verification

✓ Length-aware Verification
✓ Extension-based Verification
✓ Sharing Computations
Partition-based Framework

➢ Length-aware Verification

\[
M(i, j) = \min(M(i-1, j)+1, M(i, j-1)+1, M(i-1, j-1)+\delta)
\]

\[
\tau=3 \quad \Delta=|x|-|r|=2 \quad \left\lfloor \frac{\tau-\Delta}{2} \right\rfloor = 0 \quad \left\lceil \frac{\tau+\Delta}{2} \right\rceil = 2
\]

Figure 7: An example for verification
Partition-based Framework

❖ Length-aware Verification

The first $i$ characters of $r$ to the first $j$ characters of $s$ with $d_1$ edit operations and then transforming the other characters in $r$ to the other characters in $s$ with $d_2$ edit operations. Based on length difference, we have $d_1 \geq |i - j|$ and $d_2 \geq |(|s| - j) - (|r| - i)| = |\triangle + (i - j)|$. If $d_1 + d_2 > \tau$, we do not need to compute $M(i, j)$, since the distance of any transformation including $M(i, j)$ is larger than $\tau$. To check whether $d_1 + d_2 > \tau$, we consider the following cases.

1. If $i \geq j$, we have $d_1 + d_2 \geq i - j + \triangle + i - j$. If $i - j + \triangle + i - j > \tau$, that is $j < i - \frac{\tau - \Delta}{2}$, we do not need to compute $M(i, j)$. In other words, we only need to compute $M(i, j)$ with $j \geq i - \frac{\tau - \Delta}{2}$.

2. If $i < j$, $d_1 = j - i$. If $j - i \leq \triangle$, $d_1 + d_2 \geq j - i + \triangle - (j - i) = \triangle$. As $\triangle \leq \tau$, there is no position constraint. We need to compute $M(i, j)$; otherwise if $j - i > \triangle$, we have $d_1 + d_2 \geq j - i + j - i - \triangle$. If $j - i + j - i - \triangle > \tau$, that is $j > i + \frac{\tau + \Delta}{2}$, we do not need to compute $M(i, j)$. In other words, we only need to compute $M(i, j)$ with $j \leq i + \frac{\tau + \Delta}{2}$.

Figure 7: An example for verification
Partition-based Framework

- Length-aware Verification

- Prefix pruning

$$M(i, j) = \min (M(i-1, j)+1, M(i, j-1)+1, M(i-1, j-1)+\delta)$$

**Early Termination:** We can further improve the performance by doing an early termination. Consider the values in row $M(i, \ast)$. A straightforward early-termination method is to check each value in $M(i, \ast)$, and if each value is larger than $\tau$, we can do an early termination. This is because the values in the following rows $M(k > i, \ast)$ must be larger than $\tau$ based on the dynamic-programming algorithm. This pruning technique is called *prefix pruning*. For example in
Partition-based Framework

- Extension-based Verification

Figure 9: Extension-based verification
Partition-based Framework

Extension-based Verification

For example, if we want to verify $s_5 = \text{“kaushuk chadhui”}$ and $s_6 = \text{“caushik chakrabar”}$. $s_5$ and $s_6$ share a segment “cha”. We have $s_{5_l} = \text{“kaushuk”}$ and $s_{6_l} = \text{“caushik”}$, and $s_{5_r} = \text{“dhui”}$ and $s_{6_r} = \text{“krabar”}$. Suppose $\tau = 3$. As $|s_{5_r}|-|s_{6_r}| = 2$, $\tau_l = \tau - 2 = 1$. We only need to verify whether the edit distance between $s_{5_l}$ and $s_{6_l}$ is not larger than $\tau_l = 1$. After we have computed $M(6, *)$, we can do an early termination as each value in $E(6, *)$ is larger than 1, as shown in Figure 7. Note that as $\tau_l = 1$ and $|s_{5_l}|-|s_{6_l}| = 0$, $\bot_i = \tau_i = 0$. Thus we only need to compute $M(i, i)$.

We discuss how to deduce a tighter bound for $\tau_l$ and $\tau_r$. Consider the $i$-th segment. If $d_l \geq i$, we can terminate the verification based on the multi-match-aware method. Thus we have $\tau_l = i - 1$. Combining with the above pruning condition, we have $\tau_l = \min(\tau - |r_r| - |s_r|, i - 1)$. As $|r_r| - |s_r| = |(|r| - p_t - l_r) - (|s| - p - l_s)| = |p - p_t - \triangle| \leq \tau + 1 - i$ (based on the multi-match-aware method), $\tau - |r_r| - |s_r| \geq i - 1$. We set $\tau_l = i - 1$. Similarly we have $\tau_r = \min(\tau - d_l, \tau + 1 - i)$. As $d_l \leq \tau_l \leq i - 1$, $\tau - d_l \geq \tau - (i - 1)$. Thus we set $\tau_r = \tau + 1 - i$.

Figure 9: Extension-based verification
Partition-based Framework

Sharing Computations

programming algorithm. We store the matrix for $r_{1_i}$ and $s_l$. For the next string $r_2$ with left part $r_{2_i}$, we use the stored matrix to compute the edit distance between $r_{2_i}$ and $s_l$. We first compute the longest common prefix between $r_{2_i}$ and $r_{1_i}$, denoted by $c$. When computing the edit distance between $s_l$ and $r_{2_i}$, we use the stored matrix on $s_l$ and $c$ which has already been computed for $s_l$ and $r_{1_i}$. Then for the characters after $c$ in $r_{2_i}$, we continue the computation using the kept matrix. Thus we avoid many unnecessary computations. Notice that we do not need to maintain multiple matrixes and only keep a single matrix for the current string. We use the same idea on the right parts($s_r$, $r_r$).
Partition-based Framework

 Verification Algorithm

**Algorithm 3: VERIFICATION(s, \( L^i_j(w) \), \( \tau \))**

Input: \( s \): A string; \( L^i_j(w) \): Inverted list; \( \tau \): Threshold  
Output: \( R = \{(s \in S, r \in S) | ED(s, r) \leq \tau\}\)

1. begin
2.  \( \tau_l = i - 1; \)
3.  \( \tau_r = \tau + 1 - i; \)
4.  for \( r \in L^i_j(w) \) do
5.    \( d_l = \text{VERIFYSTRINGPAIR}(s_l, r_l, \tau_l); \)
6.    if \( d_l \leq \tau_l \) then
7.      \( d_r = \text{VERIFYSTRINGPAIR}(s_r, r_r, \tau_r); \)
8.      if \( d_r \leq \tau_r \) then \( R \leftarrow (r, s); \)
9.  end

**Function VERIFYSTRINGPAIR(s, r, \( \tau' \))**

Input: \( s \): A string; \( r \): A string; \( \tau' \): A threshold  
Output: \( d = \min(\tau' + 1, ED(s, r))\)

1. begin
2.  Using the length-aware verification with the threshold \( \tau' \)  
   and sharing the computations on common prefixes;
3.  if Early Termination then \( d = \tau' + 1; \)
4.  else \( d = ED(s, r); \)
5.  end

Figure 10: Verification algorithm
Multi-threads Improving
Multi-threads Improving

Algorithm 1: Pass-Join ($S, \tau$)

Input: $S$: A collection of strings

$\tau$: A given edit-distance threshold

Output: $A = \{(s \in S, r \in S) \mid \text{ED}(s, r) \leq \tau\}$

1 begin
2   Sort $S$ first by string length and second in alphabetical order;
3   for $s \in S$ do
4       for $L^i$ ($|s| - \tau \leq i \leq |s|, 1 \leq \tau + 1$) do
5           \begin{align*}
6               \mathcal{W}(s, L^i) &= \text{SUBSTRINGSELECTION}(s, L^i); \\
7               \text{for } w \in \mathcal{W}(s, L^i) \text{ do} \\
8                   \text{if } w \text{ is in } L^i \text{ then} \\
9                     \text{VERIFICATION}(s, L^i(w), \tau);
10           \end{align*}
11       Partition $s$ and add its segments into $L^i_s$;
12   end
13 end

Function SUBSTRINGSELECTION ($s, L^i$)

Input: $s$: A string; $L^i$: Inverted index

Output: $\mathcal{W}(s, L^i)$: Selected substrings

1 begin
2   $\mathcal{W}(s, L^i) = \{w \mid w \text{ is a substring of } s\}$;
3 end

Function VERIFICATION ($s, L^i(w), \tau$)

Input: $s$: A string; $L^i(w)$: Inverted list; $\tau$: Threshold

Output: $A = \{(s \in S, r \in S) \mid \text{ED}(s, r) \leq \tau\}$

1 begin
2   for $r \in L^i(w)$ do
3     if $\text{ED}(s, r) \leq \tau$ then $A \leftarrow (s, r)$;
4 end

Figure 3: Pass-Join algorithm
Trie Improving

Candidates: \(<3, 5>\); \(<4, 5>\)
Answer: \(\emptyset\)
Trie Improving

Trie-Join: Efficient Trie-based String Similarity Joins with Edit-Distance Constraints

Jiannan Wang       Jianhua Feng       Guoliang Li
Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 10084, China
wjn08@mails.thu.edu.cn; fengjh@tsinghua.edu.cn; liguoliang@tsinghua.edu.cn
Thanks for patience

➢ Email: yhycai@gmail.com