

A Belief Framework for Similarity Evaluation of Textual or Structured Data

Sergej Znamenskij
E-mail: svz@latex.pereslavl.ru



Ailamazyan Program Systems Institute of RAS

SiSAp 2015, October 12–14

Glasgow, 2015



- 1 Edit distance problem
 - Traditional definition shortcomings
 - Align valuable substrings
 - The string similarity problem
- 2 Believed similarity and framework
 - Belief function
 - Believed similarity
 - Framework idea
- 3 The ideas of algorithm
 - Start with longer alignments
 - General scheme
 - Fast search of long aligned substrings

The example of non-expressive LCS

Find the edit distance from (A) to (B) :

(A) preference being reversed

(B) be a reversed preference

(A) ~~preference being reversed~~

(B) be a reversed preference

This edit distance is 15 deletions + 14 insertions = 29.

(A) ~~preference being reversed~~

(B) be a reversed preference.

LCS (Longest Common Subsequence)-optimal edit distance

$d(A, B) = 14 + 13 = 27$.

Which edition is more simple and natural?

The example of non-expressive LCS

Find the edit distance from (A) to (B) :

(A) preference being reversed

(B) be a reversed preference

(A) **preference** ~~being reversed~~

(B) be a reversed **preference**

This edit distance is 15 deletions + 14 insertions = 29.

(A) ~~preference being reversed~~

(B) be a reversed preference.

LCS (Longest Common Subsequence)-optimal edit distance

$d(A, B) = 14 + 13 = 27$.

Which edition is more simple and natural?

The example of non-expressive LCS

Find the edit distance from (A) to (B) :

(A) preference being reversed

(B) be a reversed preference

(A) ~~preference being reversed~~

(B) be a reversed preference

This edit distance is 15 deletions + 14 insertions = 29.

(A) ~~preference being reversed~~

(B) be a reversed preference.

LCS (Longest Common Subsequence)-optimal edit distance

$d(A, B) = 14 + 13 = 27$.

Which edition is more simple and natural?

The example of non-expressive LCS

Find the edit distance from (A) to (B) :

(A) preference being reversed

(B) be a reversed preference

(A) ~~preference being reversed~~

(B) be a reversed preference

This edit distance is 15 deletions + 14 insertions = 29.

(A) ~~preference being reversed~~

(B) be a reversed preference.

LCS (Longest Common Subsequence)-optimal edit distance

$d(A, B) = 14 + 13 = 27$.

Which edition is more simple and natural?

The example of counterintuitive edit distance

Which string is more close to (B) , (A) or (C) ?

(A) preference being reversed

(B) be a reversed preference

(C) a pure repaired refresher

(B) ~~be a reversed preference~~

(C) a pure repaired refresher

$$d(B, C) = 9 + 10 = 19 < d(A, B) = 27$$

Informative substrings reversed and preference has been lost.

The example of counterintuitive edit distance

Which string is more close to (B) , (A) or (C) ?

(A) preference being reversed

(B) be a reversed preference

(C) a pure repaired refresher

(B) ~~be a reversed preference~~

(C) a pure repaired refresher

$$d(B, C) = 9 + 10 = 19 < d(A, B) = 27$$

Informative substrings reversed and preference has been lost.

The example of counterintuitive edit distance

Which string is more close to (B) , (A) or (C) ?

(A) preference being reversed

(B) be a reversed preference

(C) a pure repaired refresher

(B) ~~be a reversed preference~~

(C) a pure repaired refresher

$$d(B, C) = 9 + 10 = 19 < d(A, B) = 27$$

Informative substrings reversed and preference has been lost.

Valuable substrings to align

$X \subset A$ is *valuable* if it is more informative than its parts:

$$X = Y \cup Z, \quad w(X) > w(Y) + w(Z).$$

where $w(X)$ is the weight (importance) to get aligned.

Applications depend on valuable substrings rather than other subsets:

- fuzzy matching of records in textual database — on names integrity;
- versioning support for documents or source code — on integrity of words, sentences, blocks, etc.;
- comparison of software logs or system call tracing logs — on hiding of long identical fragments;
- genetic data analysis — on identical genes or other substantial entities selection

Valuable substrings to align

$X \subset A$ is *valuable* if it is more informative than its parts:

$$X = Y \cup Z, \quad w(X) > w(Y) + w(Z).$$

where $w(X)$ is the weight (importance) to get aligned.

Applications depend on valuable substrings rather than other subsets:

- fuzzy matching of records in textual database — on names integrity;
- versioning support for documents or source code — on integrity of words, sentences, blocks, etc.;
- comparison of software logs or system call tracing logs — on hiding of long identical fragments;
- genetic data analysis — on identical genes or other substantial entities selection

Valuable substrings to align

$X \subset A$ is *valuable* if it is more informative than its parts:

$$X = Y \cup Z, \quad w(X) > w(Y) + w(Z).$$

where $w(X)$ is the weight (importance) to get aligned.

Applications depend on valuable **substrings** rather than other subsets:

- fuzzy matching of records in textual database — on names integrity;
- versioning support for documents or source code — on integrity of words, sentences, blocks, etc.;
- comparison of software logs or system call tracing logs — on hiding of long identical fragments;
- genetic data analysis — on identical genes or other substantial entities selection

Valuable substrings to align

$X \subset A$ is *valuable* if it is more informative than its parts:

$$X = Y \cup Z, \quad w(X) > w(Y) + w(Z).$$

where $w(X)$ is the weight (importance) to get aligned.

Applications depend on valuable **substrings** rather than other subsets:

- fuzzy matching of records in textual database — on names integrity;
- versioning support for documents or source code — on integrity of words, sentences, blocks, etc.;
- comparison of software logs or system call tracing logs — on hiding of long identical fragments;
- genetic data analysis — on identical genes or other substantial entities selection

Valuable substrings to align

$X \subset A$ is *valuable* if it is more informative than its parts:

$$X = Y \cup Z, \quad w(X) > w(Y) + w(Z).$$

where $w(X)$ is the weight (importance) to get aligned.

Applications depend on valuable **substrings** rather than other subsets:

- fuzzy matching of records in textual database — on names integrity;
- versioning support for documents or source code — on integrity of words, sentences, blocks, etc.;
- comparison of software logs or system call tracing logs — on hiding of long identical fragments;
- genetic data analysis — on identical genes or other substantial entities selection

Valuable substrings to align

$X \subset A$ is *valuable* if it is more informative than its parts:

$$X = Y \cup Z, \quad w(X) > w(Y) + w(Z).$$

where $w(X)$ is the weight (importance) to get aligned.

Applications depend on valuable **substrings** rather than other subsets:

- fuzzy matching of records in textual database — on names integrity;
- versioning support for documents or source code — on integrity of words, sentences, blocks, etc.;
- comparison of software logs or system call tracing logs — on hiding of long identical fragments;
- genetic data analysis — on identical genes or other substantial entities selection


If valuable substrings aren't known

Suppose any substring to be counted with the same expected value.

Definition of NCS (number of common substrings).

Select common subsequence to maximise the number w_{NCS} of all the common to a and b substrings in it.

The distance $d_{\text{NCS}}(a, b)$ is a number of all other substrings in a and b .

A pointer 

locates a **substring** inside the string below


If valuable substrings aren't known

Suppose any substring to be counted with the same expected value.

Definition of NCS (number of common substrings).

Select common subsequence to maximise the number w_{NCS} of all the common to a and b substrings in it.

The distance $d_{\text{NCS}}(a, b)$ is a number of all other substrings in a and b .

A pointer

 locates a **substring** inside the string below


If valuable substrings aren't known

Suppose any substring to be counted with the same expected value.

Definition of NCS (number of common substrings).

Select common subsequence to maximise the number w_{NCS} of all the common to a and b substrings in it.

The distance $d_{\text{NCS}}(a, b)$ is a number of all other substrings in a and b .

A pointer  **u** **substring**

locates a **substring** inside the string below


If valuable substrings aren't known

Suppose any substring to be counted with the same expected value.

Definition of NCS (number of common substrings).

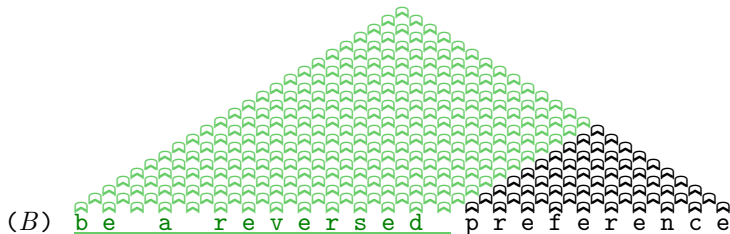
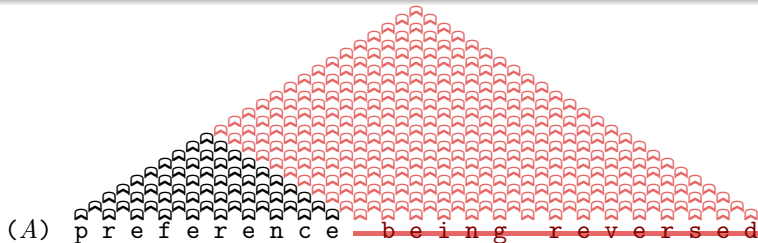
Select common subsequence to maximise the number w_{NCS} of all the common to a and b substrings in it.

The distance $d_{\text{NCS}}(a, b)$ is a number of all other substrings in a and b .

A pointer 

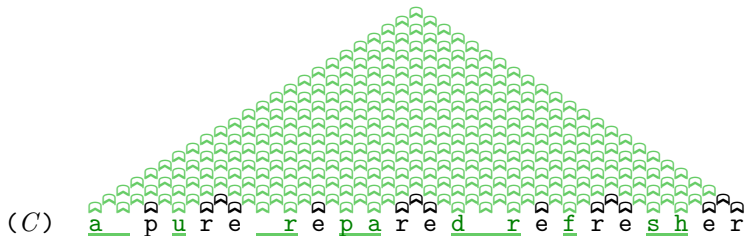
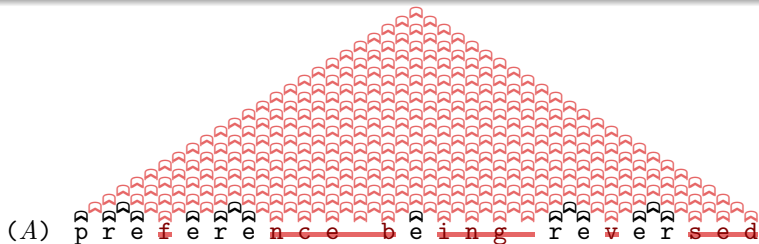
locates a **substring** inside the string below

Counting all the substrings: (A) vs (B)



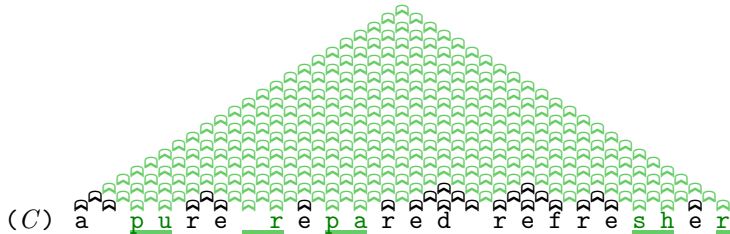
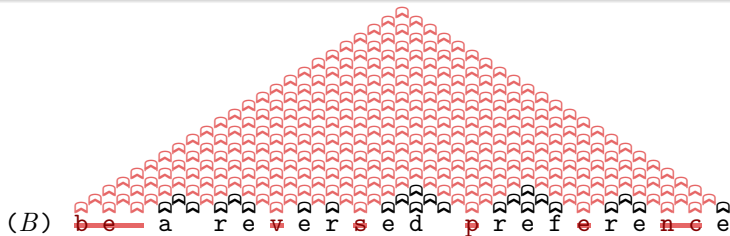
$$w_{\text{NCS}}(a, b) = \frac{10 \cdot 11}{2} = 55, \quad d_{\text{NCS}}(a, b) = \left(\frac{25 \cdot 26}{2} - 55\right) + \left(\frac{24 \cdot 25}{2} - 55\right) = 515$$

Counting all the substrings: (A) vs (C)



$$w_{\text{NCS}}(a, c) = 3 \cdot 1 + 4 \cdot 3 = 15, \quad d_{\text{NCS}}(a, c) = \left(\frac{25 \cdot 26}{2} - 15\right) + \left(\frac{24 \cdot 25}{2} - 15\right) = 595$$

Counting all the substrings: (B) vs (C)

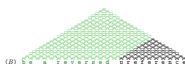
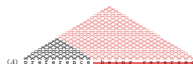


$$w_{\text{NCS}}(a, b) = 3 \cdot 1 + 3 \cdot 3 + 2 \cdot 6 = 24, \quad d_{\text{NCS}}(b, c) = 2 \cdot \left(\frac{24 \cdot 25}{2} - 24 \right) = 552$$

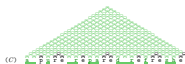
Counting all the substrings: the predictable results

Which string is more close to (B) , (A) or (C) ?

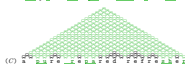
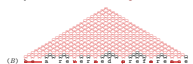
- (A) preference being reversed
- (B) be a reversed preference
- (C) extsfa pure repared refresher



$$d_{\text{NCS}}(a, b) = 515, w_{\text{NCS}}(a, b) = 55.$$



$$d_{\text{NCS}}(a, c) = 595, w_{\text{NCS}}(a, b) = 15.$$



$$d_{\text{NCS}}(b, c) = 552, w_{\text{NCS}}(a, b) = 24.$$

The greater similarity of (A) and (B) has been properly detected now.

The calculation of NCS

$A_l = \{a_1, \dots, a_l\}, B_m = \{b_1, \dots, b_m\}$ — input strings.

n — length of their common ending:

$$a_{l-k} = b_{m-k} \quad \forall k=0, \dots, n-1,$$

$$a_{l-n} \neq b_{m-n}.$$

The simple recurrent formula comes from the identification of aligned substring at the end:

$$w_{\text{NCS}}(A_l, B_m) = \max_{\substack{k=0, \dots, n \\ (i,j) \in \{(k-1,k), (k,k-1)\}}} \left(w_{\text{NCS}}(A_{l-i}, B_{m-j}) + \frac{k(k+1)}{2} \right)$$

It was used in a dynamic programming algorithm with the cubic complexity upper bound.

The calculation of NCS

$A_l = \{a_1, \dots, a_l\}, B_m = \{b_1, \dots, b_m\}$ — input strings.

n — length of their common ending:

$$a_{l-k} = b_{m-k} \quad \forall k=0, \dots, n-1,$$

$$a_{l-n} \neq b_{m-n}.$$

The simple recurrent formula comes from the identification of aligned substring at the end:

$$w_{\text{NCS}}(A_l, B_m) = \max_{\substack{k=0, \dots, n \\ (i,j) \in \{(k-1, k), (k, k-1)\}}} \left(w_{\text{NCS}}(A_{l-i}, B_{m-j}) + \frac{k(k+1)}{2} \right)$$

It was used in a dynamic programming algorithm with the cubic complexity upper bound.

NCS vs LCS for huge data

Challenge: Identify changes in a versions of huge data

Optimization required for large blocks being relocated and other changes being mostly clustered

- LCS lacks NCS ability to detect the large block relocations.
- Needs clarifying for NCS: Similarity of elements instead of equality, knowledge of valuable substrings use, random noise filtering.
- Alike to sequences, NCS may be considered also for graphs (count isomorphic subgraphs with similar data in vertexes or edges).
- Sequential algorithm for NCS is slightly more complex and sequential, novel approaches needed.

Valuable substrings search looks promising as a base of framework

NCS vs LCS for huge data

Challenge: Identify **changes** in a versions of **huge data**

Opimization required for large blocks being relocated and other changes being mostly clustered

- LCS lacks NCS ability to detect the large block relocations.
- Needs clarifying for NCS: Similarity of elements instead of equality, knowledge of valuable substrings use, random noise filtering.
- Alike to sequences, NCS may be considered also for graphs (count isomorphic subgraphs with similar data in vertexes or edges).
- Sequential algorithm for NCS is slightly more complex and sequential, novel approaches needed.

Valuable substrings search looks promising as a base of framework

NCS vs LCS for huge data

Challenge: Identify changes in a versions of huge data

Optimization required for large blocks being relocated and other changes being mostly clustered

- LCS lacks NCS ability to detect the large block relocations.
- Needs clarifying for NCS: Similarity of elements instead of equality, knowledge of valuable substrings use, random noise filtering.
- Alike to sequences, NCS may be considered also for graphs (count isomorphic subgraphs with similar data in vertexes or edges).
- Sequential algorithm for NCS is slightly more complex and sequential, novel approaches needed.

Valuable substrings search looks promising as a base of framework

NCS vs LCS for huge data

Challenge: Identify changes in a versions of huge data

Optimization required for large blocks being relocated and other changes being mostly clustered

- LCS lacks NCS ability to detect the large block relocations.
- Needs clarifying for NCS: Similarity of elements instead of equality, knowledge of valuable substrings use, random noise filtering.
- Alike to sequences, NCS may be considered also for graphs (count isomorphic subgraphs with similar data in vertexes or edges).
- Sequential algorithm for NCS is slightly more complex and sequential, novel approaches needed.

Valuable substrings search looks promising as a base of framework

NCS vs LCS for huge data

Challenge: Identify changes in a versions of huge data

Optimization required for large blocks being relocated and other changes being mostly clustered

- LCS lacks NCS ability to detect the large block relocations.
- Needs clarifying for NCS: Similarity of elements instead of equality, knowledge of valuable substrings use, random noise filtering.
- Alike to sequences, NCS may be considered also for graphs (count isomorphic subgraphs with similar data in vertexes or edges).
- Sequential algorithm for NCS is slightly more complex and sequential, novel approaches needed.

Valuable substrings search looks promising as a base of framework

The value of aligned part is a belief function

The superadditivity:

$$\forall_{i,j} A_i \cap A_j = \emptyset \implies w\left(\bigcup_i A_i\right) \geq \sum_i w(A_i)$$

Such w is **not a measure**. It is called a *belief function* (also a *fuzzy measure* or a *capacity*) Here

$$w(p) = \sum_{p' \subset p} v(p')$$

where v is the the Möbius transformation of w

$$v(p) = \sum_{p' \subset p} (-1)^{|p \setminus p'|} w(p')$$

which is non-negative for belief function w and $v(p) > 0$ only if p is valuable.

The value of aligned part is a belief function

The **additivity**:

$$\forall_{i,j} A_i \cap A_j = \emptyset \implies w\left(\bigcup_i A_i\right) = \sum_i w(A_i)$$

Such w is **not a measure**. It is called a *belief function* (also a *fuzzy measure* or a *capacity*) Here

$$w(p) = \sum_{p' \subset p} v(p'),$$

where v is the the Möbius transformation of w

$$v(p) = \sum_{p' \subset p} (-1)^{|p \setminus p'|} w(p')$$

which is non-negative for belief function w and $v(p) > 0$ only if p is valuable.

The value of aligned part is a belief function

The superadditivity:

$$\forall_{i,j} A_i \cap A_j = \emptyset \implies w\left(\bigcup_i A_i\right) \geq \sum_i w(A_i)$$

Such w is **not a measure**. It is called a *belief function* (also a *fuzzy measure* or a *capacity*) Here

$$w(p) = \sum_{p' \subset p} v(p')$$

where v is the the Möbius transformation of w

$$v(p) = \sum_{p' \subset p} (-1)^{|p \setminus p'|} w(p')$$

which is non-negative for belief function w and $v(p) > 0$ only if p is valuable.

The value of aligned part is a belief function

The superadditivity:

$$\forall_{i,j} A_i \cap A_j = \emptyset \implies w\left(\bigcup_i A_i\right) \geq \sum_i w(A_i)$$

Such w is **not a measure**. It is called a *belief function* (also a *fuzzy measure* or a *capacity*) Here

$$w(p) = \sum_{p' \subset p} v(p'),$$

where v is the the Möbius transformation of w

$$v(p) = \sum_{p' \subset p} (-1)^{|p \setminus p'|} w(p')$$

which is non-negative for belief function w and $v(p) > 0$ only if p is valuable.

Believed similarity definition

Definition

A **weight** w of **aligned part** p is the Choquet integral of given similarity s of aligned elements over w .

$$w(p) = (C) \int_p s \, dw \quad (1)$$

Here the Choquet integral can be calculated by formula

$$w(p) = (C) \int_p s \, dw = \sum_{p' \subset p} m(p') \min_{o \in p'} s(o)$$

where the sum is taken over all valuable alignments.

Definition

A **believed similarity** $S(X, Y) = \max_p w(p)$ where maximum is taken over all acceptable alignments.

The idea of framework

The framework is aimed to provide fast and fine results on practical data by combination of flexible problem statements with strong and adaptable general algorithm.

The frames are following:

- Acceptable alignment set for optimal selection that may include non-ordered alignments if useful;
- Objective belief function on alignments, that may be domain specific and adaptable;
- The elements similarity function or relation;
- The generic algorithm scheme;

Optimize long alignments before

Let there is no intact alignable substring of the length between l and $2l - 1$.

We call *skeleton* the union of longer than l strings in the alignment.

Remark

The shorter than l common strings does not affect the NCS-optimal skeleton identification:

Probably we can begin with the long common string identification

Algorithm scheme

The algorithm is the sequence of following tasks:

- Find all the longest alignments
(in parallel, with close to linear complexity),
- Refine and range them by value
(utilize knowledge of valuable strings),
- Use branch and bounds method to get a skeleton of optimal alignment,
- Refine and extend a skeleton by a local search.

Only the third (and usually easy) task can't be done in parallel.

Substring indicator

$X \in A$ abbreviates "X isomorphic to a substring in A"

Fixed vector of short test strings $\mathfrak{s} = (\mathfrak{s}_1, \dots, \mathfrak{s}_q)$ defines the *substring indicator* of a string X by the formula $\mathbf{i}_{\mathfrak{s}}(X) = \{i : \mathfrak{s}_i \in X\}$. The substring indicator $\mathbf{i}_{\mathfrak{s}}(X)$ is a subset of $\{1, \dots, q\}$ and is stored as bit array or a long integer.

left to right implication always, right to left probably

$$X_1 \in X_2 \iff \mathbf{i}_{\mathfrak{s}}(X_1) \subset \mathbf{i}_{\mathfrak{s}}(X_2)$$

The probability of right to left implication depends in data and test vector and varies until 0.999999 for 64-bit index.

Substring indicator

$X \in A$ abbreviates "X isomorphic to a substring in A"

Fixed vector of short test strings $\mathfrak{s} = (\mathfrak{s}_1, \dots, \mathfrak{s}_q)$ defines the *substring indicator* of a string X by the formula $\mathbf{i}_{\mathfrak{s}}(X) = \{i : \mathfrak{s}_i \in X\}$. The substring indicator $\mathbf{i}_{\mathfrak{s}}(X)$ is a subset of $\{1, \dots, q\}$ and is stored as bit array or a long integer.

left to right implication always, right to left probably

$$X_1 \in X_2 \iff \mathbf{i}_{\mathfrak{s}}(X_1) \subset \mathbf{i}_{\mathfrak{s}}(X_2)$$

The probability of right to left implication depends in data and test vector and varies until 0.999999 for 64-bit index.

Substring indicator

$X \in A$ abbreviates "X isomorphic to a substring in A"

Fixed vector of short test strings $\mathfrak{s} = (\mathfrak{s}_1, \dots, \mathfrak{s}_q)$ defines the *substring indicator* of a string X by the formula $\mathbf{i}_\mathfrak{s}(X) = \{i : \mathfrak{s}_i \in X\}$. The substring indicator $\mathbf{i}_\mathfrak{s}(X)$ is a subset of $\{1, \dots, q\}$ and is stored as bit array or a long integer.

left to right implication **always**, **right to left** probably

$$X_1 \in X_2 \iff \mathbf{i}_\mathfrak{s}(X_1) \subset \mathbf{i}_\mathfrak{s}(X_2)$$

The probability of right to left implication depends in data and test vector and varies until 0.999999 for 64-bit index.

Substring indicator

$X \in A$ abbreviates "X isomorphic to a substring in A"

Fixed vector of short test strings $\mathfrak{s} = (\mathfrak{s}_1, \dots, \mathfrak{s}_q)$ defines the *substring indicator* of a string X by the formula $\mathbf{i}_{\mathfrak{s}}(X) = \{i : \mathfrak{s}_i \in X\}$. The substring indicator $\mathbf{i}_{\mathfrak{s}}(X)$ is a subset of $\{1, \dots, q\}$ and is stored as bit array or a long integer.

left to right implication always, right to left probably



$$X_1 \in X_2 \iff \mathbf{i}_{\mathfrak{s}}(X_1) \subset \mathbf{i}_{\mathfrak{s}}(X_2)$$

The probability of right to left implication depends in data and test vector and varies until 0.999999 for 64-bit index.

Substring indicator

$$\mathbf{i}_s(X) = \{i : s_i \in X\}$$

Then $X_1 \in X_2 \iff \mathbf{i}_s(X_1) \subset \mathbf{i}_s(X_2)$ with a high probability.

- Cover $A = (a_1, \dots, a_n)$ by the overlapping strings
 $X_i = (a_{N(i-1)+1}, \dots, a_{N(i+1)})$:  A
- Split B to strings $Y_j = (b_{N(j-1)+1}, \dots, b_{Nj})$.  B
- Find spy indexes for each X_i, Y_j in parallel and linear time.
- Check the feasibility of conditions $\mathbf{i}_s(X_i) \subset \mathbf{i}_s(Y_j)$
 in nm/N^2 simple logical operation on long integers.

Now each string $Z \in A$ with $Z \in B$ of the length $l \in [(m-1)N, mN]$ is found as



$$Y_j \in X_i, \quad Y_{j+1} \in X_{i+1}, \quad \dots, \quad Y_{j+m} \in X_{i+m}$$

with started position being localised inside a segment
 $[N(i-1)+1, N(i+1)]$ in A and inside a segment $[N(j-1)+1, Nj]$ in B
 and Z can be found by usual technique.

Substring indicator

$$\mathbf{i}_s(X) = \{i : s_i \in X\}$$

Then $X_1 \in X_2 \iff \mathbf{i}_s(X_1) \subset \mathbf{i}_s(X_2)$ with a high probability.

- Cover $A = (a_1, \dots, a_n)$ by the overlapping strings
 $X_i = (a_{N(i-1)+1}, \dots, a_{N(i+1)})$: 
- Split B to strings $Y_j = (b_{N(j-1)+1}, \dots, b_{Nj})$. 
- Find spy indexes for each X_i, Y_j in parallel and linear time.
- Check the feasibility of conditions $\mathbf{i}_s(X_i) \subset \mathbf{i}_s(Y_j)$
 in nm/N^2 simple logical operation on long integers.

Now each string $Z \in A$ with $Z \in B$ of the length $l \in [(m-1)N, mN]$ is found as



$$Y_j \in X_i, \quad Y_{j+1} \in X_{i+1}, \quad \dots, \quad Y_{j+m} \in X_{i+m}$$

with started position being localised inside a segment
 $[N(i-1)+1, N(i+1)]$ in A and inside a segment $[N(j-1)+1, Nj]$ in B
 and Z can be found by usual technique.

Substring indicator

$$\mathbf{i}_s(X) = \{i : s_i \in X\}$$

Then $X_1 \in X_2 \iff \mathbf{i}_s(X_1) \subset \mathbf{i}_s(X_2)$ with a high probability.

- Cover $A = (a_1, \dots, a_n)$ by the overlapping strings
 $X_i = (a_{N(i-1)+1}, \dots, a_{N(i+1)})$: 
- Split B to strings $Y_j = (b_{N(j-1)+1}, \dots, b_{Nj})$. 
- Find spy indexes for each X_i, Y_j in parallel and linear time.
- Check the feasibility of conditions $\mathbf{i}_s(X_i) \subset \mathbf{i}_s(Y_j)$
 in nm/N^2 simple logical operation on long integers.

Now each string $Z \in A$ with $Z \in B$ of the length $l \in [(m-1)N, mN]$ is found as



$$Y_j \in X_i, \quad Y_{j+1} \in X_{i+1}, \quad \dots, \quad Y_{j+m} \in X_{i+m}$$

with started position being localised inside a segment
 $[N(i-1)+1, N(i+1)]$ in A and inside a segment $[N(j-1)+1, Nj]$ in B
 and Z can be found by usual technique.

Substring indicator

$$\mathbf{i}_s(X) = \{i : s_i \in X\}$$

Then $X_1 \in X_2 \iff \mathbf{i}_s(X_1) \subset \mathbf{i}_s(X_2)$ with a high probability.

- Cover $A = (a_1, \dots, a_n)$ by the overlapping strings
 $X_i = (a_{N(i-1)+1}, \dots, a_{N(i+1)})$: 
- Split B to strings $Y_j = (b_{N(j-1)+1}, \dots, b_{Nj})$. 
- Find spy indexes for each X_i, Y_j in parallel and linear time.
- Check the feasibility of conditions $\mathbf{i}_s(X_i) \subset \mathbf{i}_s(Y_j)$
 in nm/N^2 simple logical operation on long integers.

Now each string $Z \in A$ with $Z \in B$ of the length $l \in [(m-1)N, mN]$ is found as

$$Y_j \in X_i, \quad Y_{j+1} \in X_{i+1}, \quad \dots, \quad Y_{j+m} \in X_{i+m}$$


with started position being localised inside a segment $[N(i-1)+1, N(i+1)]$ in A and inside a segment $[N(j-1)+1, Nj]$ in B and Z can be found by usual technique.


Substring indicator

$$\mathbf{i}_s(X) = \{i : s_i \in X\}$$

Then $X_1 \in X_2 \iff \mathbf{i}_s(X_1) \subset \mathbf{i}_s(X_2)$ with a high probability.

- Cover $A = (a_1, \dots, a_n)$ by the overlapping strings

$X_i = (a_{N(i-1)+1}, \dots, a_{N(i+1)})$: 

- Split B to strings $Y_j = (b_{N(j-1)+1}, \dots, b_{Nj})$. 

- Find spy indexes for each X_i, Y_j in parallel and linear time.

- Check the feasibility of conditions $\mathbf{i}_s(X_i) \subset \mathbf{i}_s(Y_j)$
 in nm/N^2 simple logical operation on long integers.

Now each string $Z \in A$ with $Z \in B$ of the length $l \in [(m-1)N, mN]$ is found as



$$Y_j \in X_i, \quad Y_{j+1} \in X_{i+1}, \quad \dots, \quad Y_{j+m} \in X_{i+m}$$

with started position being localised inside a segment $[N(i-1)+1, N(i+1)]$ in A and inside a segment $[N(j-1)+1, Nj]$ in B and Z can be found by usual technique.

Substring indicator

$$\mathbf{i}_s(X) = \{i : s_i \in X\}$$

Then $X_1 \in X_2 \iff \mathbf{i}_s(X_1) \subset \mathbf{i}_s(X_2)$ with a high probability.

- Cover $A = (a_1, \dots, a_n)$ by the overlapping strings
 $X_i = (a_{N(i-1)+1}, \dots, a_{N(i+1)})$: 
- Split B to strings $Y_j = (b_{N(j-1)+1}, \dots, b_{Nj})$. 
- Find spy indexes for each X_i, Y_j in parallel and linear time.
- Check the feasibility of conditions $\mathbf{i}_s(X_i) \subset \mathbf{i}_s(Y_j)$
 in nm/N^2 simple logical operation on long integers.

Now each string $Z \in A$ with $Z \in B$ of the length $l \in [(m-1)N, mN]$ is found as

$$Y_j \in X_i, \quad Y_{j+1} \in X_{i+1}, \quad \dots, \quad Y_{j+m} \in X_{i+m}$$


with started position being localised inside a segment
 $[N(i-1)+1, N(i+1)]$ in A and inside a segment $[N(j-1)+1, Nj]$ in B
 and Z can be found by usual technique.


Substring indicator

$$\mathbf{i}_s(X) = \{i : s_i \in X\}$$

Then $X_1 \in X_2 \iff \mathbf{i}_s(X_1) \subset \mathbf{i}_s(X_2)$ with a high probability.

- Cover $A = (a_1, \dots, a_n)$ by the overlapping strings

$X_i = (a_{N(i-1)+1}, \dots, a_{N(i+1)})$: 

- Split B to strings $Y_j = (b_{N(j-1)+1}, \dots, b_{Nj})$. 

- Find spy indexes for each X_i, Y_j in parallel and linear time.

- Check the feasibility of conditions $\mathbf{i}_s(X_i) \subset \mathbf{i}_s(Y_j)$
 in nm/N^2 simple logical operation on long integers.

Now each string $Z \in A$ with $Z \in B$ of the length $l \in [(m-1)N, mN]$ is found as



$$Y_j \in X_i, \quad Y_{j+1} \in X_{i+1}, \quad \dots, \quad Y_{j+m} \in X_{i+m}$$

with started position being localised inside a segment $[N(i-1)+1, N(i+1)]$ in A and inside a segment $[N(j-1)+1, Nj]$ in B and Z can be found by usual technique.

Substring indicator

$$\mathbf{i}_s(X) = \{i : s_i \in X\}$$

Then $X_1 \in X_2 \iff \mathbf{i}_s(X_1) \subset \mathbf{i}_s(X_2)$ with a high probability.

- Cover $A = (a_1, \dots, a_n)$ by the overlapping strings
 $X_i = (a_{N(i-1)+1}, \dots, a_{N(i+1)})$: 
- Split B to strings $Y_j = (b_{N(j-1)+1}, \dots, b_{Nj})$. 
- Find spy indexes for each X_i, Y_j in parallel and linear time.
- Check the feasibility of conditions $\mathbf{i}_s(X_i) \subset \mathbf{i}_s(Y_j)$
 in nm/N^2 simple logical operation on long integers.

Now each string $Z \in A$ with $Z \in B$ of the length $l \in [(m-1)N, mN]$ is found as

$$Y_j \in X_i, \quad Y_{j+1} \in X_{i+1}, \quad \dots, \quad Y_{j+m} \in X_{i+m}$$

with started position being localised inside a segment
 $[N(i-1)+1, N(i+1)]$ in A and inside a segment $[N(j-1)+1, Nj]$ in B
 and Z can be found by usual technique.

Resume

The talk discovered the major shortcomings of the Levenshtein Distance/LCS method

The belief function theory leads to a flexible framework for similarity evaluation.

A sketch of better sequence alignment algorithm illustrates the frameworks utility.

Thank you for your time and attention!

Your questions are welcome!