Efficient Edit Distance based String Similarity Search using Deletion Neighborhoods

Shashwat Mishra, Tejas Gandhi, Akhil Arora, Arnab Bhattacharya

Special Interest Group in Data,
IIT Kanpur

String Similarity Search/Join Workshop,
EDBT 2013

March 21, 2013

Introduction

Main Task

Introduction

Main Task

- Given a dictionary of strings, perform fast lookups/joins of *similar* strings in the dictionary.
- Use edit distance to quantify the notion of similarity between strings.

Introduction

Main Task

- Given a dictionary of strings, perform fast lookups/joins of *similar* strings in the dictionary.
- Use edit distance to quantify the notion of similarity between strings.
- Two tracks:
  1. Join: Perform self-join of strings. Given a threshold τ, list all string pairs in the dictionary with edit-distance $\leq \tau$.
  2. **Search**: Process range queries as specified in a supplied query file.

Introduction

Search Track: Problem Statement

Introduction

Search Track: Problem Statement

Given a set of strings, D, and a query tuple containing a query string, q, and an edit distance threshold τ, identify all pairs $< q, s >$ s.t. $s \in D$ and $EditDistance(q, s) \leq \tau$.

Introduction

Search Track: Problem Statement

Given a set of strings, D, and a query tuple containing a query string, q, and an edit distance threshold τ, identify all pairs $< q, s >$ s.t. $s \in D$ and $EditDistance(q, s) \leq \tau$.

- Generate and maintain an index structure for the dictionary respecting certain time and memory constraints.
- Use the index structure to process a list of queries (2-tuples). List all *answers* in the specified format.
- **Evaluation Parameter:** Minimize total time taken to process all queries.

Problem Statement: Further Details

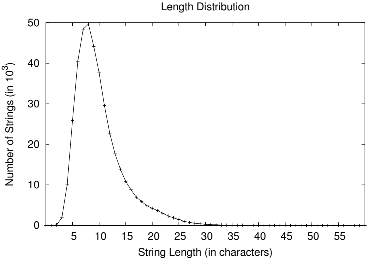
- Index construction time not counted towards the score, but must be less than 3 hrs.
- Peak memory consumption must be less than 48 GB.
- Score dependent solely on $T_{effective}$ where

$$T_{effective} = t_{end} - t_{begin}$$

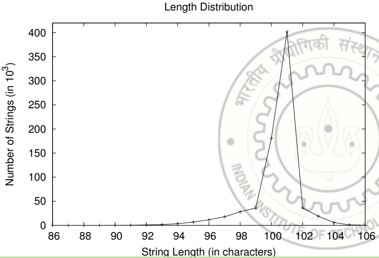
  where t_{begin} and t_{end} are time instances marking the beginning and the end of the processing of the query file, respectively.
- Evaluation environment: 8 cores, 64 GB RAM, FC 17.

The Dictionary

- Dictionary 1: Geographical names
  - Names of places from across the globe.
  - Character-set $|\Sigma|=255$
  - Mean string length $\mu=10$
  - Dictionary size $|D|=400\text{K}$



- Dictionary 2: Human genome read data
  - Strings containing human genomic data.
  - Character-set $|\Sigma|=4$
  - Mean string length $\mu=100.3$
  - Dictionary size $|D|=750\text{K}$



Typical Methodology

- Observation: Modern methods follow a general scheme.

Typical Methodology

- Observation: Modern methods follow a general scheme.
  - Generate a signature for each string in the dictionary.
  - Maintain an inverted index for generated signatures.
  - Signature must result in a (tight) filtering criteria.

Typical Methodology

- Observation: Modern methods follow a general scheme.
  - Generate a signature for each string in the dictionary.
  - Maintain an inverted index for generated signatures.
  - Signature must result in a (tight) filtering criteria.
  - Given a query string, q, use the filter and query signature to generate a candidate list.
  - For each string s' in the candidate list, verify if s' is answer.

Typical Methodology

- Observation: Modern methods follow a general scheme.
  - Generate a signature for each string in the dictionary.
  - Maintain an inverted index for generated signatures.
  - Signature must result in a (tight) filtering criteria.
  - Given a query string, q, use the filter and query signature to generate a candidate list.
  - For each string s' in the candidate list, verify if s' is answer.
- *Signature results in a filtering criteria.* **?**
  - Signature Function $SF(s)$ maps string s to a signature space.
  - Filtering condition is specific to the choice of the signature function.
  - In general, for string s, query string q, threshold τ, filtering condition is an inequality.

$$F(s, q, \tau, SF) <>\neq 0$$

Signature

- Completeness of solution required $\rightarrow$ *No False Dismissals*.
- Filter condition must ensure no false dismissals.

Signature

- Completeness of solution required $\rightarrow$ *No False Dismissals*.
- Filter condition must ensure no false dismissals.
- For query $< q, \tau >$, if $ED(s, q) \leq \tau$ then $F(s, q, \tau, SF)$ must hold.
- Computation of signature should be computationally non-expensive.
- Filter condition should be tight. Resultant candidate list should be small.
- Popular signature schemes in literature:
  - Q-Gram
  - Deletion Neighborhood

Signature: Deletion Neighborhood

- Defined w.r.t. a specified edit distance threshold.
- For a string s and edit distance τ, deletion neighborhood signature of s, $SF_{\tau}(s)$ is the set of all strings that can be generated by deleting *at-most* τ characters from s.

Signature: Deletion Neighborhood

- Defined w.r.t. a specified edit distance threshold.
- For a string s and edit distance τ, deletion neighborhood signature of s, $SF_\tau(s)$ is the set of all strings that can be generated by deleting *at-most* τ characters from s.
- Ex. $s = "SIGDATA"$

Signature: Deletion Neighborhood

- Defined w.r.t. a specified edit distance threshold.
- For a string s and edit distance τ, deletion neighborhood signature of s, $SF_\tau(s)$ is the set of all strings that can be generated by deleting *at-most* τ characters from s.
- Ex. $s =$ *"SIGDATA"*
  - $SF_1(s) =$ {*"IGDATA"*, *"SGDATA"*, *"SIDATA"*, *"SIGATA"*, *"SIGDTA"*, *"SIGDAA"*, *"SIGDAT"*, *"SIGDATA"*}

Signature: Deletion Neighborhood

- Defined w.r.t. a specified edit distance threshold.
- For a string s and edit distance τ, deletion neighborhood signature of s, $SF_\tau(s)$ is the set of all strings that can be generated by deleting *at-most* τ characters from s.
- Ex. $s = "SIGDATA"$
  - $SF_1(s) = \{"IGDATA","SGDATA","SIDATA","SIGATA","SIGDTA", "SIGDAA","SIGDAT", "SIGDATA"\}$
- Filtering Condition: -

Signature: Deletion Neighborhood

- Defined w.r.t. a specified edit distance threshold.
- For a string s and edit distance τ, deletion neighborhood signature of s, $SF_\tau(s)$ is the set of all strings that can be generated by deleting *at-most* τ characters from s.
- Ex. $s = $ *"SIGDATA"*
  - $SF_1(s)$ = {*"IGDATA","SGDATA","SIDATA","SIGATA","SIGDTA", "SIGDAA","SIGDAT", "SIGDATA"*}
- Filtering Condition: -

$$\text{If } ED(s_1,s_2) \leq \tau, \text{ then } |SF_\tau(s_1) \cap SF_\tau(s_2)| > 0$$

Signature: Deletion Neighborhood

- Defined w.r.t. a specified edit distance threshold.
- For a string s and edit distance τ, deletion neighborhood signature of s, $SF_\tau(s)$ is the set of all strings that can be generated by deleting *at-most* τ characters from s.
- Ex. $s = $ *"SIGDATA"*
  - $SF_1(s) = \{$*"IGDATA","SGDATA","SIDATA","SIGATA","SIGDTA", "SIGDAA","SIGDAT", "SIGDATA"*$\}$
- Filtering Condition: -

$$If\ ED(s_1,s_2) \leq \tau,\ then\ |SF_\tau(s_1) \cap SF_\tau(s_2)| > 0$$

- Intuition: Both strings s_1, s_2 should be reducible to a common form after at-most τ deletion operations.

Signature: Q-Gram

- For a string s, Q-Gram signature of s, $SF(s)$ is the set of all strings that can be generated by taking q contiguous characters from s.
- Ex. $s = "SIGDATA"$, $Q = 3$
  - $SF(s) = \{"SIG","IGD","GDA","DAT","ATA"\}$

Signature: Q-Gram

- For a string s, Q-Gram signature of s, $SF(s)$ is the set of all strings that can be generated by taking q contiguous characters from s.
- Ex. $s =$ "*SIGDATA*", $Q = 3$
  - $SF(s) =$ {"*SIG*","*IGD*","*GDA*","*DAT*","*ATA*"}
- Filtering Condition: -

Signature: Q-Gram

- For a string s, Q-Gram signature of s, $SF(s)$ is the set of all strings that can be generated by taking q contiguous characters from s.
- Ex. $s = "SIGDATA"$, $Q = 3$
  - $SF(s) = \{"SIG","IGD","GDA","DAT","ATA"\}$
- Filtering Condition: -

$$If\ ED(s_1,s_2) \leq \tau,\ then\ |SF(s_1) \cap SF(s_2)| \geq max(|s_1|,|s_2|) + 1 - (\tau+1).Q$$

Signature: Q-Gram

- For a string s, Q-Gram signature of s, $SF(s)$ is the set of all strings that can be generated by taking q contiguous characters from s.
- Ex. $s =$ *"SIGDATA"*, $Q = 3$
  - $SF(s) = \{$"SIG","IGD","GDA","DAT","ATA"$\}$
- Filtering Condition: -

$$\text{If } ED(s_1, s_2) \leq \tau, \text{ then } |SF(s_1) \cap SF(s_2)| \geq max(|s_1|, |s_2|) + 1 - (\tau + 1).Q$$

- In practice, RHS $= |q| + 1 - (\tau + 1).Q$, where q is the query string.

Signature: Q-Gram

- For a string s, Q-Gram signature of s, $SF(s)$ is the set of all strings that can be generated by taking q contiguous characters from s.
- Ex. $s =$ *"SIGDATA"*, $Q = 3$
  - $SF(s) = \{$"SIG","IGD","GDA","DAT","ATA"$\}$
- Filtering Condition: -

$$If\ ED(s_1,s_2) \leq \tau,\ then\ |SF(s_1) \cap SF(s_2)| \geq max(|s_1|,|s_2|) + 1 - (\tau+1).Q$$

- In practice, RHS $= |q| + 1 - (\tau + 1).Q$, where q is the query string.
- For filter to be tight, RHS $> 0 \rightarrow |q| > (\tau + 1).Q - 1$.
  - Serious limitation ! For $Q = 2$, $\tau = 2$, only queries with $|q| \geq 6$ can be processed, for $\tau = 3$, $|q| \geq 8$.

Choice of Signature

- Deletion Neighborhood seems to be better than Q-Gram.
  - No restriction on query size.
  - Every inverted list should (expectedly) be more selective.

Choice of Signature

- Deletion Neighborhood seems to be better than Q-Gram.
  - No restriction on query size.
  - Every inverted list should (expectedly) be more selective.
- However, they have **high** space requirement.
- For $|s| = 14$, $\tau = 2$,
  - $|SF_{3-Gram}| = |s| - Q + 1 = 12$ $O(|s|)$
  - $|SF_{Deletion}| = \binom{|s|}{\tau} = 91$ $O(|s|^{\tau})$

Our System

Design Decision

Decided to implement a system following the generic scheme and using deletion neighborhood as the signature scheme.

Choice of Signature

- Deletion Neighborhood seems to be better than Q-Gram.
  - No restriction on query size.
  - Every inverted list should (expectedly) be more selective.
- However, they have **high** space requirement.
- For $|s| = 14$, $\tau = 2$,
  - $|SF_{3-Gram}| = |s| - Q + 1 = 12$ $\qquad O(|s|)$
  - $|SF_{Deletion}| = \binom{|s|}{\tau} = 91$ $\qquad O(|s|^\tau)$
- **Challenge:** Reduce space complexity of a deletion neighborhood signature based system while maintaining completeness of solution.

Choice of Signature

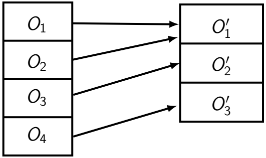
- Deletion Neighborhood seems to be better than Q-Gram.
  - No restriction on query size.
  - Every inverted list should (expectedly) be more selective.
- However, they have **high** space requirement.
- For $|s| = 14$, $\tau = 2$,
  - $|SF_{3-Gram}| = |s| - Q + 1 = 12$ $\qquad O(|s|)$
  - $|SF_{Deletion}| = \binom{|s|}{\tau} = 91$ $\qquad O(|s|^\tau)$
- **Challenge:** Reduce space complexity of a deletion neighborhood signature based system while maintaining completeness of solution.
- Signature defined w.r.t. a threshold, need dedicated index structures, I_τ, for each threshold, $\tau = [0 : 4]$.

Reducing Space Requirement

- **Key Idea:** Introduce collisions among objects in $SF_{\tau}(s)$.

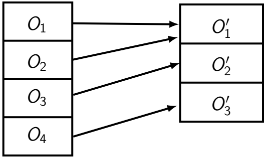
Reducing Space Requirement

- **Key Idea:** Introduce collisions among objects in $SF_\tau(s)$.



Reducing Space Requirement

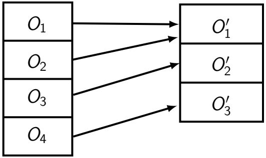
- **Key Idea:** Introduce collisions among objects in $SF_\tau(s)$.



- Possible Solution ?

Reducing Space Requirement

- **Key Idea:** Introduce collisions among objects in $SF_\tau(s)$.



- Possible Solution ? **Hashing !**
- For $O \in SF_\tau(s)$, $h(O) = suffix_L(O)$.
  - Hashing results in a reduced set representation of $SF_\tau(s)$.
  - Ex. for $s =$"$SIGDATA$", $\tau = 1$, $L = 3$,
    - $h(SF(s)) = \{$"ATA", "DTA", "DAA", "DAT"$\}$
    - $|h(SF_\tau(s))| = 4 < |SF_\tau(s)| = 8$
- Added Benefit: Need not generate all elements in $SF_\tau(s)$. $|h(SF_\tau(s))| = O(\binom{L+\tau}{\tau})$.

Our System

Design Decision

To implement a system following the generic scheme and using deletion neighborhood as the signature.

Our System

Design Decision

To implement a system following the generic scheme and using deletion neighborhood as the signature.

Reduction of Space Requirement

To hash generated signature and index the resultant strings.

Hashing: Completeness Guarantee ?

- Any hashing scheme guarantees the completeness of solution.

Hashing: Completeness Guarantee ?

- Any hashing scheme guarantees the completeness of solution.
  - If s is an answer for a query $< q, \tau >$, then $\exists$ o s.t. $o \in SF_\tau(s)$ and $o \in SF_\tau(q)$.
  - Thus, $h(o) \in h(SF_\tau(s)), h(SF_\tau(q))$, and hence s is in the candidate list.

Hashing: Completeness Guarantee ?

- Any hashing scheme guarantees the completeness of solution.
  - If s is an answer for a query $< q, \tau >$, then $\exists$ o s.t. $o \in SF_\tau(s)$ and $o \in SF_\tau(q)$.
  - Thus, $h(o) \in h(SF_\tau(s)), h(SF_\tau(q))$, and hence s is in the candidate list.
- Why suffix ?

Hashing: Completeness Guarantee ?

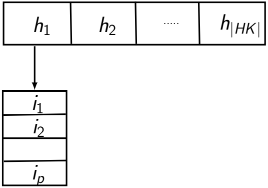
- Any hashing scheme guarantees the completeness of solution.
  - If s is an answer for a query $< q, \tau >$, then $\exists$ o s.t. $o \in SF_\tau(s)$ and $o \in SF_\tau(q)$.
  - Thus, $h(o) \in h(SF_\tau(s)), h(SF_\tau(q))$, and hence s is in the candidate list.
- Why suffix ?
  - No real reason. Initial design decision.
  - Performed well, stuck around.
- Possibly lucrative to try other hash functions/schemes.

Hashing: Layout

- So what does I_τ look like ?

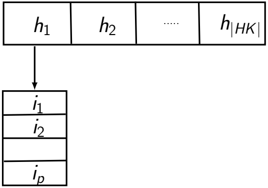
Hashing: Layout

- So what does I_τ look like ?
- **Hash Table !**



Hashing: Layout

- So what does I_τ look like ?
- **Hash Table !**



  - Keys consist of string resulting from hash (suffix operation).
  - List consists of *ids* of strings in D that generate the key.

Bucketing

Bucketing

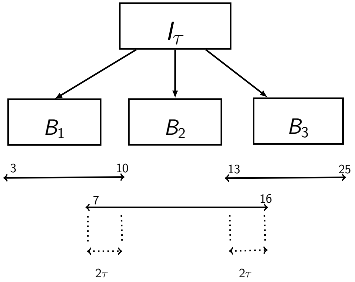
- Should I_τ be a single structure (hash-table) for all strings ?

Bucketing

- Should I_τ be a single structure (hash-table) for all strings ?
- No reason why !
- Infact it helps to partitions strings on the basis of length.

Bucketing

- Should I_τ be a single structure (hash-table) for all strings ?
- No reason why !
- Infact it helps to partitions strings on the basis of length.
  - I_τ consists of buckets.
  - Every bucket responsible for indexing strings within a particular length range.



Bucketing: Why ?

- Ex. Let $s =$"*PLACATING*", $|s| = 9$.
- Query $< BATING, 2 >$. $|q| = 6$.
- For l_2, if $L = 4$, s will be in the candidate list.
  - "*TING*" common hash-key.

Bucketing: Why ?

- Ex. Let $s =$"*PLACATING*", $|s| = 9$.
- Query $< BATING, 2 >$. $|q| = 6$.
- For l_2, if $L = 4$, s will be in the candidate list.
  - "*TING*" common hash-key.
- $abs(|s| - |q|) > \tau$. s is ruled out.
- Apply length filter higher up in the pipeline.

Bucketing: Why ?

- Ex. Let $s =$"*PLACATING*", $|s| = 9$.
- Query $< BATING, 2 >$. $|q| = 6$.
- For I_2, if $L = 4$, s will be in the candidate list.
  - "*TING*" common hash-key.
- $abs(|s| - |q|) > \tau$. s is ruled out.
- Apply length filter higher up in the pipeline.
- Say I_τ had B_1 s.t. the bucket was responsible for range [1 : 8].
  - Query could be answered by B_1 alone.
  - B_1 would not index "*PLACATING*".

Bucketing: Why ?

- Ex. Let $s =$"*PLACATING*", $|s| = 9$.
- Query $< BATING, 2 >$. $|q| = 6$.
- For I_2, if $L = 4$, s will be in the candidate list.
    - "*TING*" common hash-key.
- $abs(|s| - |q|) > \tau$. s is ruled out.
- Apply length filter higher up in the pipeline.
- Say I_τ had B_1 s.t. the bucket was responsible for range [1 : 8].
    - Query could be answered by B_1 alone.
    - B_1 would not index "*PLACATING*".
- 25% reduction in average search time for $\tau = 2$.
  $150\mu s \rightarrow 110\mu s$.

Our System

Design Decision

To implement a system following the generic scheme and using deletion neighborhood as the signature.

Reduction of Space Requirement

To hash generated signature and index the resultant strings.

Our System

Design Decision

To implement a system following the generic scheme and using deletion neighborhood as the signature.

Reduction of Space Requirement

To hash generated signature and index the resultant strings.

Bucketing: Reducing collisions in Hash-Table

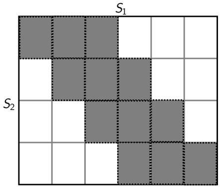
Partition strings on basis of length. Apply early length filtering. Results in smaller individual hash-tables. Increases space requirement.

Verification

- How to verify if $s \in$ *Candidate-List* is an answer ? Check if $ED(s, q) \leq \tau$

Verification

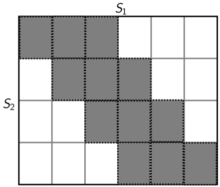
- How to verify if $s \in$ *Candidate-List* is an answer ? Check if $ED(s, q) \leq \tau$
- Naive Method: Explicitly compute $ED(s, q)$.
- Not interested in value of $ED(s, q)$.



- For each row i, only need to examine j s.t. $i - \left\lfloor \frac{\tau-\delta}{2} \right\rfloor < j < i + \left\lfloor \frac{\tau+\delta}{2} \right\rfloor$. $\delta = abs(|s_1| - |s_2|)$.

Verification

- How to verify if $s \in$ *Candidate-List* is an answer ? Check if $ED(s, q) \leq \tau$
- Naive Method: Explicitly compute $ED(s, q)$.
- Not interested in value of $ED(s, q)$.



- For each row i, only need to examine j s.t. $i - \left\lfloor \frac{\tau - \delta}{2} \right\rfloor < j < i + \left\lfloor \frac{\tau + \delta}{2} \right\rfloor$. $\delta = abs(|s_1| - |s_2|)$.
- Li et al., VLDB 2012.

Execution

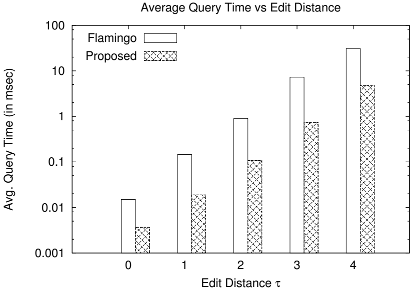
- Generate dedicated index structures, I_τ for each τ.
- Read and group all queries depending on threshold τ.
- Sequentially process queries for each $\tau = [0:4]$.
  - Multiple threads read their own queries.
  - Thread j responsible for all queries s.t. $id_q \ \% \ 8 = j$.
  - Result of each query written to a global buffer in memory. Contention between threads on memory write.
- Flush the buffer to the disk.

Results

- So how do we fare against a state-of-the-art ?

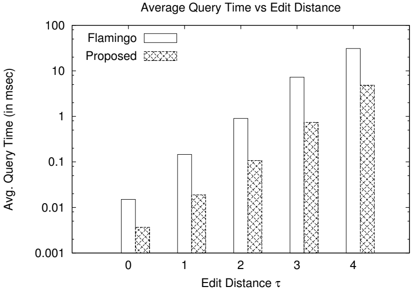
Results

- So how do we fare against a state-of-the-art ?



Results

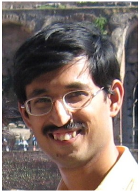
- So how do we fare against a state-of-the-art ?



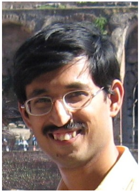
τ	Avg. query time (ms)	
	Flamingo	Proposed
0	0.015	0.004
1	0.146	0.019
2	0.901	0.108
3	7.245	0.736
4	30.906	4.801

The Team

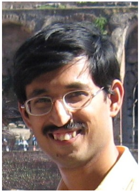
The Team



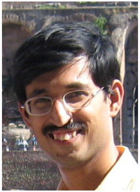
The Team



The Team



The Team



Thank You !