Approximate Queries on String Collections

Xiaochun Yang

Institute of Computer Software and Theory
School of Information Science and Engineering
Northeastern University, CHINA
yangxc@mail.neu.edu.cn
Outline

• What is approximate query?
• Q-gram based algorithms
• Our research results
  – VGRAM [VLDB’07]
  – Gram selection [SIGMOD’08]
Example: a movie database

The user doesn’t know the exact spelling!

<table>
<thead>
<tr>
<th>Star</th>
<th>Title</th>
<th>Year</th>
<th>Genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keanu Reeves</td>
<td>The Matrix</td>
<td>1999</td>
<td>Sci-Fi</td>
</tr>
<tr>
<td>Samuel Jackson</td>
<td>Iron man</td>
<td>2008</td>
<td>Sci-Fi</td>
</tr>
<tr>
<td>Schwarzenegger</td>
<td>The Terminator</td>
<td>1984</td>
<td>Sci-Fi</td>
</tr>
<tr>
<td>Samuel Jackson</td>
<td>The man</td>
<td>2006</td>
<td>Crime</td>
</tr>
</tbody>
</table>
Gap between Queries and Data

- Errors in the query
  - The user doesn’t remember a string exactly
  - The user unintentionally types a wrong string
  - Limited input device

Query: Schwarrzenger. Data: Schwarzenegger
Data integration and cleansing

- Errors in the database:
  - Data often is not clean by itself, especially true in data integration and cleansing
  - Typos, Web data, OCR

<table>
<thead>
<tr>
<th>Relation R</th>
<th>Relation S</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Star</strong></td>
<td><strong>Star</strong></td>
</tr>
<tr>
<td>Keanu Reeves</td>
<td>Keanu Reeves</td>
</tr>
<tr>
<td>Samuel Jackson</td>
<td>Samuel L. Jackson</td>
</tr>
<tr>
<td>Schwarzenegger</td>
<td>Schwarzenegger</td>
</tr>
<tr>
<td>Samuel Jackson</td>
<td>Samuel L. Jackson</td>
</tr>
</tbody>
</table>
Google

nec univerge platform

Results 1 - 10 of about 22,100 for nec univerge platform (0.12 seconds)

Did you mean: nec univerge platform

Freightmasters Improves Workforce Flexibility, Productivity with ...
With the Loeller Companies' assistance, Freightmasters implemented NEC's UNIVERGE SV7000 MPS communications platform for small- and mid-sized businesses in ...
findarticles.com/p/articles/mi_m0EINjs_2007_August_20/ai_n19453265 - 25k - Cached - Similar pages - Note this

NEC Univerge VOIP Platform - Telnet-Tech.com
NEC Univerge VOIP Platform, NEC Univerge IP Phones Powerful IP Communication Tools For The Desktop And Beyond. The NEAX/DPS-DMR ...
www.telnet-tech.com/sip/nec.html - 5k - Cached - Similar pages - Note this

Debt and Credit Advisors Launches New Call Center with NEC's ...
*Debt and Credit Advisors opened for business on October 1 using the NEC UNIVERGE platform and UC for Business as the technology centerpiece of our call ...
www.tmcnet.com/usuomi/-debt-credit-advisors-launches-new-call-center-with ...

NEC Announces UNIVERGE(R) OW5000 Application Platform for ...
NEC Announces UNIVERGE(R) OW5000 Application Platform for Developer Partners.
www.tmcnet.com/.../2007/05/02/2573951.htm - 131k - Cached - Similar pages - Note this
Spell checking

You choose a very very hot season (Olympic Games) :)
If we go, we'll probably visit Beijing at the end of July. I bet we can't find hotel in Aug.:(

In fact, it only costs 4 hours from Beijing to Shenyang by train, or 40 mins by air plane.
So, you're also welcome to visit Northeastern University (NEU) at your convenience.

Let's try to figure out a good schedule for us to join together.

Bests,
Xiaochun
Approximate query

• Approximate search
• Approximate join
Approximate search

Collection of strings $s$

Query $q$
Samuel Jackson

Output: strings $s$ that satisfy $\text{Sim}(q, s) \leq \delta$
Approximate join

Collection R

<table>
<thead>
<tr>
<th>Jenny Stamatopoulou</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Paul McDougal</td>
</tr>
<tr>
<td>Aldridge Rodriguez</td>
</tr>
<tr>
<td>Panos Ipeirotis</td>
</tr>
<tr>
<td>John Smith</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Output: string pair that satisfy $\text{Sim}(r,s) \leq \delta$

Collection S

<table>
<thead>
<tr>
<th>Panos Ipeirotis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jonh Smith</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Jenny Stamatopoulou</td>
</tr>
<tr>
<td>John P. McDougal</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Al Dridge Rodriguez</td>
</tr>
</tbody>
</table>

...
Similarity functions

- Edit distance
- Jaccard similarity
- Hamming distance
- Cosine similarity
- …
Performance is a big issue

- Answer queries interactively
- Many queries on a server

<table>
<thead>
<tr>
<th>Response time</th>
<th>5ms/query</th>
<th>20ms/query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>200 queries/second</td>
<td>50 queries/second</td>
</tr>
</tbody>
</table>
Outline

• What is approximate query?
• Q-gram based algorithms
• Our research results
  – VGRAM [VLDB’07]
  – Gram selection [SIGMOD’08]
"q-grams" of strings

2-grams

\( V(s, q) \)

(\( un, ni, iv, ve, er, rs, sa, al \))

String with length \( L \rightarrow L - q + 1 \) q-grams
Similarity between gram sets
q-gram based inverted lists index

<table>
<thead>
<tr>
<th>id</th>
<th>strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rich</td>
</tr>
<tr>
<td>1</td>
<td>stick</td>
</tr>
<tr>
<td>2</td>
<td>stich</td>
</tr>
<tr>
<td>3</td>
<td>stuck</td>
</tr>
<tr>
<td>4</td>
<td>static</td>
</tr>
</tbody>
</table>
Approximate query processing

- Query: “shtick”, $\text{ED}(\text{shtick}, ?) \leq 1$
  
  \begin{align*}
  \text{sh} & \quad \text{ht} & \quad \text{ti} & \quad \text{ic} & \quad \text{ck} \\
  # \text{of common grams} & \geq 3
\end{align*}

<table>
<thead>
<tr>
<th>id</th>
<th>strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rich</td>
</tr>
<tr>
<td>1</td>
<td>stick</td>
</tr>
<tr>
<td>2</td>
<td>stich</td>
</tr>
<tr>
<td>3</td>
<td>stuck</td>
</tr>
<tr>
<td>4</td>
<td>static</td>
</tr>
</tbody>
</table>

2-grams:

- at: 4
- ch: 0 → 2
- ck: 0 → 3
- ic: 0 → 1 → 2 → 4
- ri: 0
- st: 1 → 2 → 3 → 4
- ta: 4
- ti: 1 → 2 → 4
- tu: 3
- uc: 3
2-grams -> 3-grams?

- Query: “shtick”, ED(shtick, ?) ≤ 1

  sht  hti  tic  ick

  # of common grams >= 1

<table>
<thead>
<tr>
<th>id</th>
<th>strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rich</td>
</tr>
<tr>
<td>1</td>
<td>stick</td>
</tr>
<tr>
<td>2</td>
<td>stich</td>
</tr>
<tr>
<td>3</td>
<td>stuck</td>
</tr>
<tr>
<td>4</td>
<td>static</td>
</tr>
</tbody>
</table>

3-grams:

- ati → 4
- ich → 0 → 2
- ick → 1
- ric → 0
- sta → 4
- sti → 1 → 2
- stu → 3
- tat → 4
- tic → 1 → 2 → 4
- tuc → 3
- uck → 3
Observation 1: skew distributions of gram frequencies

- DBLP: 276,699 article titles
- Popular 5-grams: ation (>114K times), tions, ystem, catio
Observation 2: dilemma of choosing “q”

- Increasing “q” causing:
  - Longer grams → Shorter lists
  - Smaller # of common grams of similar strings

<table>
<thead>
<tr>
<th>id</th>
<th>strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rich</td>
</tr>
<tr>
<td>1</td>
<td>stick</td>
</tr>
<tr>
<td>2</td>
<td>stich</td>
</tr>
<tr>
<td>3</td>
<td>stuck</td>
</tr>
<tr>
<td>4</td>
<td>static</td>
</tr>
</tbody>
</table>

2-grams:
- at: 4
- ch: 0 → 2
- ck: 1 → 3
- ic: 0 → 1 → 2 → 4
- ri: 0
- st: 1 → 2 → 3 → 4
- ta: 4
- ti: 1 → 2 → 4
- tu: 3
- uc: 3
Motivation

- Small index size (memory)
- Small running time
  - Scan matched inverted lists
  - Calculate ED(query, candidate)
What we got?

• VLDB 2007
  – **VGRAM**: Improving Performance of Approximate Queries on String Collections Using Variable-Length Grams

• SIGMOD 2008
  – Cost-Based Variable-Length-Gram Selection for String Collections to Support Approximate Queries Efficiently
VGRAM: Main idea

- Grams with **variable lengths** (between $q_{min}$ and $q_{max}$)
  - zebra
    - $ze(123)$
  - **corrasion**
    - $co(5213)$, $cor(859)$, $corr(171)$

- **Advantages**
  - Reduce index size 😊
  - Reducing running time 😊
  - Adoptable by many algorithms 😊
Challenges

- Generating variable-length grams?
- Constructing a high-quality gram dictionary?
- Relationship between string similarity and their gram-set similarity?
- Adopting VGRAM in existing algorithms?
Challenge 1: String $\rightarrow$ Variable-length grams?

- Fixed-length 2-grams

```
universal
```

- Variable-length grams

[2,4]-gram dictionary

```
universal
```

```
| ni | ivr | sal | uni | vers |
```
Representing gram dictionary as a trie

- Fixed-length 2-grams
  
  \textit{universal}

- Variable-length grams

  \textbf{[2,4]-gram}

  \begin{tabular}{c|c|c|c|c|c|c}
    ni &ivr &sal &uni &vers \\
  \end{tabular}
Challenge 2: Constructing gram dictionary

- selecting grams
  - Pruning trie using a frequency threshold \( T \) (e.g., 2)

A gram-frequency trie: \([2, 4]\)-gram

<table>
<thead>
<tr>
<th>id</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>stick</td>
</tr>
<tr>
<td>1</td>
<td>stich</td>
</tr>
<tr>
<td>2</td>
<td>such</td>
</tr>
<tr>
<td>3</td>
<td>stuck</td>
</tr>
</tbody>
</table>
Challenge 2: Constructing gram dictionary

- selecting grams
  - Pruning trie using a frequency threshold \( T \) (e.g., 2)

A gram-frequency trie: \([2, 4]\)-gram
Final gram dictionary

(a) strings

Final grams
• VGRAM
  – Main idea
  – Decomposing strings to grams
  – Choosing good grams
  – Effect of edit operations on grams
  – Adopting vgram in existing algorithms
• Experiments
Challenge 3: Edit operation’s effect on grams

$k$ operations could affect $k \times q$ grams
Deletion affects variable-length grams.

Not affected

Affected

Not affected

\[ i - q_{\text{max}} + 1 \leq i \leq i + q_{\text{max}} - 1 \]
Grams affected by a deletion

\[ \text{Affected?} \]

\[ i-q_{\text{max}} + 1 \quad \text{Deletion} \quad i \quad i+q_{\text{max}} - 1 \]

\text{[2,4]-grams}

\[
\begin{array}{c}
\text{ni} \\
\text{ivr} \\
\text{sal} \\
\text{uni} \\
\text{vers}
\end{array}
\]
# of grams affected by each operation

Deletion/substitution
\[
\begin{array}{cccccccccc}
0 & 1 & 1 & 1 & 1 & 2 & 1 & 2 & 2 & 1 \\
\end{array}
\]

Insertion
\[
\begin{array}{cccccccccc}
\_ & u & n & i & v & e & r & s & a & l \_ \\
\_ & u & n & i & v & e & r & s & a & l \_ \\
\end{array}
\]
# of grams affected by k operations

- $k$-Max: Summation of $k$ largest values
  \[ \text{NAG}(s,2) = 3 + 3 = 6 \]

Too pessimistic?
Tightening NAG(s,k) [SIGMOD'08]

- Dynamic programming: tightening NAG(s,k)
  - Subproblems: NAG(s[1,j], i)

![Diagram showing string s with subproblems op_i highlighted]
Dynamic programming

• Recurrence function
Dynamic programming

\[
P(i, j) = \max \begin{cases} 
P(i, j - 1), & \text{(no operation at } j) \\
P(i - 1, R(j)) + B[j], & \text{(operations at } j) 
\end{cases}
\]
**Lower bound on # of common grams**

---

**Fixed length** ($q$)

If $ed(s1, s2) \leq k$, then their # of common grams $\geq$:

$$\left(|s_1|- q + 1\right) - k \times q$$

**Variable lengths**: # of grams of $s1$ – $NAG(s1, k)$
Challenge 4: adopting VGRAM

Easily adoptable by many algorithms

Basic interfaces:

- String $s \rightarrow$ grams
- String $s_1, s_2$ such that $ed(s_1, s_2) \leq k \rightarrow \min \# \text{ of their common grams}$
Example: algorithm using inverted lists

- Query: “shtick”, $ED(\text{shtick}, ?) \leq 1$

```
Query: “shtick”, ED(shtick, ?)≤1

2-grams

<table>
<thead>
<tr>
<th>2-grams</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ck</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ic</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>tic</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Lower bound = 3

2-4 grams

<table>
<thead>
<tr>
<th>2-4 grams</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ck</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>ic</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>ich</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>tic</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>tick</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Lower bound = 1
```
Gram selection [SIGMOD’08]

- Cost-based quantitative approach
  - Analyze and estimate query performance when adding each gram
  - Automatically find high-quality grams
Outline

• Effects of adding a gram on index and queries
• Cost-based construction of gram dictionary
• Experiments
Effects on inverted lists

Gram dictionary

**ab**  
**bc**

string  **--bc--**

Gram dictionary

**ab**  
**bc**  
**abc**

add gram **abc**
Effects on query performance

- Decrease query’s inverted list
- Change lower bound
- Change # of candidates
Effects on query’s inverted lists

Adding a new gram \textit{abc} will not change or decrease the query’s inverted lists.
Effects on lower bound

- Query: Q, ED(Q, ?) ≤ 1

Query Q

- - - - a b c d - - - -

- - - - a b c d - - - -

Query Q
Effects on lower bound

• Query: “bingon”, \( ED(\text{bingon, } ?) \leq 1 \)

| Dictionary | Gram set: VG(Q) | | VG(Q)| | NAG(Q,1) | Lower bound |
|-------------|-----------------|--------|--------|---------|------------|
| D₀          | \{bi, in, ng, go, on\} | 5      | 2      | 3       |
| D₁ (+ ing)  | \{bi, ing, go, on\} | 4      | 2      | 2       |
| D₂ (+ bin)  | \{bin, ing, go, on\} | 4      | 2      | 2       |
Effects on # of candidates

- Change lower bound $\Rightarrow$ change # of candidates

Gram dictionary

- $ab$
- $bc$

Gram dictionary

- $ab$
- $bc$
- $abc$

Query $Q$

- - - - a b c d - - - -

- - - - a b c d - - - -
# Effects of queries

<table>
<thead>
<tr>
<th>Query</th>
<th>Dictionary</th>
<th>Gram set</th>
<th>Scanned list size</th>
<th>Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>bingo</td>
<td>$D_0 {bi, in, ng, go, on}$</td>
<td>19</td>
<td>1,2,3,4,6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$D_1 (+ \text{ing}) {bi, \text{ing}, go, on}$</td>
<td>11</td>
<td>1,3,4,6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$D_2 (+ \text{bin}) {bin, \text{ing}, go, on}$</td>
<td>8</td>
<td>1,6</td>
<td></td>
</tr>
<tr>
<td>bitting</td>
<td>$D_0 {bi, it, tt, ti, in, ng}$</td>
<td>21</td>
<td>3,4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$D_1 (+ \text{ing}) {bi, it, tt, ti, \text{ing}}$</td>
<td>13</td>
<td>3,4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$D_2 (+ \text{bin}) {bi, it, tt, ti, \text{ing}}$</td>
<td>12</td>
<td>1,3,4</td>
<td></td>
</tr>
</tbody>
</table>
Outline

• Effects of adding a gram on index and queries
• Cost-based construction of gram dictionary
• Experiments
Construct a gram dictionary [VLDB’07]
Cost-base construction [SIGMOD’08]
Construct a gram dictionary

\( q_{\text{min}} = 2 \)

<table>
<thead>
<tr>
<th>id</th>
<th>strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bingo</td>
</tr>
<tr>
<td>2</td>
<td>bioinng</td>
</tr>
<tr>
<td>3</td>
<td>bitingin</td>
</tr>
<tr>
<td>4</td>
<td>biting</td>
</tr>
<tr>
<td>5</td>
<td>boing</td>
</tr>
<tr>
<td>6</td>
<td>going</td>
</tr>
</tbody>
</table>
## Experiments -- Data sets

<table>
<thead>
<tr>
<th>Data set</th>
<th>String #</th>
<th>Length</th>
<th>Range of # of injected edit operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>Article Titles</td>
<td>277,000</td>
<td>6</td>
<td>207</td>
</tr>
<tr>
<td>Movie Titles</td>
<td>855,000</td>
<td>8</td>
<td>249</td>
</tr>
<tr>
<td>Actor Names</td>
<td>1,200,000</td>
<td>4</td>
<td>74</td>
</tr>
</tbody>
</table>

**Environment:**
GNU C++, Dell GX620 PC with an Intel Pentium 2.40Hz Dual Core CPU, 2GB memory, 250GB disk, Ubuntu (Linux) O.S.
Index structure were assumed to be in memory
Improving algorithm ProbeCount

Dataset 1: Person name
Improving algorithm ProbeCluster

[VLDB’07]

Dataset 1: Person name
Improving algorithm PartEnum

[VLDB’07]

Dataset 1: Person name
Comparison with algorithm Prune

Dataset: 1M article titles
Prune: qmin=5, qmax=7, T=2000, LargeFirst policy
GramGen: 1% sampling ratio, 2000 queries, (qmin=5 automatically determined)
Comparison with algorithm Prune

[SIGMOD’08]

Prune: qmin=5, qmax=7, T=2000, LargeFirst policy
GramGen: 1% sampling ratio, 2000 queries, (qmin=5 automatically determined)
Conclusions

• VGRAM + gram selection
  – variable-length
  – high-quality
• Adoptable in existing algorithms
  – Reduce index size
  – Reduce running time
How to do research

- Use scientific method
How to do research

• Real application -- Make your approach useful
How to do research

Work hard

Enjoy your work

Make a difference
How to do research

- Focus on detail
- Be efficient
- Beat the problem to death!
Can we run faster?
Thank you