Efficient Approximate Search on String Collections
Part II

- Marios Hadjieleftheriou
  
  Chen Li
  
  at&t
  
  University of California, Irvine
Overview

- Sketch based algorithms
- Compression
- Selectivity estimation
- Transformations/Synonyms
Selection Queries Using Sketch Based Algorithms
What is a Sketch

- An approximate representation of the string
- With size much smaller than the string
- That can be used to upper bound similarity (or lower bound the distance)
  - String $s$ has sketch $\text{sig}(s)$
  - If $\text{Sim}_{\text{sig}}(\text{sig}(s), \text{sig}(t)) < \theta$, prune $t$
- And $\text{Sim}_{\text{sig}}$ is much more efficient to compute than the actual similarity of $s$ and $t$
Using Sketches for Selection Queries

- Naïve approach:
  - Scan all sketches and identify candidates
  - Verify candidates

- Index sketches:
  - Inverted index
  - LSH (Gionis et al.)
  - The inverted sketch hash table (Chakrabarti et al.)
Known sketches

- Prefix filter (CGK06)
  - Jaccard, Cosine, Edit distance
- Mismatch filter (XWL08)
  - Edit distance
- Minhash (BCFM98)
  - Jaccard
- PartEnum (AGK06)
  - Hamming, Jaccard
Prefix Filter

- Construction
  - Let string $s$ be a set of q-grams:
    - $s = \{q_4, q_1, q_2, q_7\}$
  - Sort the set (e.g., lexicographically)
    - $s = \{q_1, q_2, q_4, q_7\}$
  - Prefix sketch $\text{sig}(s) = \{q_1, q_2\}$

- Use sketch for filtering
  - If $|s \cap t| \geq \theta$ then $\text{sig}(s) \cap \text{sig}(t) \neq \emptyset$
Example

- Sets of size 8:

  \[
  s = \{q_1, q_2, q_4, q_6, q_8, q_9, q_{10}, q_{12}\}
  \]
  \[
  t = \{q_1, q_2, q_5, q_6, q_8, q_{10}, q_{12}, q_{14}\}
  \]

  \[s \cap t = \{q_1, q_2, q_6, q_8, q_{10}, q_{12}\}, |s \cap t| = 6\]

- For any subset \(t' \subset t\) of size 3: \(s \cap t' \neq \emptyset\)
  - In worst case we choose \(q_5, q_{14}, \text{and } ??\)

- If \(|s \cap t| \geq \theta\) then \(\forall t' \subset t\) s.t. \(|t'| \geq |s| - \theta + 1\), \(t' \cap s \neq \emptyset\)
Example continued

- Instead of taking a **subset** of \( t \), we **sort** and take **prefixes** from both \( s \) and \( t \):
  - \( \text{pf}(s) = \{q_1, q_2, q_4\} \)
  - \( \text{pf}(t) = \{q_1, q_2, q_5\} \)
  - If \( |s \cap t| \geq 6 \) then \( \text{pf}(s) \cap \text{pf}(t) \neq \emptyset \)
- **Why is that true?**
  - Best case we are left with at most 5 matching elements beyond the elements in the sketch
Generalize to Weighted Sets

- Example with weighted vectors

\[ w(s \cap t) \geq \theta \quad (w(s \cap t) = \sum_{q \in s \cap t} w(q)) \]

- Sort by *weights* (not lexicographically anymore)

- Keep prefix \( pf(s) \) s.t. \( w[pf(s)] \geq w(s) - \alpha \)

\[
\begin{array}{cccccccc}
1' & 2' & 4' & 6' & 8' & 10' & 12' & 14'\\
s & w_1 & w_2 & 0 & 0 & w_5 & & & \\
t & 0 & w_2 & w_3 & 0 & w_5 & & & \\
\end{array}
\]

\[
\begin{array}{c}
\text{pf(s)}
\end{array}
\]

\[
\begin{array}{c}
w(s) - \alpha
\end{array}
\]

\[
\begin{array}{c}
\alpha
\end{array}
\]

\[
\sum_{q \in sf(s)} w(q) = \alpha
\]
Continued

- Best case: \( w[\text{sf}(s) \cap \text{sf}(t)] = \alpha \)
  - In other words, the suffixes match perfectly

- \( w(s \cap t) = w[\text{pf}(s) \cap \text{pf}(t)] + w[\text{sf}(s) \cap \text{sf}(t)] \)
  - Consider the prefix and the suffix separately

- \( w(s \cap t) \geq \theta \Rightarrow \)
  \[ w[\text{pf}(s) \cap \text{pf}(t)] + w[\text{sf}(s) \cap \text{sf}(t)] \geq \theta \]
  \[ w[\text{pf}(s) \cap \text{pf}(t)] \geq \theta - w[\text{sf}(s) \cap \text{sf}(t)] \]

- To avoid false negatives, minimize rhs
  \[ w[\text{pf}(s) \cap \text{pf}(t)] > \theta - \alpha \]
Properties

- $w[\text{pf}(s) \cap \text{pf}(t)] \geq \theta - \alpha$
- Hence $\theta \geq \alpha$
- Hence $\alpha = \theta_{\text{min}}$
- Small $\theta_{\text{min}} \Rightarrow$ long prefix $\Rightarrow$ large sketch

- For short strings, keep the whole string
- Prefix sketches easy to index
  - Use Inverted Index
How do I Choose $\alpha$?

- I need

$$|pf(s) \cap pf(t)| \neq \emptyset \Rightarrow$$

$$w[pf(s) \cap pf(t)] \geq 0 \Rightarrow$$

$$\theta = \alpha$$
Extend to Jaccard

- Jaccard(s, t) = \frac{w(s \cap t)}{w(s \cup t)} \geq \theta \Rightarrow \frac{w(s \cap t)}{w(s \cup t)} \geq \theta w(s \cup t)

- w(s \cup t) = w(s) + w(t) - w(s \cap t)

\Rightarrow \ldots

\Rightarrow \ldots

w[pf(s) \cap pf(t)] \geq \beta - w[sf(s) \cap sf(t)]

\beta = \frac{\theta}{1 + \theta} \cdot [w(s) + w(t)]

- To avoid false negatives:

\quad w[pf(s) \cap pf(t)] > \beta - \alpha
Technicability

\[ w[pf(s) \cap pf(t)] > \beta - \alpha \]

\[ \beta = \theta / (1 + \theta) [w(s) + w(t)] \]

- \( \beta \) depends on \( w(s) \), which is unknown at prefix construction time
- Use length filtering
  - \( \theta \ w(t) \leq w(s) \leq w(t) / \theta \)
Extend to Edit Distance

- Let string $s$ be a set of q-grams:
  - $s = \{q_{11}, q_{3}, q_{67}, q_{4}\}$
- Now the absolute position of q-grams matters:
  - Sort the set (e.g., lexicographically) but maintain positional information:
    - $s = \{(q_{3}, 2), (q_{4}, 4), (q_{11}, 1), (q_{67}, 3)\}$
  - Prefix sketch $\text{sig}(s) = \{(q_{3}, 2), (q_{4}, 4)\}$
Edit Distance Continued

- \( \text{ed}(s, t) \leq \theta \):
  - Length filter: \( \text{abs}(|s| - |t|) \leq \theta \)
  - Position filter: Common q-grams must have matching positions (within \( \pm \theta \))
  - Count filter: \( s \) and \( t \) must have at least
    \[
    \beta = [\max(|s|, |t|) - Q + 1] - Q \theta
    \]
    \( Q \)-grams in common
    \( s = \text{"Hello"} \) has 5-2+1 2-grams
  - One edit affects at most \( q \) q-grams
    "Hello" 1 edit affects at most 2 2-grams
Edit Distance Candidates

- Boils down to:
  1. Check the string lengths
  2. Check the positions of matching q-grams
  3. Check intersection size: $|s \cap t| \geq \beta$

- Very similar to Jaccard
Constructing the Prefix

- $|s \cap t| \geq \max(|s|, |t|) - q + 1 - q\theta$

- $|pf(s) \cap pf(t)| > \beta - \alpha$

\[ \beta = \max(|s|, |t|) - q + 1 - q\theta \]

A total of $(|s| - q + 1)$ q-grams
Choosing $\alpha$

\[ |pf(s) \cap pf(t)| > \beta - \alpha \]
\[ \beta = \max(|s|, |t|) - q + 1 - q\theta \]

- Set $\beta = \alpha$
  - $|pf(s)| \geq (|s|-q+1) - \alpha \Rightarrow$
    \[ |pf(s)| = q\theta+1 \text{ q-grams} \]
  - If $ed(s, t) \leq \theta$ then $pf(s) \cap pf(t) \neq \emptyset$
Pros/Cons

- Provides a loose bound
  - Too many candidates
- Makes sense if strings are long
- Easy to construct, easy to compare
Mismatch Filter

- When dealing with edit distance:
  - **Position of mismatching** q-grams within pf(s), pf(t) conveys a lot of information

- Example:
  - Clustered edits:
    - s = “submit by **Dec.**”
    - t = “submit by **Sep.**”

  - Non-clustered edits:
    - s = “**sabmit be Set.**”
    - t = “**submit by Sep.**”

  - 4 mismatching 2-grams
    - 2 edits can fix all of them

  - 6 mismatching 2-grams
    - Need 3 edits to fix them
Mismatch Filter Continued

- What is the minimum edit operations that cause the mismatching q-grams between s and t?
  - This number is a lower-bound on \( \text{ed}(s, t) \)
  - It is equal to the minimum edit operations it takes to **destroy** every mismatching q-gram
  - We can compute it using a greedy algorithm
  - We need to sort q-grams by position first (\( n \log n \))
Mismatch Condition

- Fourth edit distance pruning condition:
  4. Mismatched q-grams in prefixes must be destroyable with at most $\theta$ edits
Pros/Cons

- Much tighter bound
- Expensive (sorting), but prefixes relatively short
- Needs long prefixes to make a difference
Minhash

- So far we **sort** q-grams
  - What if we **hash** instead?

- **Minhash construction:**
  - Given a string $s = \{q_1, \ldots, q_m\}$
  - Use $k$ functions $h_1, \ldots, h_k$ from independent family of hash functions, $h_i: q \rightarrow [0, 1]$
  - Hash $s$, $k$ times and keep the $k$ q-grams $q$ that hash to the smallest value each time
  - $\text{sig}(s) = \{q_{mh1}, q_{mh2}, \ldots, q_{mhk}\}$
How to use minhash

● Example:
  ● $s = \{q_4, q_1, q_2, q_7\}$
  ● $h_1(s) = \{0.01, 0.87, 0.003, 0.562\}$
  ● $h_2(s) = \{0.23, 0.15, 0.93, 0.62\}$
  ● $\text{sig}(s) = \{0.003, 0.15\}$

● Given two sketches $\text{sig}(s), \text{sig}(t)$:
  ● $\text{Jaccard}(s, t)$ is the percentage of hash-values in $\text{sig}(s)$ and $\text{sig}(t)$ that match
  ● Probabilistic: $(\varepsilon, \delta)$-guarantees $\Rightarrow$ False negatives
Pros/Cons

- Has false negatives
- To drive errors down, sketch has to be pretty large
  - long strings
- Will give meaningful estimations only if actual similarity between two strings is large
  - good only for large $\theta$
PartEnum

- Lower bounds Hamming distance:
  - \( \text{Jaccard}(s, t) \geq \theta \Rightarrow H(s, t) \leq 2|s| \frac{(1 - \theta)}{(1 + \theta)} \)

- Partitioning strategy based on pigeonhole principle:
  - Express strings as vectors
  - Partition vectors into \( \theta + 1 \) partitions
  - If \( H(s, t) \leq \theta \) then at least one partition has hamming distance zero.
  - To boost accuracy create all combinations of possible partitions
Example

\[ \text{sig}(s) = h(\text{sig}_1) \cup h(\text{sig}_2) \cup \ldots \]
Pros/Cons

- Gives guarantees
- Fairly large sketch
- Hard to tune three parameters
  - Actual data affects performance
Compression
(BJL+09)
A Global Approach

- For disk resident lists:
  - Cost of disk I/O vs Decompression tradeoff
  - Integer compression
    - Golomb, Delta coding
  - Sorting based on non-integer weights??

- For main memory resident lists:
  - Lossless compression not useful
  - Design lossy schemes
Simple strategies

- Discard lists:
  - Random, Longest, Cost-based
  - Discarding lists tag-of-war:
    - Reduce candidates: ones that appear only in the discarded lists disappear
    - Increase candidates: Looser threshold $\theta$ to account for discarded lists

- Combine lists:
  - Find similar lists and keep only their union
Combining Lists

- Discovering candidates:
  - Lists with high Jaccard containment/similarity
  - Avoid multi-way Jaccard computation:
    - Use minhash to estimate Jaccard
    - Use LSH to discover clusters

- Combining:
  - Use cost-based algorithm based on query workload:
    - Size reduction
    - Query time reduction
    - When we meet both budgets we stop
General Observation

- V-grams, sketches and compression use the distribution of q-grams to optimize
  - Zipf distribution
  - A small number of lists are very long
  - Those lists are fairly unimportant in terms of string similarity
    - A q-gram is meaningless if it is contained in almost all strings
Selectivity Estimation for Selection Queries
The Problem

- Estimate the number of strings with:
  - Edit distance smaller than $\theta$
  - Cosine similarity higher than $\theta$
  - Jaccard, Hamming, etc…

- Issues:
  - Estimation accuracy
  - Size of estimator
  - Cost of estimation
Flavors

- Edit distance:
  - Based on clustering (JL05)
  - Based on min-hash (MBK+07)
  - Based on wild-card q-grams (LNS07)

- Cosine similarity:
  - Based on sampling (HYK+08)
Edit Distance

- Problem:
  - Given query string $s$
  - Estimate number of strings $t \in D$
  - Such that $ed(s, t) \leq \theta$
Clustering - Sepia

- Partition strings using clustering:
  - Enables pruning of whole clusters
- Store per cluster histograms:
  - Number of strings within edit distance 0, 1, …, \( \theta \) from the cluster center
- Compute global dataset statistics:
  - Use a training query set to compute frequency of data strings within edit distance 0, 1, …, \( \theta \) from each query
- Given query:
  - Use cluster centers, histograms and dataset statistics to estimate selectivity
Minhash - VSol

- We can use Minhash to:
  - Estimate $\text{Jaccard}(s, t) = \frac{|s \cap t|}{|s \cup t|}$
  - Estimate the size of a set $|s|$
  - Estimate the size of the union $|s \cup t|$
VSoI Estimator

- Construct one inverted list per q-gram in D and compute the minhash sketch of each list:

Inverted list

- Minhash
Selectivity Estimation

- Use edit distance count filter:
  - If \( ed(s, t) \leq \theta \), then \( s \) and \( t \) share at least
    \[
    \beta = \max(|s|, |t|) - q + 1 - q\theta
    \]
  - \( q \)-grams

- Given query \( t = \{q_1, \ldots, q_m\} \):
  - We have \( m \) inverted lists
  - Any string contained in the intersection of at least \( \beta \) of these lists passes the count filter
  - Answer is the size of the union of all non-empty \( \beta \)-intersections (there are \( m \) choose \( \beta \) intersections)
Example

- $\theta = 2$, $q = 3$, $|t| = 14 \Rightarrow \beta = 6$

- Look at all subsets of size 6

- $A = \left| \bigcup_{\{i_1, \ldots, i_6\} \in \binom{10}{6}} (t_{i_1} \cap t_{i_2} \cap \ldots \cap t_{i_6}) \right|$
The m-β Similarity

- We do not need to consider all subsets individually.
- There is a closed form estimation formula that uses minhash.
- Drawback:
  - Will overestimate results since many β-intersections result in duplicates.
OptEQ – wild-card q-grams

- Use extended q-grams:
  - Introduce wild-card symbol ‘?’
  - E.g., “ab?” can be:
    - “aba”, “abb”, “abc”, …

- Build an extended q-gram table:
  - Extract all 1-grams, 2-grams, …, q-grams
  - Generalize to extended 2-grams, …, q-grams
  - Maintain an extended q-grams/frequency hashtable
## Example

### Dataset

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td></td>
</tr>
<tr>
<td>sigmod</td>
<td>vldb</td>
</tr>
<tr>
<td>icde</td>
<td>...</td>
</tr>
</tbody>
</table>

### q-gram table

<table>
<thead>
<tr>
<th>q-gram</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab</td>
<td>10</td>
</tr>
<tr>
<td>bc</td>
<td>15</td>
</tr>
<tr>
<td>de</td>
<td>4</td>
</tr>
<tr>
<td>ef</td>
<td>1</td>
</tr>
<tr>
<td>gh</td>
<td>21</td>
</tr>
<tr>
<td>hi</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>?b</td>
<td>13</td>
</tr>
<tr>
<td>a?</td>
<td>17</td>
</tr>
<tr>
<td>?c</td>
<td>23</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>abc</td>
<td>5</td>
</tr>
<tr>
<td>def</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Assuming Replacements Only

- Given query q=“abcd”
- θ=2
- There are 6 base strings:
  - “??cd”, “?b?d”, “?bc?”, “a??d”, “a?c?”, “ab??”
- Query answer:
  - \( S_1 = \{ s \in D : s \in "??cd" \} \), \( S_2 = \{ s \in D : s \in "?b?d" \} \),
    \( S_3 = \{ \ldots \}, \ldots, S_6 = \{ \ldots \} \)
  - \( A = |S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5 \cup S_6| = \sum_{1 \leq n \leq 6} (-1)^{n-1} |S_1 \cap \ldots \cap S_n| \)
Replacement Intersection Lattice

\[ A = \sum_{1 \leq n \leq 6} (-1)^{n-1} |S_1 \cap \ldots \cap S_n| \]

- Need to evaluate size of all 2-intersections, 3-intersections, …, 6-intersections
- Use frequencies from q-gram table to compute sum A
- Exponential number of intersections
- But ... there is well-defined structure
Replacement Lattice

- Build replacement lattice:

  ![Replacement Lattice Diagram]

- Many intersections are empty
- Others produce the same results
  - we need to count everything only once

- 2 ‘?’
- 1 ‘?’
- 0 ‘?’
General Formulas

- Similar reasoning for:
  - r replacements
  - d deletions

- Other combinations difficult:
  - Multiple insertions
  - Combinations of insertions/replacements

- But … we can generate the corresponding lattice algorithmically!
  - Expensive but possible
Hashed Sampling

- Used to estimate selectivity of TF/IDF, BM25, DICE

- Main idea:
  - Take a sample of the inverted index
    - Simply answer the query on the sample and scale up the result
    - Has high variance
  - We can do better than that
Visual Example

Inverted list

Sampled Inverted lists

Answer the query using the sample and scale up
Construction

- Draw samples deterministically:
  - Use a hash function $h: \mathbb{N} \rightarrow [0, 100]$
  - Keep ids that hash to values smaller than $\phi$
  - This is called a bottom-K sketch

- Invariant:
  - If a given id is sampled in one list, it will always be sampled in all other lists that contain it
Example

- Any similarity function can be computed correctly using the sample
  - Not true for simple random sampling
Selectivity Estimation

- Any union of sampled lists is a $\varphi\%$ random sample
- Given query $t = \{q_1, \ldots, q_m\}$:
  - $A = \frac{|A_s| \cdot |q_1 \cup \ldots \cup q_m|}{|q_{s1} \cup \ldots \cup q_{sm}|}$:
    - $A_s$ is the query answer size from the sample
    - The fraction is the actual scale-up factor
    - But there are duplicates in these unions!
- We need to know:
  - The distinct number of ids in $q_1 \cup \ldots \cup q_m$
  - The distinct number of ids in $q_{s1} \cup \ldots \cup q_{sm}$
Count Distinct

- Distinct $|q_{s1} \cup \ldots \cup q_{sm}|$ is easy:
  - Scan the sampled lists

- Distinct $|q_1 \cup \ldots \cup q_m|$ is hard:
  - Scanning the lists is the same as computing the exact answer to the query ... naively
  - We are lucky:
    - Each sampled list doubles up as a bottom-k sketch by construction!
    - We can use the list samples to estimate the distinct $|q_1 \cup \ldots \cup q_m|$
The Bottom-k Sketch

- It is used to estimate the distinct size of arbitrary set unions (the same as FM sketch):
  - Take hash function $h: \mathbb{N} \rightarrow [0, 100]$
  - Hash each element of the set
  - The $r$-th smallest hash value is an unbiased estimator of count distinct:

\[ h_r \]

\[ r \]

\[ 0 \]

\[ h_r \]

\[ 100 \]

\[ ? \]
Transformations/Synonyms (ACGK08)
Transformations

- No similarity function can be cognizant of domain-dependent variations
- Transformation rules should be provided using a declarative framework
  - We derive different rules for different domains
    - Addresses, names, affiliations, etc.
  - Rules have knowledge of internal structure
    - Address: Department, School, Road, City, State
    - Name: Prefix, First name, Middle name, Last name
Observations

- Variations are orthogonal to each other
  - Dept. of Computer Science, Stanford University, California
  - We can combine any variation of the three components and get the same affiliation

- Variations have general structure
  - We can use a simple generative rule to generate variations of all addresses

- Variations are specific to a particular entity
  - California, CA
  - Need to incorporate external knowledge
Augmented Generative Grammar

- G: A set of rules, predicates, actions

<table>
<thead>
<tr>
<th>Rule</th>
<th>Predicate</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;name&gt; → &lt;prefix&gt; &lt;first&gt; &lt;middle&gt; &lt;last&gt;</td>
<td></td>
<td>first; last</td>
</tr>
<tr>
<td>&lt;name&gt; → &lt;last&gt;, &lt;first&gt; &lt;middle&gt;</td>
<td></td>
<td>first; last</td>
</tr>
<tr>
<td>&lt;first&gt; → &lt;letter&gt;.</td>
<td></td>
<td>letter</td>
</tr>
<tr>
<td>&lt;first&gt; → F</td>
<td>F in Fnames</td>
<td>F</td>
</tr>
</tbody>
</table>

- Variable F ranges over a fixed set of values
  - For example all names in the database
- Given an input record r and G we derive “clean” variations of r (might be many)
Efficiency

- We do not need to generate and store all record transformations
  - Generate a combined sketch for all variations (the union of sketches)
  - Transformations have high overlap, hence the sketch will be small
- Generate a derived grammar on the fly
  - Replace all variables with constants from the database that are related to r (e.g., they are substrings of r or similar to r)
Conclusion
Conclusion

- Approximate selection queries have very important applications
- Not supported very well in current systems (think of Google Suggest)
- Work on approximate selections has matured greatly within the past 5 years
- Expect wide adoption soon!
Thank you!
References

- [BJL+09] Space-Constrained Gram-Based Indexing for Efficient Approximate String Search. Alexander Behm, Shengyue Ji, Chen Li, and Jiaheng Lu. ICDE 2009
- [CGK06] A Primitive Operator for Similarity Joins in Data Cleaning. Surajit Chaudhuri, Venkatesh Ganti, Raghav Kaushik. ICDE06
- [CCGX08] An Efficient Filter for Approximate Membership Checking. Kaushik Chakrabarti, Surajit Chaudhuri, Venkatesh Ganti, Dong Xin. SIGMOD08
References

- [JL05] Selectivity Estimation for Fuzzy String Predicates in Large Data Sets. Liang Jin, Chen Li. VLDB 2005.
- [JLL+09] Efficient Interactive Fuzzy Keyword Search. Shengyue Ji, Guoliang Li, Chen Li, and Jianhua Feng. WWW 2009
- [JLV08] SEPIA: Estimating Selectivities of Approximate String Predicates in Large Databases. Liang Jin, Chen Li, Rares Vernica. VLDBJ08
- [LNS07] Extending Q-Grams to Estimate Selectivity of String Matching with Low Edit Distance. Hongrae Lee, Raymond T. Ng, Kyuseok Shim. VLDB 2007
References

- [SK04] Efficient set joins on similarity predicates. Sunita Sarawagi, Alok Kirpal. SIGMOD 2004
- [XWL+08] Efficient similarity joins for near duplicate detection. Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu. WWW 2008
- [YWL08] Cost-Based Variable-Length-Gram Selection for String Collections to Support Approximate Queries Efficiently. Xiaochun Yang, Bin Wang, and Chen Li. SIGMOD 2008