Similarity Query Processing Algorithms: Use of Enumeration and Divide and Conquer Techniques

Wei Wang
University of New South Wales
Australia
http://www.cse.unsw.edu.au/~weiw
Roadmap

A. Overview of our Works
B. Similarity queries - Motivations
C. Basic techniques based on enumeration
D. Variations of enumeration
E. Conclusions
Research Areas

- Similarity query processing
- Keyword search on (semi-) structured data
  - SPARK
  - XML Keyword Search
- High-dimensional indexing
  - LSH
Major Work in Similarity Query Processing

- Similarity search for Sets and Vectors
  - Jaccard / cosine / Dice [WWW08, SIGMOD10, TODS11]
  - Hamming [SSDBM13]

- Similarity search for Strings
  - Edit distance [PVLDB08, SIGMOD11, TODS11, TKDE12, PVLDB13, TODS13]

- Similarity search with Rules
  - Rule mining [DEXA11], similarity definition and execution [SIGMOD13]

- Similarity search for Graphs
  - Containment/sub/super-graph search, graph similarity search, etc [SIGMOD10, SSDBM10, DASFAA10, ICDE12, VLDBJ13]

- Application: large scale cross document coreference resolution (CDCR)
Roadmap

A. Overview of our Works
B. Similarity queries - Motivations
C. Basic techniques based on enumeration
D. Variations of enumeration
E. Conclusions
App1: Fixing Small Errors

● Typographical errors
  • Person’s names
  • Web queries

● OCR errors
  • 13 vs B

● Lack of consistency
  • tf-idf, tf.idf, tf*idf
Try their names (good luck!)

UCSD
Yannis Papakonstantinou

Case Western
Meral Ozsoyoglu

AT&T--Research
Marios Hadjieleftheriou

http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/index.html
Source: Hadjieleftheriou & Li, VLDB09 tutorial
Better system?

http://dblp.ics.uci.edu/authors/
4. **Efficient Approximate Search on String Collections (Tutorial)**, Marios Hadjieleftheriou and Chen Li, VLDB 2009. [PDF], [Part I], [Part II].
5. **Efficient Approximate Search on String Collections (Tutorial)**, Marios Hadjieleftheriou, Chen Li, ICDE 2009, [PPT-Part1], [PPT-part2].
6. **Quality-Aware Retrieval of Data Objects from Autonomous Sources for Web-Based Repositories**, Houtan Shirani-Mehr, Chen Li, Gang Liang, Michal Shmueli-Scheuer, ICDE 2008 (poster). [PDF]
7. **Communication-Efficient Query Answering with Quality Guarantees in Client-Server**
App2: Image & Video Dedup

- Semantically equivalent objects

A photo and its digitally modified version are bit-wise different!
Similarity Search

- The solution
  - Represent objects in a digital format
    - Typically each object represented as a set/vector/sequence of features
  - Define a similarity function between objects’ featurized representation
    - \( \text{sim}(x, y) \) in \([0, 1]\), or define a distance function
  - Similarity query
    - Find all objects in the DB such that their similarities with the query is no less than a threshold

Also many applications in other areas (e.g., machine learning, bioinformatics, etc.)
Problem Definition: Similarity Search

- **Input**
  - a set of objects: $R$
  - a query object: $q$
  - a similarity function: $\text{sim}(r, q)$
  - a threshold: $t$

- **Output**
  - All objects $r \in R$, such that $\text{sim}(r, q) \geq t$

- **Variations**
  - $\text{dist}(r, q) \leq d$
Hamming Distance Search

- Object similarity search $\Rightarrow$ Hamming distance search on (binary) vectors
  - Manually defined/extracted features:
    - E.g., Google’s image search, fingerprints of chemical compounds
  - LSH
    - Minhash (shingling), p-stable, simhash
  - Learned hash functions
    - Similarity preserving hash functions [Norouzi and Fleet, ICML11] [Zhang et al, SIGIR10] [Zhang et al, SIGIR12]

- Other types of similarity search $\Rightarrow$ Hamming distance search
  
  $$J(x, y) \geq t \iff H(x, y) \leq \frac{1 - t}{1 + t} \cdot (|x| + |y|)$$
Image Search

mapping

similar?

Object

104-dim vector

Object

104-dim vector

dist() < ε

nokia n8

获得约 2,270,000 条结果（用时 0.03 秒）

Nokia N8
315 x 337 - 31k - jpg
techknowbits.com
查找相似图片

NOKIA N8
450 x 401 - 37k - jpg
gsm-mind.blogspot.com
查找相似图片

Post image for
600 x 523 - 51k - jpg
latestngadgets.com
查找相似图片

Nokia 早前公布
480 x 468 - 70k - jpg
tieao.com

Nokia N8

Nokia N8

Nokia N8

Nokia N8

标签： 诺基亚N8 nokia
Google’s Image Clustering [Liu, Rosenberg & Rowley, WACV07]

- Use MR + Spill Tree for kNN search in a feature space
- **Image features = 104-dim real vectors**
  - Normalize color intensities & picture size (to 64 x 64)
  - Extract and quantize Haar wavelet features
    - Quantize largest 60 coefficients to +/- 1
    - Others ➞ 0
  - Dimensionality reduction
    - 64 * 64 * 3-dim binary vector ➞ 100-dim vector via random projection
  - Add avg color values + picture aspect ratio
- k-NN search using (probably) $L_2$ distance
Sentence Reuse Detection [Zhang, Wu, Ding & Wang, SIGIR12]

- **Sig = 32-dim binary vectors**
  - sig(sentence) = OR(sig(word₁), sig(word₂), ...)
  - sig(word) are learned from a training corpus via integer linear programming

- **Query processing**
  - Cand-sentences = d-query with Hamming distance (d in [0, 5])
  - Post-processing to verify the candidates
  - d in [2, 4] to achieve a good recall (≥ 90%)
Roadmap

A. Overview of our Works
B. Similarity queries - Motivations
C. Basic techniques based on enumeration
D. Variations of enumeration
E. Conclusions
Variants & Enumeration on Query

- Vectors (of N dimensions) = $\sum^N$ (String = $\sum^*$)
- $k$-query (Hamming distance)
  - Finding vectors which differ in at most $k$ dimension with the query vector $Q$
- $k$-variants($V$) = \{ $V'$ $\in$ $\sum^N$ | Hamming($V$, $V'$) $\leq$ $k$ \}
- Example: $\sum$ = \{0, 1, 2\}
  - $1$-variants(000) = \{000, 100, 200, 010, 020, 001, 002\}
Enumeration on Data

- Generate and index all the 1-variants for each data vector

\[ N=3, \ k=1 \]

**Q:**

<table>
<thead>
<tr>
<th>000</th>
<th>v1, v3</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>v1, v2, v4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Naïve Enumeration

- Naïve enumeration on query
  - Index all data strings as they are in I
  - For each Q' in k-variants(Q), return I[Q']

- Naïve enumeration on data
  - Index all strings in k-variants (S_i) in I
  - Return I[Q]

- Comment
  - Super-fast query processing time when k is small
  - May result in super-linear index size

Nothing can beat the O(1+occ) algorithm !!!

The $|\Sigma|^k$ factor can be dropped by using the deletion variants

1-query can be efficiently solved !!!
Dealing with large error thresholds

- Problem with naïve enumeration
  - Cannot deal with large $k$
  - Cannot deal with large $|\Sigma|$

- Idea 1: Divide and conquer (or partitioning)
  - One level partitioning
  - Two level partitioning

- Idea 2: Use deletion-variants
Enumeration using Deletion Variants

- $k$-del-variants($V$) = \{all $V'$ generated by substituting $k$ dimensions with ‘#’\}
- Symmetric generation of deletion variants for both data and query

Space = $O(n^k N^k)$
Time = $O(N^k + \text{occ})$

Example:

$N=3$, $k=1$

- 1-del-variant(001)
- 1-del-variant(021)
- 1-del-variant(100)
- 1-del-variant(201)

Q:

```
0 0 0
# 0 0
0 # 0
0 0 #
```

$\Rightarrow v_1, v_3$

```
#00  \Rightarrow v_1, v_3
0#1  \Rightarrow v_1, v_2, v_4
00#  \Rightarrow v_1
001  \Rightarrow v_1
```
Google’s Method [Manku, Jain and Sarma, WWW07]

- **Background**
  - $n$ docs mapped to sketches of $N$-bits each (using a heuristic implementation of simhash [Charikar, STOC02])
  - given a new document, generate its sketch $Q$
  - need to return all sketches $V_i$ that has Hamming distance at most $k$ from $Q$
  - $N = 64$ and $k = 3$ in the paper

- **Naïve solutions**
  - Enum on Query
    - too many queries
  - Enum on Data
    - too much space

Exp: 
$$
\binom{64}{3} \times 7 + \binom{64}{2} \times 3 + \binom{64}{1} \times 1 + 1 = 297761
$$
Google’s 1 Level Method [Manku, Jain and Sarma, WWW07]

- if $V$ is an answer, $V$ and $Q$ differ by at most $k$ bits
  - but these $k$ bits can be anywhere within the $N$ dimensions

\[ \text{solution: partition} \]

\[ N=6, \ k=2 \]

\[ \begin{array}{cccccc}
Q & 1 & 1 & 0 & 1 & 0 & 0 \\
V_1 & 1 & 1 & 0 & 1 & ? & ? \\
V_2 & 1 & ? & 0 & 1 & ? & 0 \\
\vdots
\end{array} \]

\[ \begin{array}{cccccc}
1 & 1 & 0 & 1 & 0 & 0 \\
\end{array} \]

\[ \begin{array}{cccccc}
\end{array} \]

\[ \begin{array}{cccccc}
\end{array} \]

\[ \begin{array}{cccccc}
? & ? & ? & ? & 0 & 0 \\
\end{array} \]

Form 3 partitions

How many partitions are preserved by any $V_i$?

\[ \binom{3}{1} = 3 \]

\[ \binom{6}{2} = 15 \]

\[ \begin{array}{c}
\text{Cand}_1 = \{a, \ldots\} \\
\text{Cand}_2 = \{x, \ldots\} \\
\text{Cand}_3 = \{m, \ldots\}
\end{array} \]
Further Details

- Requires further verification after union’ing the candidates
- How to find Cand₂?
  - Replicate vectors with dim3 & dim 4 permuted to the beginning
  - Do binary search
- Candᵢ size ≈ n / (|∑|²)

Form 3 partitions

At least 1 partition is preserved by any Vᵢ

Elements in ∪ᵢ Cᵢ need further verification
PartEnum [Arasu, Ganti and Kaushik, VLDB06]

• **Part + Part + Enum**

| N=9, k=5 | form 3 partitions |
| 1 0 1 0 1 1 0 1 0 |

**At least one partition has error ≤ \[\lfloor k/3 \rfloor = 1\]** ← **Pigeon hole principle**

**Part** (n1=3 partitions)

- Each record generates \( n1 \left( \left\lfloor \frac{n2}{k/n1} \right\rfloor \right) \) signatures

- Hamming(u, v) ≤ k \( \Rightarrow \) sigs(u) \( \cap \) sigs(v) \( \neq \phi \)

**Enum** (n2=2 partitions)

- Hard to tune the parameters
- Seems only competitive (for ed) when k=1
## Google’s 2 Level Method [Manku et al, WWW07]

- **Trade space for time!**

<table>
<thead>
<tr>
<th>N=9, k=2</th>
<th>Form 3 partitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 1 0 1 0</td>
<td>1 0 1 0 1 0 1 0</td>
</tr>
</tbody>
</table>

**At least 1 partition is preserved**

<table>
<thead>
<tr>
<th>Q</th>
<th>Cand 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 1 0 1 0</td>
<td>Cand 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cand 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 1 0 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cand 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0 1 0 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cand 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 1 0 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cand 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 1 0 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cand 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 1 0 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cand 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 1 0 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cand 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 1 0 1 0</td>
</tr>
</tbody>
</table>

2 errors in the remaining dimensions!

\[ \begin{pmatrix} 3 \\ 1 \end{pmatrix} \times \begin{pmatrix} 3 \\ 1 \end{pmatrix} \]
HEngine [Liu, Shen and Torng, ICDE11]

- Ideas:
  - Reduce d-query to multiple d’-query, where $1 \leq d' < d$
    - Essentially only $d' = 1$ is practical
    - Fewer # of replications by substitution
  - Can be deemed as a special case of PartEnum which always partitions into $\lceil (k+2)/2 \rceil$ partitions (hence at least one partition with at most 1 error).
HEngine Example

- Replicate DB 2 times rather than 2*3 times (as does Google’s method)
- Substitute ? in the prefix with chars from \( \sum \) \( \Rightarrow \) range queries on \( DB_i \)
HmSearch [Zhang et al, SSDBM13]

- Observation 1
  - Reduce d-query to 1-queries
  - Index data’s 1-variants or 1-deletion variants to answer 1-queries
  - Hence, can handle large $|\Sigma|$
**HmSearch** [Zhang et al, SSDBM13]

- **Observation 2**
  - HEngine results in 2 partitions for both $k=2$ and $3$; and the rest of the query processing is the same!

- **Idea**
  - When $k = 2$, the errors made by a candidate must be $(1,1)$ or $(0,2)$ ➔ In either case, the candidate must be returned by at least two matching “variants”
  - (1) Partition into $\lceil (k+3)/2 \rceil$ partitions; (2) Enhanced filtering condition for odd $k$ and also based on exact or error-1-match
  - Benefit: can handle larger amount of errors than existing approaches
Other Optimization in HmSearch

- Filtering-as-verification based on hierarchical binary representation
  - Effective filtering for free (i.e., never degrade performance)
- Dimension reordering
  - Better accommodates data skew
HmSearch Example

- $V_1$ is not a candidate
  - Only common 1-deletion variant is “45#”

- $V_2$ shares 3 common 1-deletion variants with Q’s
  - “45#”, “57”, and “47”

- $V_2$ will accumulate two errors by performing hierarchical verification to the second least significant bit
HmSearch Experiments

- **Datasets**
  - Audio: 64 dims, $|\Sigma| = 16$, generated by 2-stable LSH functions, $n = 54,387$
  - PubChem: 881 dims, $|\Sigma| = 2$, generated by some fingerprinting algorithm, $n = 1,000,000$

- **Algorithms**
  - Google: $k+1$ partitions, indexing the partitions
  - HEngine: $\lceil (k+2)/2 \rceil$ partitions, replicate the data
  - ScanCount: Index every dimension values and perform merge
  - HSV/HSD: HmSearch with indexed 1-variants/1-del variants
Average Query Time

- HSV < HSD < others
- When $k$ is small, Google < HEngine
- When $k$ is large, Google > Hengine; eventually both > ScanCount

![Graphs showing average query time vs. Hamming Distance for different methods: HSV, HSD, Google, HEngine, and ScanCount.](image)
Dimension Reordering

- Substantial impact on PubChem; little on Audio (LSH)
## Summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Index Size, ( O() )</th>
<th>Query Time, ( O() )</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-variants on Data</td>
<td>( n \times N \times N^k \times</td>
<td>\Sigma</td>
</tr>
<tr>
<td>k-variants on Query</td>
<td>( n \times N )</td>
<td>( N^k \times</td>
</tr>
<tr>
<td>k-deletion variants</td>
<td>( n \times N \times N^k )</td>
<td>( N^k + \text{VT} )</td>
</tr>
<tr>
<td>Google’s 1-level</td>
<td>( n \times N \times k )</td>
<td>( k \times \log(n) + \text{VT} )</td>
</tr>
<tr>
<td>PartEnum</td>
<td>( n \times k^{2.39} )</td>
<td>( k^{2.39} + \text{VT} )</td>
</tr>
<tr>
<td>Google’s 2-level</td>
<td>( n \times N \times k^2 )</td>
<td>( k^2 \times \log(n) + \text{VT} )</td>
</tr>
<tr>
<td>HEngine</td>
<td>( n \times N \times k )</td>
<td>( N \times</td>
</tr>
<tr>
<td>HmSearch+1-variants</td>
<td>( n \times N \times k \times</td>
<td>\Sigma</td>
</tr>
<tr>
<td>HmSearch+1-del-variants</td>
<td>( n \times N \times k )</td>
<td>( N + \text{VT} )</td>
</tr>
</tbody>
</table>

- \( n \) vectors in DB; \( N = \#\text{-dims} \); VTs are all different
Roadmap

A. Overview of our Works
B. Similarity queries - Motivations
C. Basic techniques based on enumeration
D. Variations of enumeration
E. Conclusions
Other Variations

1. Reduced alphabet variant generation
2. Extension to edit distance function
3. Adaptive enumeration
4. Space efficient representation
5. Truncated enumeration
1. Reduced Alphabet 1-Variants

- Idea: \( \Sigma \rightarrow \Sigma' \)
- 1-deletion variant is a special case where \(|\Sigma'| = 1\)
- Example when \(|\Sigma'| = 2\)
  - \( \Sigma = [a-z] \)
  - \( \Sigma' = [01] \), where [aeiou] \( \rightarrow 0 \) and others \( \rightarrow 1 \)
  - \( S = \text{"the"} \rightarrow S' = \text{"110"} \)
  - 1-variants(\( S' \)) = \{110, 010, 100, 111\)
2. Deletion Variants for Edit Distance

Overlap threshold = 1

Works very well for short strings and \( d = 1 \) as complexity is \( O(|S|^d) \)

FastSS Algorithm
[Bocek, Hunt & Stiller, 2007]
Overlap threshold = 1
Works for long strings and small $d$ as complexity is $O(C \times d^2)$

NGPP [Wang et al, SIGMOD 09]

$d = 2$

Partition into $\lceil (d+1)/2 \rceil$ partitions

1-deletion variants

Shifting + Scaling

{abxd, abx, abxdefghi, efghi, ddefghi}
3. Adaptive Enumeration [Xiao et al, PVLDB 13]

- **IncNGTrie** [Xiao et al, PVLDB 13]

- **Ideas**
  - Index all the variants of all data strings in a trie
  - Perform only exact-match or err during the query processing
    - i.e., follow-the-next-query-char or follow-#

- **Benefits:**
  - Only “necessary” enumerations were performed
Example

- Q = task, k = 1
- DFS in actual implementation

Example

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>(1, 0)</td>
</tr>
<tr>
<td>t</td>
<td>(2,0), (17, 1)</td>
</tr>
<tr>
<td>ta</td>
<td>(12, 1)</td>
</tr>
<tr>
<td>tas</td>
<td>(13, 1)</td>
</tr>
<tr>
<td>task</td>
<td>∅</td>
</tr>
</tbody>
</table>

Effectively, only 2 (instead of 4) 1-deletion variants of the query are ever enumerated.

Figure 1: Example of IncNGTrie ($s_1 = \text{test, } s_2 = \text{text}$)
Experimental Results (for Edit Distance)

- Up to 1000x speed-up against previous methods
- Does not degrade with $|\Sigma|$
4. Space Efficient Enumeration
[Boytsov, SISAP12]

- **Space complexity**
  - String $|S| \Rightarrow |S|$ 1-deletion variants of length $|S|$
  - If total data strings size = $n^*L$, then their 1-deletion variants occupies $n^*L^*L$ space

- **Solution 1:**
  - $\text{var}(S) \Rightarrow \text{hash}(\text{var}(S))$
  - No false negatives, but with possible false positives
  - Works well for filtering-based approaches, as they perform (naïve) verification anyway
Using Perfect Hash Function

Solution 2:

- Observation: \( \text{var}(S) \) and \( S \) only different by 1 dimension
- Record such dimension number \( \Delta(S) \), and the original char \( S[\Delta(S)] \) \( \Rightarrow \) \( S \) can be reconstructed from \( \text{var}(S) \), \( \Delta(S) \), \( S[\Delta(S)] \)
- Hash \( \text{var}(S) \) \( \Rightarrow \) \(<\text{hash}(\text{var}(S)), \Delta(S), S[\Delta(S)]>\)
- To eliminate false positive due to hashing, reconstruct \( S' \) from \( \Delta(S) \), \( S[\Delta(S)] \), and check if \( S' \) is indeed a string in the database
  - Requires \text{hash()} to be perfect hash function
5. Truncated Enumeration

- The length L prefix of a vector v is $v_{[L]}$

- Necessary condition for $H(v, Q) \leq k$ is $H(v_{[L]}, Q_{[L]}) \leq k$, which entails $\text{var}(v_{[L]}, Q_{[L]}) \leq k$ [Wang, Xiao, Lin & Zhang, SIGMOD09] [Xiao et al, PVLDB13]
  - Benefit: only need to enumerate up to L (rather than |v|)

- Can be applied recursively
  - E.g., take another prefix at L-k [Bast & Celikik, TOIS13]

- Can be generalized to edit distance search/joins [Wang, Xiao, Lin & Zhang, SIGMOD09] [Xiao et al, PVLDB13] [Bast & Celikik, TOIS13]
Roadmap

A. Overview of our Works
B. Similarity queries - Motivations
C. Basic techniques based on enumeration
D. Variations of enumeration
E. Conclusions
Conclusions

- Similarity queries are fundamental to many applications
- Efficient algorithms available for many practical cases
- Illustrates several works centered around the techniques of
  - Enumeration
  - Divide-and-conquer / partitioning
- Very fast algorithms can be obtained
  - At the cost of super-linear index size
Q & A

Our Similarity Query Processing Project Homepage:
http://www.cse.unsw.edu.au/~weiw/project/simjoin.html

Ad: ICDE2014 “Strings, Texts and Keyword Search” track
References

- [Arasu, Ganti and Kaushik, VLDB06]
  - Arvind Arasu, Venkatesh Ganti, Raghav Kaushik: Efficient Exact Set-Similarity Joins. VLDB 2006: 918-929

- [Bast & Celikik, TOIS13]

- [Boytsov, SISAP12]

- [Liu, Rosenberg & Rowley, WACV07]
References

- [Liu, Shen and Torng, ICDE11]
  - Alex X. Liu, Ke Shen, Eric Torng: Large scale Hamming distance query processing. ICDE 2011: 553-564

- [Manku, Jain and Sarma, WWW07]
  - Gurmeet Singh Manku, Arvind Jain, Anish Das Sarma: Detecting near-duplicates for web crawling. WWW 2007: 141-150

- [Norouzi and Fleet, ICML11]

- [Wang, Xiao, Lin & Zhang, SIGMOD09]
References

- [Xiao et al, PVLDB13]

- [Zhang et al, SSDBM13]
  - Xiaoyang Zhang, Jianbin Qin, Wei Wang, Yifang Sun, Jiaheng Lu. HmSearch: An Efficient Hamming Distance Query Processing Algorithm. SSDBM 2013.

- [Zhang et al, SIGIR10]
  - Qi Zhang, Yue Zhang, Haomin Yu, Xuanjing Huang: Efficient partial-duplicate detection based on sequence matching. SIGIR 2010: 675-682

- [Zhang, Wu, Ding & Wang, SIGIR12]
  - Qi Zhang, Yan Wu, Zhuoye Ding, Xuanjing Huang: Learning hash codes for efficient content reuse detection. SIGIR 2012: 405-414