Retrieval-Technologien für die Plagiaterkennung in Programmen

Fabian Loose, Steffen Becker, Martin Potthast, Benno Stein @ webis.de
Bauhaus University Weimar

Outline
· Overview
· Retrieval Models for Source Code
· Hash-based Search
Overview
Overview

Plagiarism is the practice of claiming, or implying, original authorship of someone else’s written or creative work, in whole or in part, into one’s own without adequate acknowledgment.  

[Wikipedia: Plagiarism]

- Plagiarism is observed in literature, music, software, scientific articles, newspaper, advertisement, Web sites, etc.
- A study among 18,000 university students in the United States shows that almost 40% of them have plagiarized at least once. [1]

Overview

Taxonomy of Plagiarism Offenses

Plagiarism offence

Detection method

Accurate copy

Identity analysis

Modified copy

Language translation

Structure analysis

Transformation

Similarity analysis

Large part of document

Global identity analysis: Document model comparison (suffix-tree)

Small part of document

Local identity analysis

Large part of document

Global analysis: Document model comparison (VSM)

Small part of document

Local similarity analysis

with reference corpus:

Chunk identity (MD5-Hash)

w/o reference corpus:

Style analysis

with reference corpus:

Fuzzy-fingerprinting

w/o reference corpus:

Style analysis
Paragraph detection
Paragraph detection
External analysis
Heuristic retrieval (focused search)

Suspicious paragraphs

Knowledge-based post processing
Paragraph detection

External analysis

Heuristic retrieval (focused search)

- Web Keyword index
- Keyword extraction
- Keyword-based retrieval
- Candidate documents
- Chunking

Comparison

- Fingerprint-based comparison
- Pairwise VSM comparison

Retrieval with precompiled index

- Wikipedia fingerprint index
- Chunking & Fingerprinting
- Fingerprint-based retrieval

Suspicious paragraphs

Knowledge-based post processing
Paragraph detection

External analysis

Heuristic retrieval (focused search)

Web Keyword index

Keyword extraction

Keyword-based retrieval

Candidate documents

Chunking

Comparison

Fingerprint-based comparison

Pairwise VSM comparison

Retrieval with precompiled index

Wikipedia fingerprint index

Chunking & Fingerprinting

Fingerprint-based retrieval

Intrinsic analysis

Suspicious paragraphs

Knowledge-based post processing
Paragraph detection

Cross language external analysis

Heuristic retrieval (focused search)

Web Keyword index

Keyword extraction

Keyword-based retrieval

Candidate documents

Chunking

Comparison

Fingerprint-based comparison

Pairwise VSM comparison

Retrieval with precompiled index

Wikipedia fingerprint index

Chunking & Fingerprinting

Fingerprint-based retrieval

Intrinsic analysis

Suspicious paragraphs

Knowledge-based post processing
Overview

Examples for Identification Technology

- **Level 1. Identity analysis for paragraphs.**
  - MD5 hashing

- **Level 2. Synchronized identity analysis for paragraphs.**
  - hashed breakpoint chunking

- **Level 3. Tolerant similarity analysis for paragraphs.**
  - Fuzzy-fingerprinting

- **Level 4. Intrinsic (style) analysis without a reference corpus.**
  - statistical outlier analysis with Bayes, meta learning with logistic regression

- **Level 5. Correct citation.**
  - knowledge-based analysis
Overview

Current research is corpus-centered, “external plagiarism analysis”.


External plagiarism analysis formulated as decision problem:

**Problem.** AV\textsc{EXTERN} (AV stands for Authorship Verification)

**Given.** A text $d$, allegedly written by author $A$, and set of texts $D$, $D = \{d_1, \ldots, d_n\}$, written by an arbitrary number of authors.

**Question.** Does $d$ contain sections whose similarity to sections in $D$ is above a threshold $\theta$?
Overview

Basic Principle

- Partition each document in meaningful sections, also called “chunks”.
- Do a pairwise comparison using a similarity function $\varphi$.

Complexity:

$n$ documents in corpus, $c$ chunks per document on average

$\Rightarrow O(n \cdot c^2)$ comparisons
Overview

Comparison with Fingerprints (Level 1)

- Partition each document into equidistant sections.
- Compute fingerprints of the chunks using a hash function $h$.
- Put all hashes into a hash table. A collision indicates matching chunks.

Complexity:

$n$ documents in corpus, $c$ chunks per document on average

$O(n \cdot c)$ operations (fingerprint generation, hash table operations)
Overview

Comparison with Fingerprints (Level 2)

- Partition each document into \textit{synchronized} sections.
- Compute fingerprints of the chunks using a hash function \( h \).
- Put all hashes into a hash table. A collision indicates matching chunks.

Complexity:

\( n \) documents in corpus, \( c \) chunks per document on average

\[ O(n \cdot c) \] operations (fingerprint generation, hash table operations)
Overview

Comparison with Fingerprints (Level 3)

Discussion:

- Hashing is fast, but sensitive to smallest changes:
  \[ h(c_1) = h(c_2) \Rightarrow c_1 = c_2 \]  
  (with very high probability)

Current research:

- Focus on fuzzy hash functions \( h_\varphi \):
  \[ h_\varphi(c_1) = h_\varphi(c_2) \Rightarrow P(\varphi(c_1, c_2) > \theta) \geq 1 - \varepsilon \]  
  [Stein 2005-07]

- Fuzzy hash functions allow for large chunk sizes (speed-up)
- Fuzzy hash functions are not sensitive to small changes
Retrieval Models for Source Code
Retrieval Models for Source Code

```java
//subloop. for each node...
for (int nodeIndex = 0; nodeIndex < n; nodeIndex++) {
    int nodeId = nodeIdPermutation[nodeIndex];
    //System.out.println("node: "+nodeId);

    //reset sums.
    for (int i = 0; i < n; i++) sumOfEdgeWeights[i] = 0;

    //sum all the edges going out to the same cluster
    int[] adjacentNodes = graph.getAdjacentNodes(nodeId);
    for (int i : adjacentNodes) {
        int clusterId = nodes2cluster[i];
        double edgeWeight = graph.getEdgeWeight(nodeId, i);
        if (edgeWeight >= threshold) {
            sumOfEdgeWeights[clusterId] += edgeWeight;
        }
    }

    //and determine the cluster of biggest sum.
    int newClusterNumber = nodes2cluster[nodeId];
    double maxWeight = 0;
    for (int i = 0; i < sumOfEdgeWeights.length; i++) {
        if ((sumOfEdgeWeights[i]) > maxWeight) {
            newClusterNumber = i;
            maxWeight = sumOfEdgeWeights[i];
        }
    }
    ...
```

| Representation | Sim. measure $\varphi$ | Compilation level for $d$ | Runtime for $\varphi$ |
Retrieval Models for Source Code

Structure-based Graph Models

//subloop. for each node...

for (int nodeIndex = 0; nodeIndex < n; nodeIndex++) {
    int nodeId = nodeIdPermutation[nodeIndex];
    //System.out.println("node: "+nodeId);

    //reset sums.
    for (int i = 0; i < n; i++) sumOfEdgeWeights[i] = 0;

    //sum all the edges going out to the same cluster
    int[] adjacentNodes = graph.getAdjacentNodes(nodeId);
    for (int i : adjacentNodes) {
        int clusterId = nodes2cluster[i];
        double edgeWeight = graph.getEdgeWeight(nodeId, i);
        if (edgeWeight >= threshold) {
            sumOfEdgeWeights[clusterId] += edgeWeight;
        }
    }

    //and determine the cluster of biggest sum.
    int newClusterNumber = nodes2cluster[nodeId];
    double maxWeight = 0;
    for (int i = 0; i < sumOfEdgeWeights.length; i++) {
        if ((sumOfEdgeWeights[i]) > maxWeight) {
            newClusterNumber = i;
            maxWeight = sumOfEdgeWeights[i];
        }
    }
...
Retrieval Models for Source Code

Attribute-based Vector Models

```java
//subloop. for each node...
for (int nodeIndex = 0; nodeIndex < n; nodeIndex++) {
    int nodeId = nodeIdPermutation[nodeIndex];
    //System.out.println("node: " + nodeId);
    //reset sums.
    for (int i = 0; i < n; i++) sumOfEdgeWeights[i] = 0;
    //sum all the edges going out to the same cluster
    int[] adjacentNodes = graph.getAdjacentNodes(nodeId);
    for (int i : adjacentNodes) {
        int clusterId = nodes2cluster[i];
        double edgeWeight = graph.getEdgeWeight(nodeId, i);
        if (edgeWeight >= threshold) {
            sumOfEdgeWeights[clusterId] += edgeWeight;
        }
    }
    //and determine the cluster of biggest sum.
    int newClusterNumber = nodes2cluster[nodeId];
    double maxWeight = 0;
    for (int i = 0; i < sumOfEdgeWeights.length; i++) {
        if ((sumOfEdgeWeights[i]) > maxWeight) {
            newClusterNumber = i;
            maxWeight = sumOfEdgeWeights[i];
        }
    }
    //...
}
```

<table>
<thead>
<tr>
<th>Representation d</th>
<th>Sim. measure ( \varphi )</th>
<th>Compilation level for ( d )</th>
<th>Runtime for ( \varphi )</th>
</tr>
</thead>
<tbody>
<tr>
<td>software metric features</td>
<td>cosine</td>
<td>none</td>
<td>( O(</td>
</tr>
<tr>
<td>all ( n ) grams</td>
<td>Jaccard</td>
<td>lexical</td>
<td>( O(</td>
</tr>
<tr>
<td>subset of all ( n ) grams</td>
<td>Jaccard</td>
<td>lexical</td>
<td>( O(</td>
</tr>
<tr>
<td>( n &lt; 5 )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

22 webis@LWA October 6, 2008
Retrieval Models for Source Code

Structure-based String Models

for (int nodeIndex = 0; nodeIndex < n; nodeIndex++) {
    int nodeId = nodeIdPermutation[nodeIndex];
    //System.out.println("node: "+ nodeId);
    //reset sums.
    for (int i = 0; i < n; i++) sumOfEdgeWeights[i] = 0;
    //sum all the edges going out to the same cluster
    int[] adjacentNodes = graph.getAdjacentNodes(nodeId);
    for (int i : adjacentNodes) {
        int clusterId = nodes2cluster[i];
        double edgeWeight = graph.getEdgeWeight(nodeId, i);
        if (edgeWeight >= threshold) {
            sumOfEdgeWeights[clusterId] += edgeWeight;
        }
    }
    //and determine the cluster of biggest sum.
    int newClusterNumber = nodes2cluster[nodeId];
    double maxWeight = 0;
    for (int i = 0; i < sumOfEdgeWeights.length; i++) {
        if ((sumOfEdgeWeights[i]) > maxWeight) {
            newClusterNumber = i;
            maxWeight = sumOfEdgeWeights[i];
        }
    }
    //...
Retrieval Models for Source Code

Comparison of Structure-based String Models

For “compression ratio”, “greedy string tiling”, and “longest common substring” the heart of $\varphi$ is substring maximization.
Retrieval Models for Source Code

Comparison of Structure-based String Models

For “compression ration”, “greedy string tiling”, and “longest common substring” the heart of $\varphi$ is substring maximization.
Retrieval Models for Source Code
Comparison of Structure-based String Models

For “compression ration”, “greedy string tiling”, and “longest common substring” the heart of $\varphi$ is substring maximization.
Retrieval Models for Source Code

Comparison of Structure-based String Models

For “compression ration”, “greedy string tiling”, and “longest common substring” the heart of $\varphi$ is substring maximization.
Retrieval Models for Source Code

Comparison of Structure-based String Models

For “compression ration”, “greedy string tiling”, and “longest common substring” the heart of $\varphi$ is substring maximization.
Retrieval Models for Source Code

Comparison of Structure-based String Models

For “compression ration”, “greedy string tiling”, and “longest common substring” the heart of $\varphi$ is substring maximization.
Retrieval Models for Source Code

Comparison of Structure-based String Models

For “compression ration”, “greedy string tiling”, and “longest common substring” the heart of $\varphi$ is substring maximization.

$$\varphi(s_q, s_x) = \frac{2 \cdot |lcs(s_q, s_x)|}{|s_q| + |s_x|}$$
Retrieval Models for Source Code
Comparison of Structure-based String Models

Corpus:
- open source project JNode, (Java New Operating System Design Effort)
- 18 subsequent release versions, 80 091 documents
- 121 215 methods

Experiment (plot below): sample of 50 000 method pairs, drawn i.i.d.
Rationale:

- the inherent quadratic situation becomes linear
- code repositories become extremely large
- because of the problem structure we are interested in plagiarism candidates; a human inspection is always necessary
Hash-based Search: Motivation
Hash-based Search: Motivation

Nearest Neighbor Search

Applications:

- elimination of duplicates / near duplicates
- identification of versioned and plagiarized documents
- retrieval of similar documents
- identification of source code plagiarism
Indexing with space partitioning methods:

- **Quad-tree.**
  
  Split the space recursively into sub-squares until only a few points left.
  
  Space exponential in dimension; time exponential in dimension.

- **Kd-tree.** Linear space; exponential query time is still possible.
Hash-based Search: Motivation

Nearest Neighbor Search

Indexing with data partitioning methods:

- R-tree.
  Bottom-up; heuristically construct minimum bounding regions for points
  Works well for low dimensions (< 10).
- Rf-tree, X-tree, ...
Hash-based Search: Motivation

Document Representation and Search

The nearest neighbor problem cannot be solved efficiently in high dimensions by partitioning methods.

“Existing methods are outperformed on average by a simple sequential scan, if the number of dimensions exceeds around 10.”

[Weber 99, Gionis/Indyk/Motwani 99-04]
Hash-based Search: Motivation

Document Representation and Search

The nearest neighbor problem cannot be solved efficiently in high dimensions by partitioning methods.

“Existing methods are outperformed on average by a simple sequential scan, if the number of dimensions exceeds around 10.”

[Weber 99, Gionis/Indyk/Motwani 99-04]

English Wikipedia:

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>Number of dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-gram space</td>
<td>3 921 588</td>
</tr>
<tr>
<td>4-gram space</td>
<td>274 101 016</td>
</tr>
<tr>
<td>8-gram space</td>
<td>373 795 734</td>
</tr>
<tr>
<td>Shingling space</td>
<td>75 659 644</td>
</tr>
</tbody>
</table>
Hash-based Search: Motivation

Document Representation and Search

Given the representation \( x_{dq} \) of a query document and a collection \( D \).

- Linear comparison under some BOW representation
  - Similarity ranking (baseline)
Hash-based Search: Motivation

Document Representation and Search

Given the representation $x_{dq}$ of a query document and a collection $D$.

- Linear comparison under some BOW representation
  $\Rightarrow$ Similarity ranking (baseline)

- Linear comparison under some compact representation
  $\Rightarrow$ Acceptable similarity ranking (85% recall at $\varphi > 0.5$)
Hash-based Search: Motivation

Document Representation and Search

Given the representation $x_{dq}$ of a query document and a collection $D$.

- Linear comparison under some BOW representation
  - Similarity ranking (baseline)

- Linear comparison under some compact representation
  - Acceptable similarity ranking (85% recall at $\varphi > 0.5$)

- Comparison in constant time with a similarity-sensitive hash function $h_\varphi$
  - Binary decision wrt. threshold $\theta$ (similar if $\varphi > \theta$ / not similar if $\varphi \leq \theta$)
Hash-based Search: Motivation

Hash-based Search is a Space Partitioning Method

\[ \theta \]

\[ x_{d1}, x_{d2}, x_{d3}, x_{d4} \]
Hash-based Search: Motivation

Hash-based Search is a Space Partitioning Method

\[ h_\phi(x_{d1}) = \{13\} \]
\[ h_\phi(x_{d2}) = \{14\} \]
\[ h_\phi(x_{d3}) = \{16\} \]
\[ h_\phi(x_{d4}) = \{16\} \]
Hash-based Search: Motivation

Hash-based Search is a Space Partitioning Method

\[ h_{\Phi}(x_{d_1}) = \{13, 24\} \]
\[ h_{\Phi}(x_{d_2}) = \{14, 24\} \]
\[ h_{\Phi}(x_{d_3}) = \{16, 24\} \]
\[ h_{\Phi}(x_{d_4}) = \{16, 26\} \]
Hash-based Search: Motivation

Hash-based Search is a Space Partitioning Method

Similarity collision condition:

\[
( h^*(x_{d1}) \cap h^*(x_{d2}) ) \neq \emptyset \iff \varphi(x_{d1}, x_{d2}) > \theta
\]

- \( h_{\varphi}(x_{d1}) = \{13, 24\} \)
- \( h_{\varphi}(x_{d2}) = \{14, 24\} \)
- \( h_{\varphi}(x_{d3}) = \{16, 24\} \)
- \( h_{\varphi}(x_{d4}) = \{16, 26\} \)
Hash-based Search: Motivation

Hash-based Search is a Space Partitioning Method

Similarity collision condition:

$$\big( h_\varphi(x_{d_1}) \cap h_\varphi(x_{d_2}) \big) \neq \emptyset \iff \varphi(x_{d_1}, x_{d_2}) > \theta$$
Hash-based Search: Motivation

Issues about Hash-based Search

- Hash-based search reduces a cont. similarity relation to a binary relation.
- Hash-based search is a space partitioning method.
- Space partitioning is realized by a similarity-sensitive hash function $h_\varphi$.

- Equal codes under $h_\varphi$ indicate similar objects with a high probability.
  
  Precision: \[ h_\varphi(x_{d_1}) \cap h_\varphi(x_{d_2}) \neq \emptyset \implies P(\varphi(x_{d_1}, x_{d_2}) > \theta) \text{ is high} \]

- $h_\varphi$ maps similar objects on equal codes with a high probability.
  
  Recall: \[ \varphi(x_{d_1}, x_{d_2}) > \theta \implies P(h_\varphi(x_{d_1}) \cap h_\varphi(x_{d_2}) \neq \emptyset) \text{ is high} \]

- $h_\varphi$ must be multi-valued if $D$ is partly unknown.

- A perfectly similarity-sensitive hash function $h^*_\varphi$ may exist for each $D$. 
Hash-based Search

Construction Principles for $h_{\phi}$: Shingling  [Broder 2000]

Embedding $\rightarrow$ Quantization $\rightarrow$ Encoding

$\pi_1 : V \rightarrow \{1, ..., |V|\}$

Synchronized random projection
Hash-based Search

Construction Principles for $h_{\varphi}$: Shingling  [Broder 2000]

Embedding $\rightarrow$ Quantization $\rightarrow$ Encoding

$\pi_1 : V \rightarrow \{1, ..., |V|\}$

Synchronized random projection

$\text{MD5}( v \mid \pi_1(v) = \text{min}(\pi_1))$
Hash-based Search

Construction Principles for \( h_\varphi \): Shingling  [Broder 2000]

\[ \pi_1 : V \rightarrow \{1, \ldots, |V|\} \]
\[ \pi_2 : V \rightarrow \{1, \ldots, |V|\} \]
\[ \vdots \]
\[ \pi_k : V \rightarrow \{1, \ldots, |V|\} \]

Synchronized random projection

\[ d \]

\[ |V| \]

\[ \text{MD5} \left( \text{v} \mid \pi_1(v) = \min(\pi_1) \right) \]
Hash-based Search

Construction Principles for $h_\varphi$: Shingling  [Broder 2000]

Embedding $\rightarrow$ Quantization $\rightarrow$ Encoding

$\pi_1 : V \rightarrow \{1, \ldots, |V|\}$
$\pi_2 : V \rightarrow \{1, \ldots, |V|\}$
$\vdots$
$\pi_k : V \rightarrow \{1, \ldots, |V|\}$

Synchronized random projection

$\text{MD5}(v \mid \pi_1(v) = \min(\pi_1))$
$\text{MD5}(v \mid \pi_2(v) = \min(\pi_2))$
$\vdots$
$\text{MD5}(v \mid \pi_k(v) = \min(\pi_k))$
Hash-based Search

Construction Principles for $h_\psi$: Shingling  [Broder 2000]

Embedding $\rightarrow$ Quantization $\rightarrow$ Encoding

\[ \pi_1 : V \rightarrow \{1, ..., |V| \} \]
\[ \pi_2 : V \rightarrow \{1, ..., |V| \} \]
\[ \vdots \]
\[ \pi_k : V \rightarrow \{1, ..., |V| \} \]

Projection and quantization of MD5 hashes.

"Super-shingling"

Fingerprint = \{2643256, 325567\} = $h_\psi(x_d)$
Hash-based Search

Construction Principles for $h_\varphi$: Fuzzy-Fingerprinting

Documents from the British National Corpus
Hash-based Search

Construction Principles for $h_\psi$: Fuzzy-Fingerprinting

- Embedding
- Quantization
- Encoding

A priori probabilities from BNC

Distribution of prefix classes in sample

Normalization and difference computation

Documents from the British National Corpus
Hash-based Search

Construction Principles for $h_\varphi$: Fuzzy-Fingerprinting

A priori probabilities from BNC

Documents from the British National Corpus

Embedding $\rightarrow$ Quantization $\rightarrow$ Encoding

$\Phi(x_d) = \sum_{i=1}^{k} \rho(y_i) \cdot r^{i-1}$

→ Fingerprint = \{2643256, 55\}
Hash-based Search

Construction Principles for $h_\varphi$: Fuzzy-Fingerprinting

A priori probabilities from BNC

Distribution of prefix classes in sample

Normalization and difference computation

Embedding $\rightarrow$ Quantization $\rightarrow$ Encoding

Documents from the British National Corpus

$\Rightarrow$ Fingerprint = \{2643256, 56\}

$h_\varphi^{(\rho)}(x_d) = \sum_{i=1}^{k} \rho(y_i) \cdot r^{i-1}$
Hash-based Search

Construction Principles for $h_\varphi$: Fuzzy-Fingerprinting

Embedding $\rightarrow$ Quantization $\rightarrow$ Encoding

A priori probabilities from BNC $\downarrow$
Distribution of prefix classes in sample $\downarrow$

Normalization and difference computation

Fuzzification

$\sum_{i=1}^{k} \rho(y_i) \cdot r^{i-1}$

Documents from the British National Corpus

$\rightarrow$ Fingerprint = \{2643256, 325567\} = h_\varphi(x_d)
Hash-based Search

Properties of \( h_{\varphi} \)

Code length controls precision.

The collision probability \( P(h_{\varphi}(x_{d_1}) \cap h_{\varphi}(x_{d_2}) \neq \emptyset \mid \varphi(x_{d_1}, x_{d_2}) \leq \theta) \) goes down if

- the number \( k \) of random vectors (p-stable LSH)
- the number \( k \) of prefix classes (Fuzzy-fingerprinting)
- \( \ldots \)

is increased.
Hash-based Search

Properties of $h_\varphi$

Code length controls precision.

The collision probability $P(h_\varphi(x_{d_1}) \cap h_\varphi(x_{d_2}) \neq \emptyset \mid \varphi(x_{d_1}, x_{d_2}) \leq \theta)$ goes down if

- the number $k$ of random vectors (p-stable LSH)
- the number $k$ of prefix classes (Fuzzy-fingerprinting)
- ...

is increased.

Code multiplicity controls recall.

The collision probability $P(h_\varphi(x_{d_1}) \cap h_\varphi(x_{d_2}) \neq \emptyset \mid \varphi(x_{d_1}, x_{d_2}) > \theta)$ goes up if

- the number $l$ of vector sets (p-stable LSH)
- the number $l$ of fuzzification schemes (Fuzzy-fingerprinting)
- ...

is increased.
Retrieval Models for Source Code

Fingerprint-based Models

Corpus: as before

Experiment (plot below): 200 queries against fingerprinted corpus

Baseline: greedy string tiling

![Graph showing precision and recall for different similarity intervals.]

- Shingling
- Fuzzy fingerprinting
Retrieval Models for Source Code

Fingerprint-based Models

Corpus: as before

Experiment (plot below): 200 queries against fingerprinted corpus

Baseline: greedy string tiling

![Graph showing precision and recall for different similarity intervals.](image)

- **Shingling**
- **Fuzzy fingerprinting**

![Graph showing retrieval of text documents.](image)
Summary

1. Survey of retrieval models for high-similarity search in source code.

2. We propose the longest common subsequence for the class of structure-based string models:
   - better suited for short source code fragments
   - $\varphi$ computation in $O(|d|^2)$ instead of in $O(|d|^3)$

3. We investigate the use of hash-based search high-similarity search in source code:
   - basis is the class of structure-based string models
   - real-world order of magnitudes become possible
   - the ad-hoc application of existing technology leads to unsatisfying recall
Thank you!