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

Fabian Loose

Steffen Becker

Martin Potthast

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.

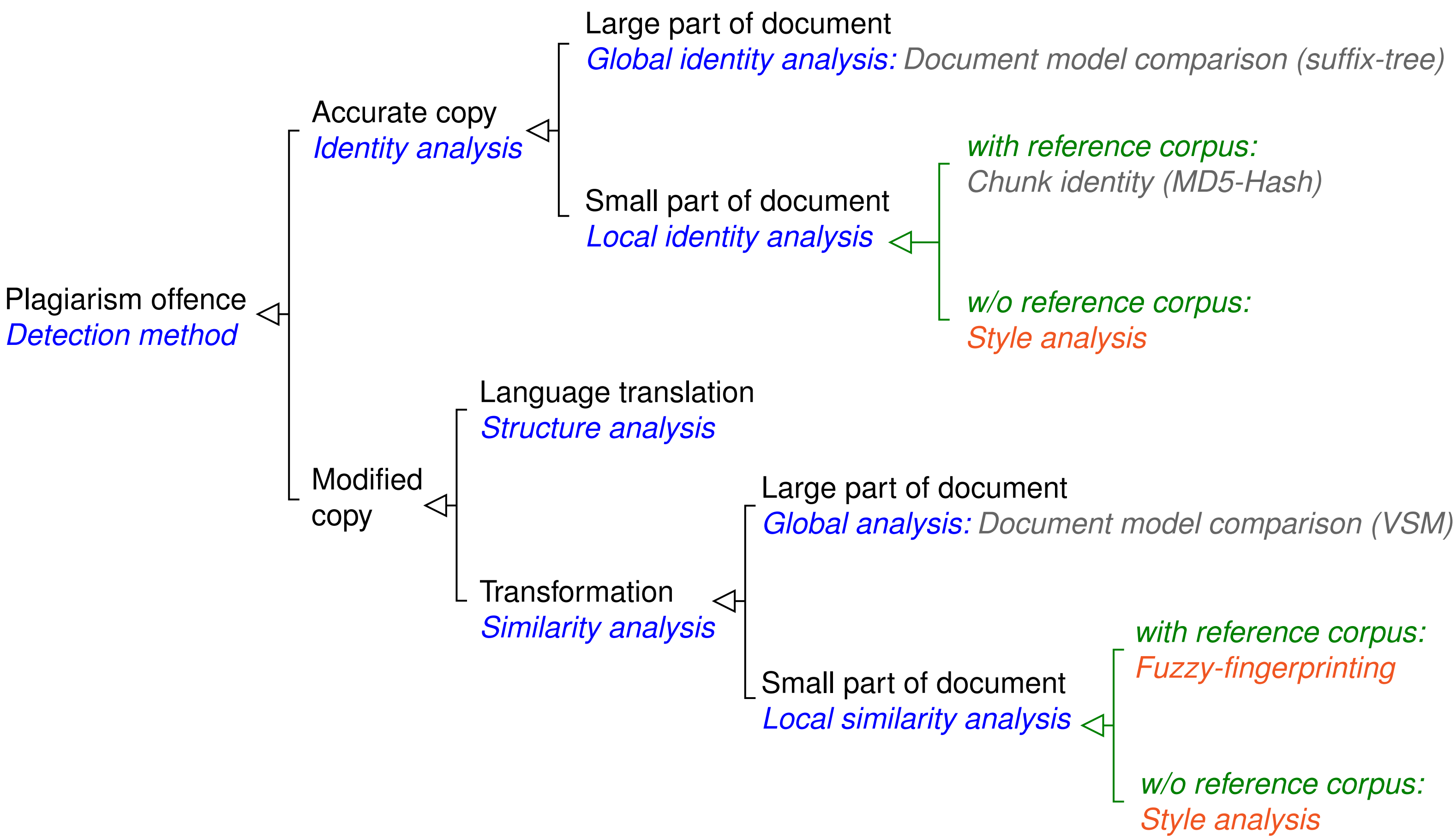
[Wikipedia: Plagiarism]

- 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]

[1] D. McCabe. Research Report of the Center for Academic Integrity. `http://www.academicintegrity.org`, 2005.

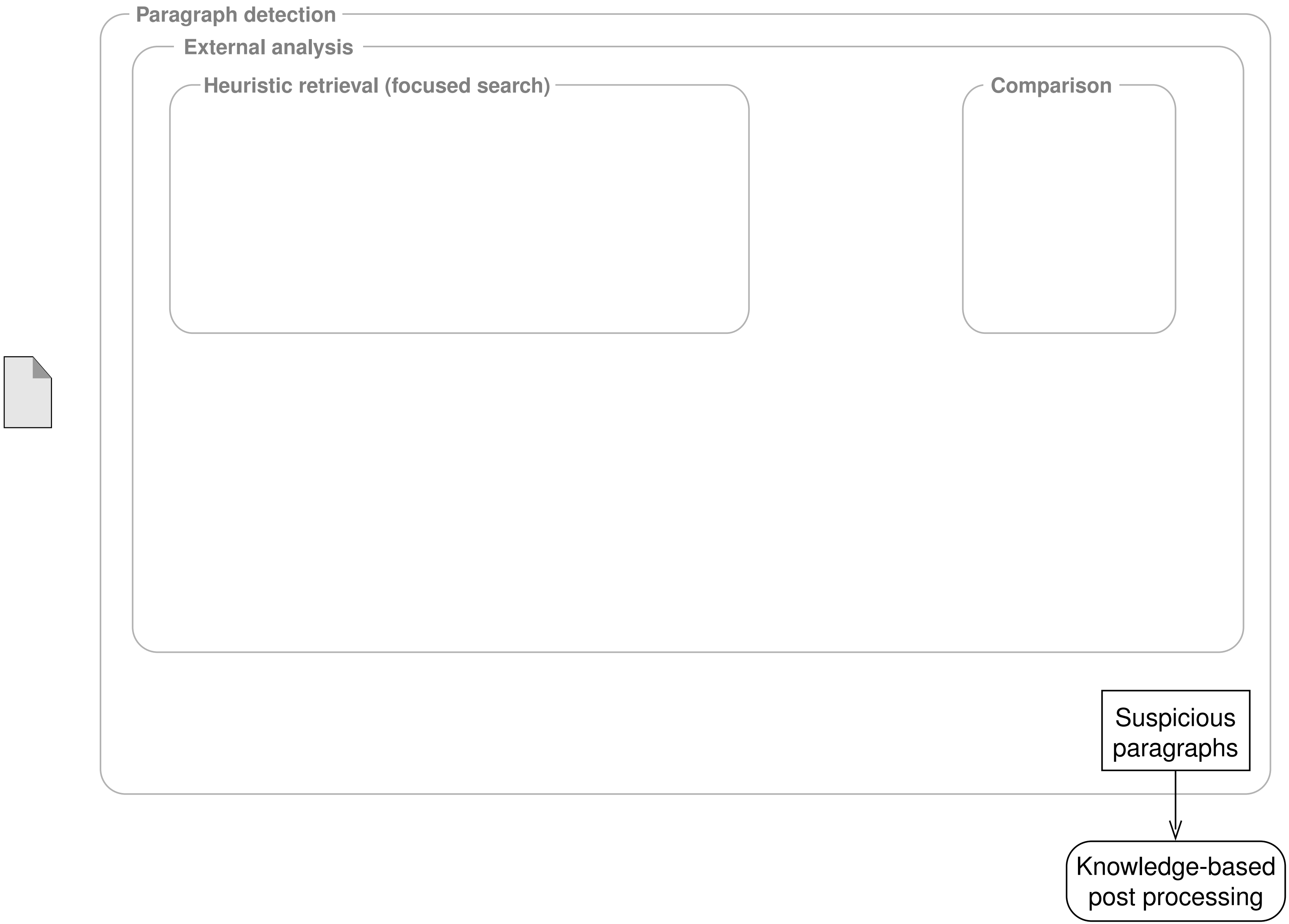
Overview

Taxonomy of Plagiarism Offenses

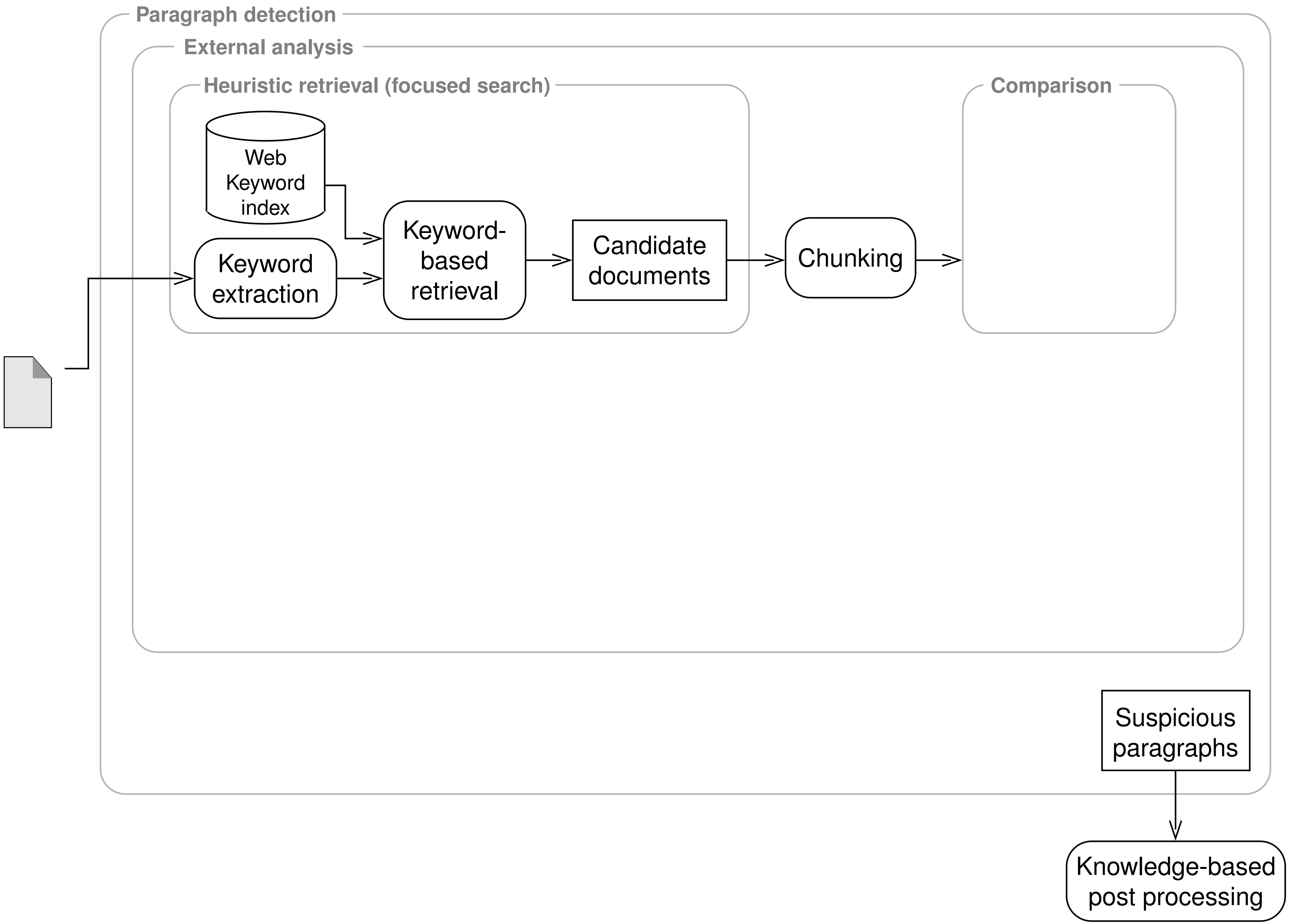


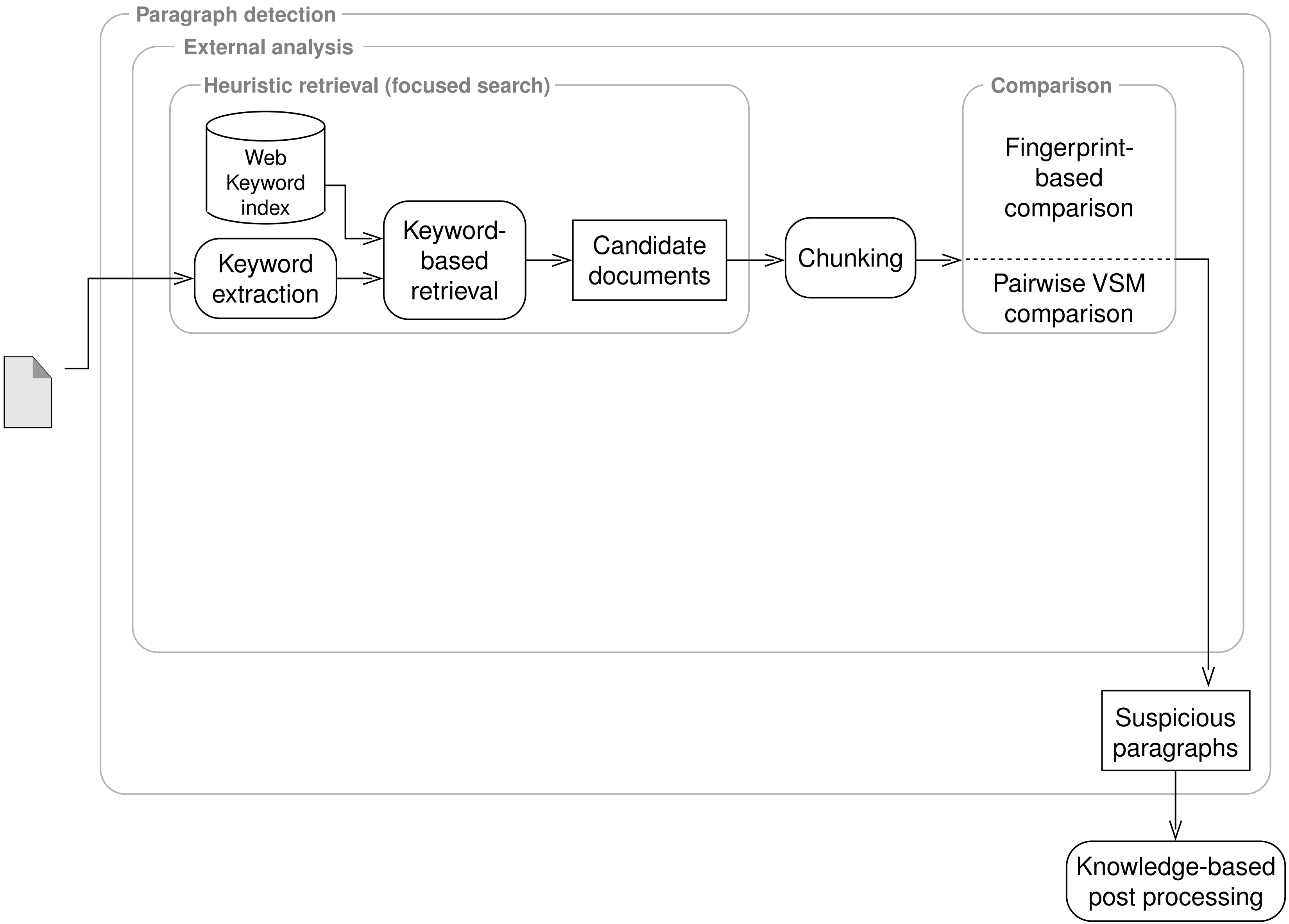
Paragraph detection

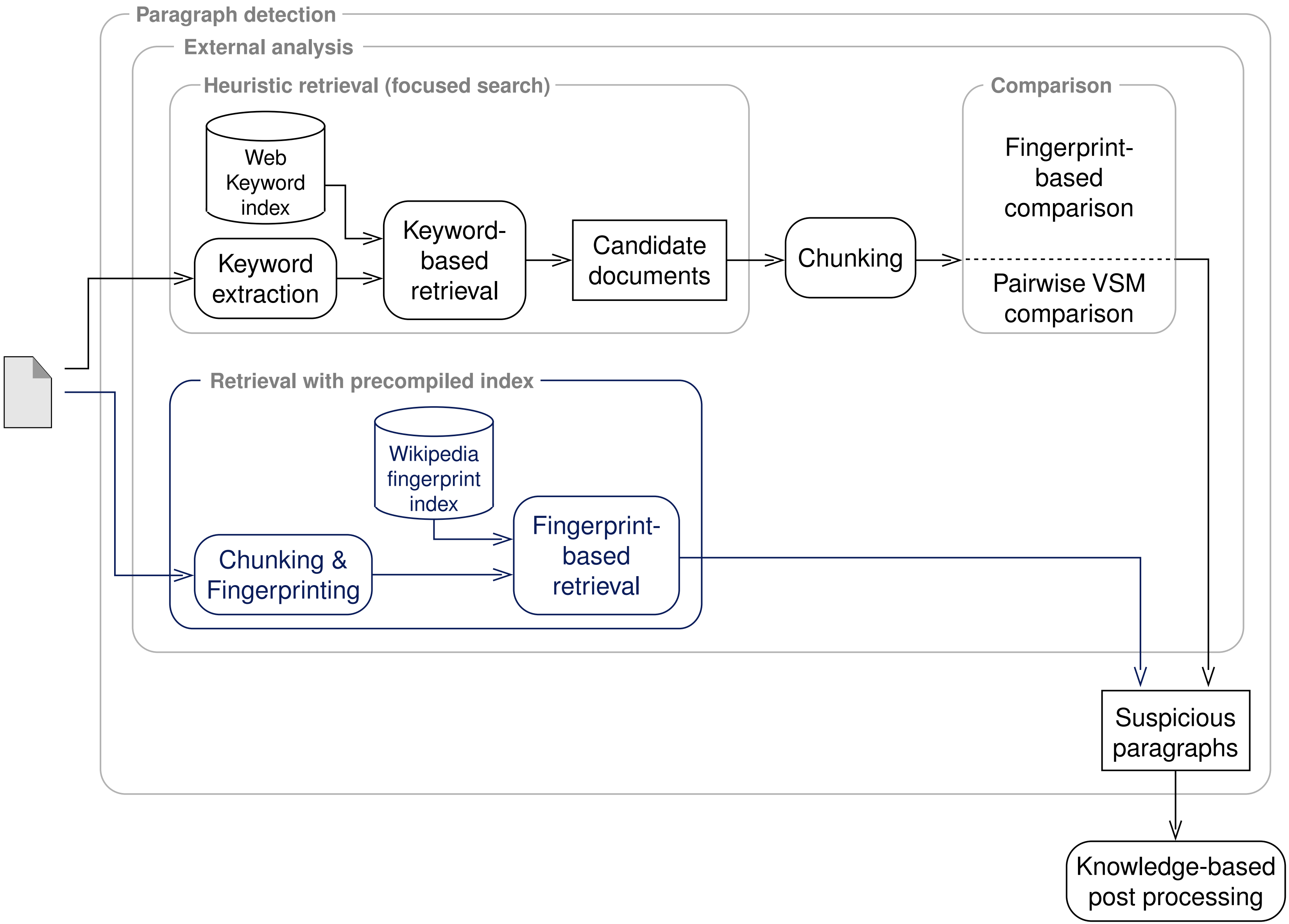
Knowledge-based post processing

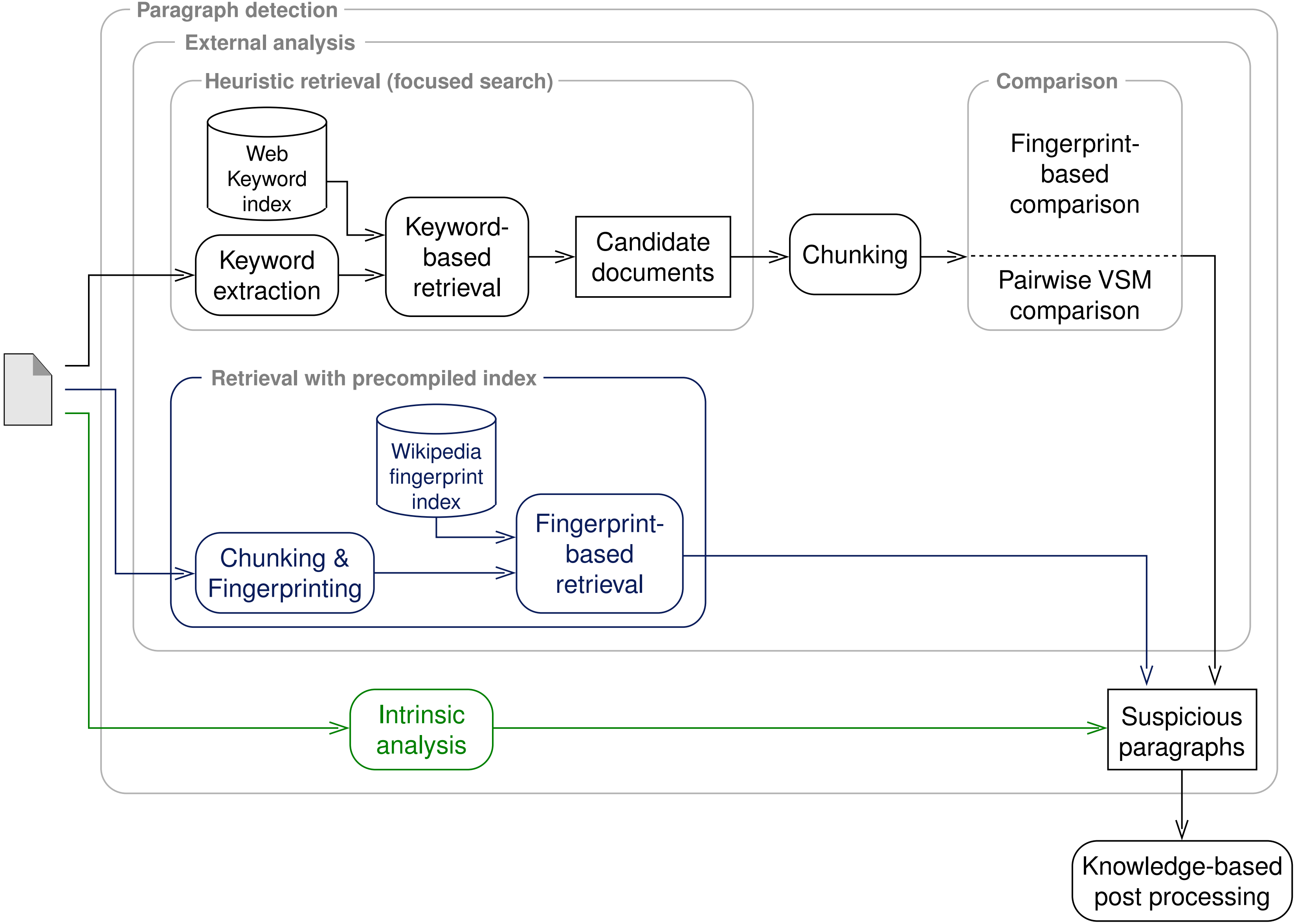


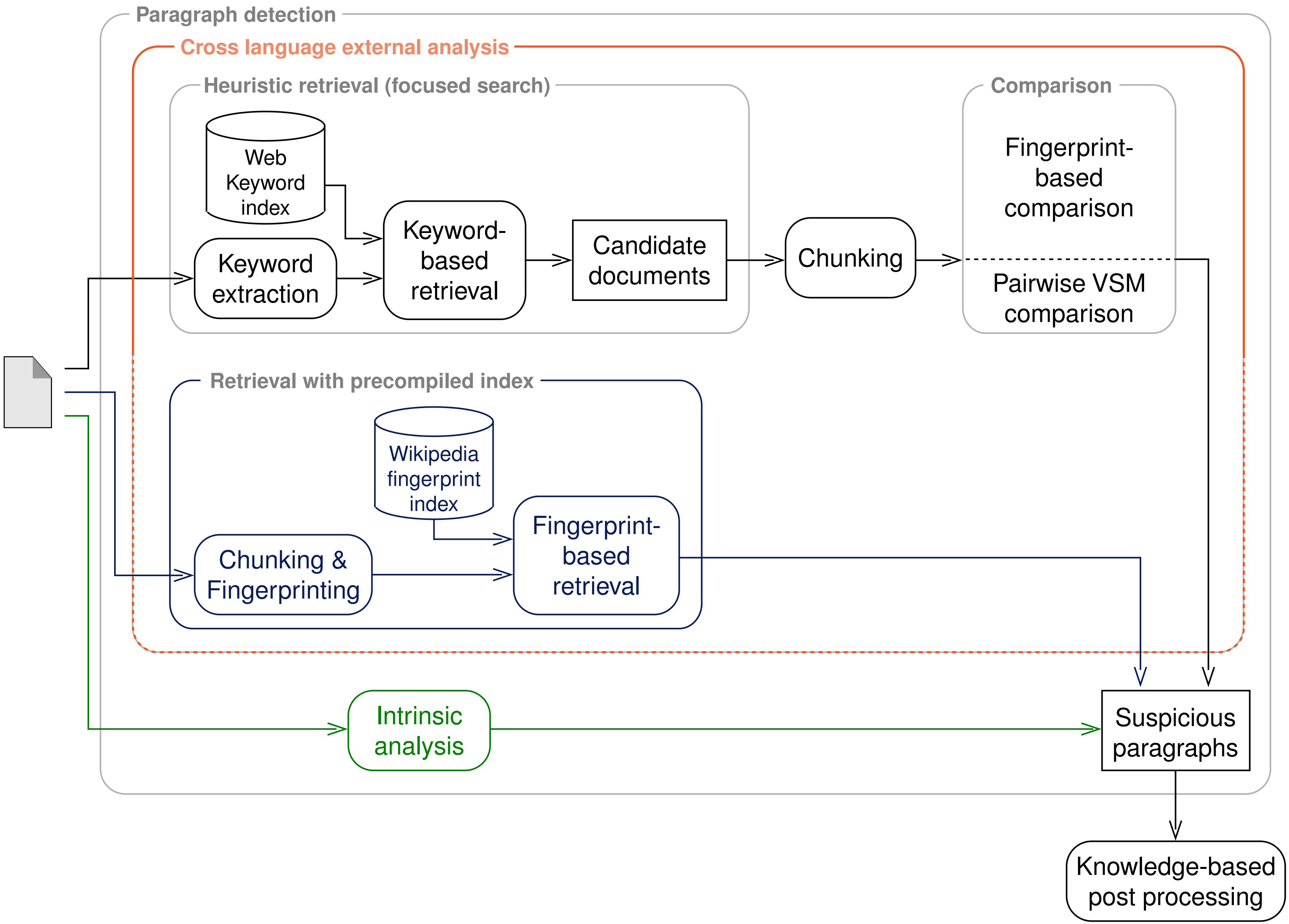
Paragraph detection
External analysis
Heuristic retrieval (focused search)
Comparison
Suspicious paragraphs
Knowledge-based post processing


Paragraph detection
External analysis
Heuristic retrieval (focused search)
Comparison
Web Keyword index
Keyword extraction
Keyword-based retrieval
Candidate documents
Chunking
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
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".

[Brin et al. 1995, Monostori et al. 2001-2004, Stein et al. 2004-2006, etc.]

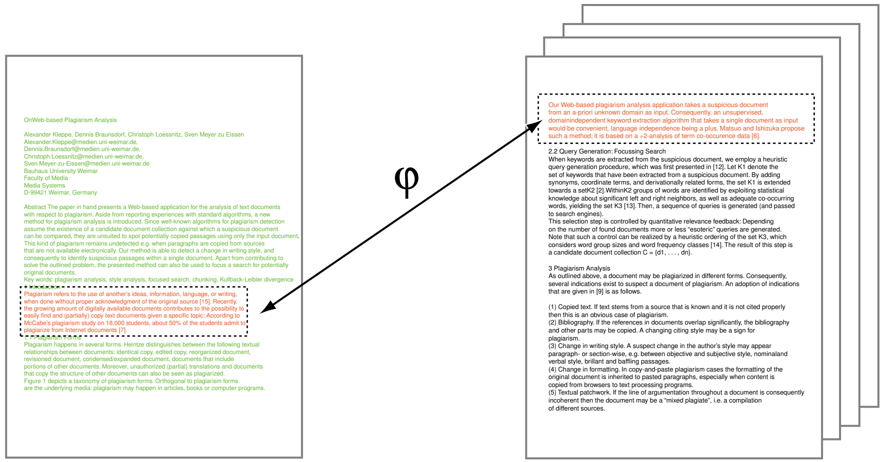
External plagiarism analysis formulated as decision problem:

Problem. AVEXTERN (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 θ?

Overview

Basic Principle

- Partition each document in meaningful sections, also called "chunks".
- Do a pairwise comparison using a similarity function φ.



suspicious document corpus documents

Complexity:

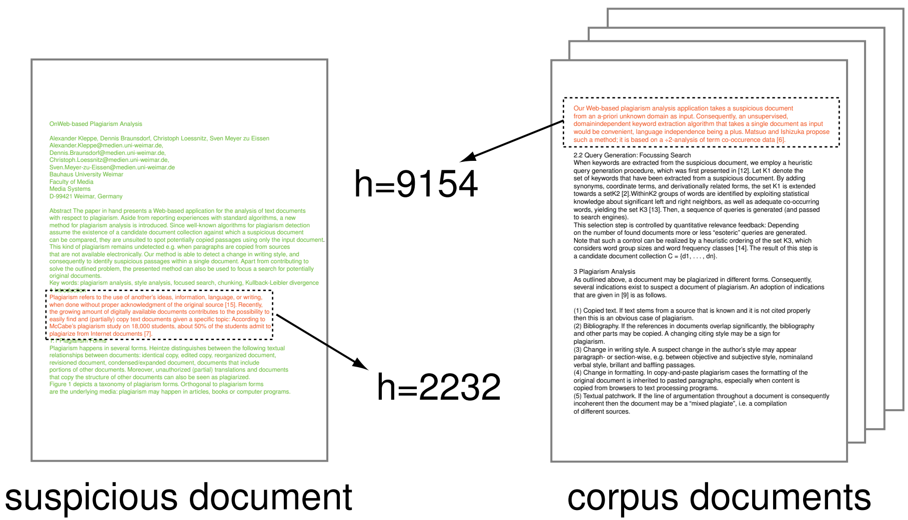
n documents in corpus, c chunks per document on average

➜ $O(n \cdot c^2)$ 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:

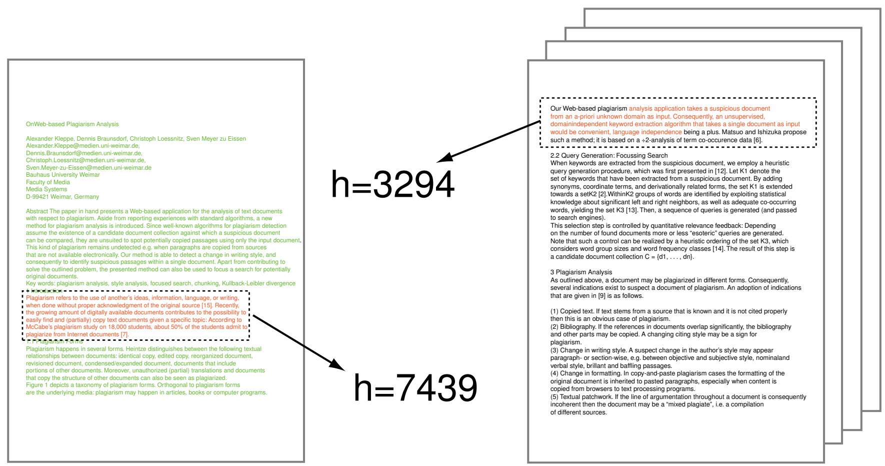
n documents in corpus, c chunks per document on average

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

Overview

Comparison with Fingerprints (Level 2)

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



suspicious document　　corpus documents

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:

  $$h(c_1) = h(c_2) \Rightarrow c_1 = c_2 \qquad \text{(with very high probability)}$$

Current research:

- Focus on *fuzzy* hash functions h_φ:

  $$h_\varphi(c_1) = h_\varphi(c_2) \quad \Rightarrow \quad P(\varphi(c_1, c_2) > \theta) \geq 1 - \varepsilon$$

  [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

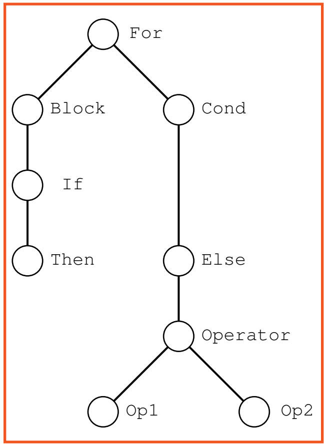
Retrieval Models for Source Code

```
//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 d	Sim. measure φ	Compilation level for d	Runtime for φ

Retrieval Models for Source Code

Structure-based Graph Models

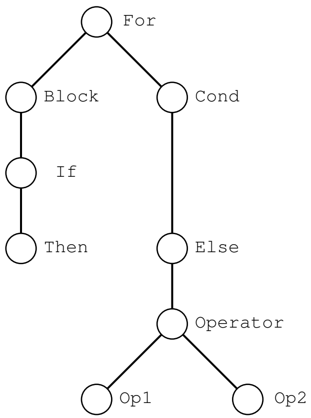



```
//subloop. for each node...
for(int nodeIndex = 0; nodeIndex<n; nodeIndex++) {
    int nodeId = nodeIdPermutation[nodeIndex];
    //System.out.println("node: "+nodeId);

    //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];
     }
   }
  ...
```

Representation $\mathbf{d}$	Sim. measure φ	Compilation level for d	Runtime for φ	
abstract syntax trees	hash-based subtree search	syntactical	$O(\lvert\mathbf{d}\rvert)$	[Baxter et al. 1998]
conceptual graphs	heuristically focused isomorphic graph search	semantic	$O(\lvert\mathbf{d}\rvert^3)$	[Mishne et al. 2004]
program dep. graphs	isomorphic graph search	semantic	NP-complete	[Liu et al. 2006]

Retrieval Models for Source Code

Attribute-based Vector 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];
      }
   }
  ...
```

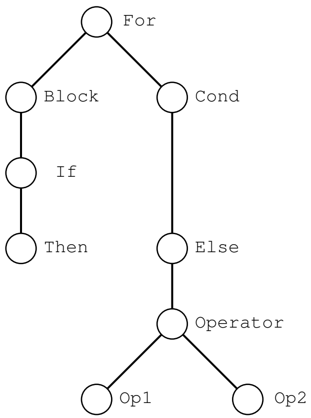
```
for ( int nodeIndex
= 0 ; nodeIndex
< n ; nodeIndex++
) { int nodeId
= nodeIdPer [ nodeIndex
] ; for (
int i = 0
; i < n
; i ++ )
sumOfEdgeWeights [ i ]
= 0 ; int
[ ] adjacentNodes =
graph . getAdNodes (
nodeId ) ; for
( int i :
adjacentNodes ) { int
clusterId = nodes2clu [
i ] ; double
edgeWeight = graph .
getEdgeWeight ( nodeId ,
i ) ; if
...
```

Representation $\mathbf{d}$	Sim. measure φ	Compilation level for d	Runtime for φ	
software metric features	cosine	none	$O(\|\mathbf{d}\|)$	[Ottenstein 1976]
all n grams	Jaccard	lexical	$O(\|\mathbf{d}\|)$	[Clough et al. 2002]
subset of all n grams	Jaccard	lexical	$O(\|\mathbf{d}\|)$	[Schleimer et al. 2003]

$n < 5$

Retrieval Models for Source Code

Structure-based String 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];
      }
   }
  ...
```

```
for ( int nodeIndex
= 0 ; nodeIndex
< n ; nodeIndex++
) { int nodeId
= nodeIdPer [ nodeIndex
] ; for (
int i = 0
; i < n
; i ++ )
sumOfEdgeWeights [ i ]
= 0 ; int
[ ] adjacentNodes =
graph . getAdNodes (
nodeId ) ; for
( int i :
adjacentNodes ) { int
clusterId = nodes2clu [
i ] ; double
edgeWeight = graph .
getEdgeWeight ( nodeId ,
i ) ; if
...
```

```
BEGINFOR VARDEF BEGINFOR ASSIGN
VARDEF ASSIGN BEGINFOR ASSIGn
ENDFOR ASSIGN ENDFOR ...
```

Representation $\mathbf{d}$	**Sim. measure** φ	**Compilation level for** d	**Runtime for** φ	
string of token types	compression ratio	lexical	$O(\|\mathbf{d}\|^2)$	[Chen et al. 2004]
string of token types	greedy string tiling	lexical	$O(\|\mathbf{d}\|^3)$	[Prechelt et al. 2000]
string of token types	longest common substring	lexical	$O(\|\mathbf{d}\|^2)$	[Burrows et al. 2000]
string of token types	longest common subseq.	lexical	$O(\|\mathbf{d}\|^2)$	[new]

Retrieval Models for Source Code

Comparison of Structure-based String Models

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

```
BEGINFOR VARDEF BEGINFOR ASSIGN VARDEF ASSIGN BEGINFOR ASSIGN
```

```
BEGINFOR VARDEF VARDEF ASSIGN CASE BEGINSWITCH BEGINFOR ASSIGN
```

Retrieval Models for Source Code

Comparison of Structure-based String Models

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

```
BEGINFOR VARDEF BEGINFOR ASSIGN VARDEF ASSIGN BEGINFOR ASSIGN

BEGINFOR VARDEF VARDEF ASSIGN CASE BEGINSWITCH BEGINFOR ASSIGN
```

Retrieval Models for Source Code

Comparison of Structure-based String Models

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

```
BEGINFOR VARDEF BEGINFOR ASSIGN VARDEF ASSIGN BEGINFOR ASSIGN

BEGINFOR VARDEF VARDEF ASSIGN CASE BEGINSWITCH BEGINFOR ASSIGN
```

Retrieval Models for Source Code

Comparison of Structure-based String Models

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

```
[BEGINFOR VARDEF] BEGINFOR ASSIGN VARDEF ASSIGN BEGINFOR ASSIGN

[BEGINFOR VARDEF] VARDEF ASSIGN CASE BEGINSWITCH BEGINFOR ASSIGN
```

Retrieval Models for Source Code

Comparison of Structure-based String Models

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

```
BEGINFOR VARDEF BEGINFOR ASSIGN VARDEF ASSIGN BEGINFOR ASSIGN

BEGINFOR VARDEF VARDEF ASSIGN CASE BEGINSWITCH BEGINFOR ASSIGN
```

Retrieval Models for Source Code

Comparison of Structure-based String Models

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

```
[BEGINFOR VARDEF] BEGINFOR ASSIGN [VARDEF ASSIGN] [BEGINFOR ASSIGN]

[BEGINFOR VARDEF] [VARDEF ASSIGN] CASE BEGINSWITCH [BEGINFOR ASSIGN]
```

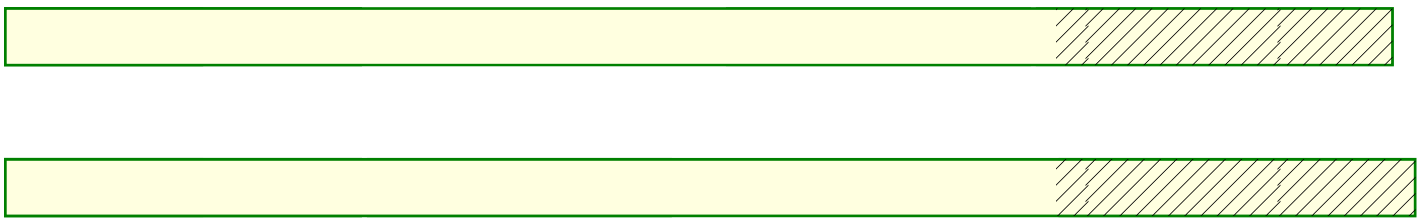
Retrieval Models for Source Code

Comparison of Structure-based String Models

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

```
[BEGINFOR VARDEF] BEGINFOR ASSIGN [VARDEF ASSIGN] [BEGINFOR ASSIGN]

[BEGINFOR VARDEF] [VARDEF ASSIGN] CASE BEGINSWITCH [BEGINFOR ASSIGN]
```

Longest common subsequence:



$$\varphi(\mathbf{s}_q, \mathbf{s}_x) = \frac{2 \cdot |\text{lcs}(\mathbf{s}_q, \mathbf{s}_x)|}{|\mathbf{s}_q| + |\mathbf{s}_x|}$$

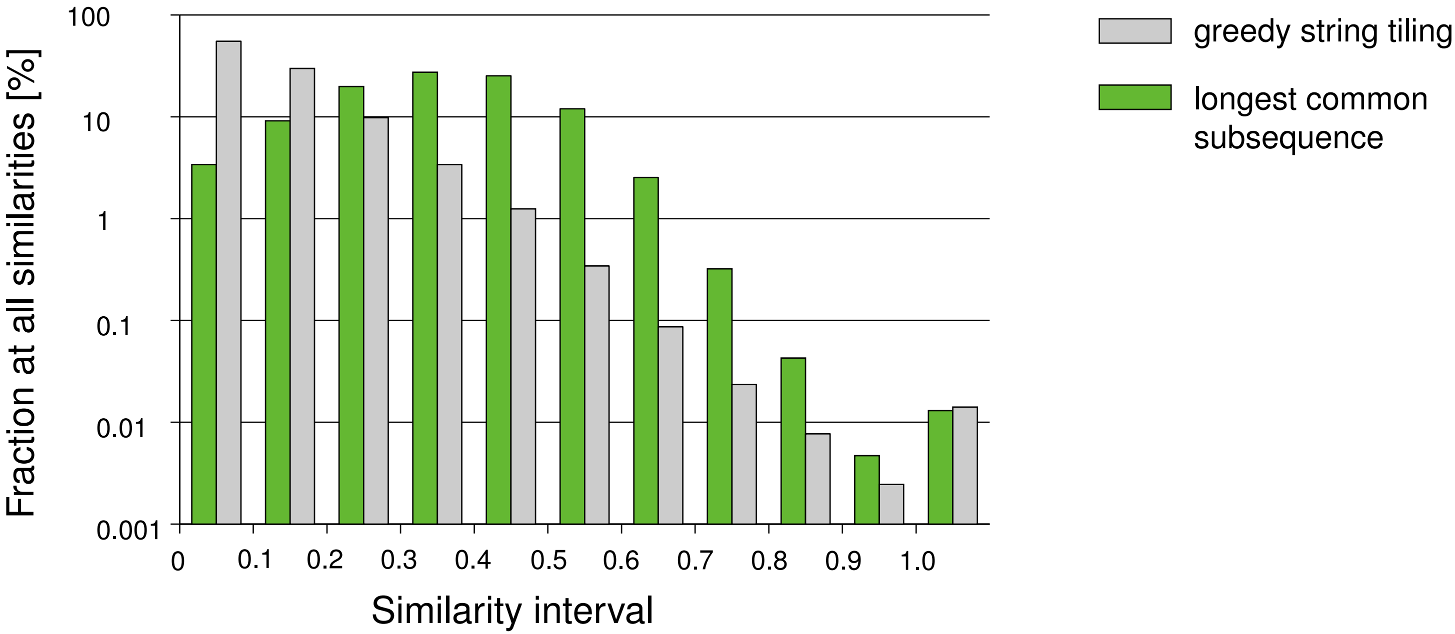
Retrieval Models for Source Code

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.



Retrieval Models for Source Code

Fingerprint-based 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];
      }
   }
  ...
```

➜

```
for ( int nodeIndex
= 0 ; nodeIndex
< n ; nodeIndex++
) { int nodeId
= nodeIdPer [ nodeIndex
] ; for (
int i = 0
; i < n
; i ++ )
sumOfEdgeWeights [ i ]
= 0 ; int
[ ] adjacentNodes =
graph . getAdNodes (
nodeId ) ; for
( int i :
adjacentNodes ) { int
clusterId = nodes2clu [
i ] ; double
edgeWeight = graph .
getEdgeWeight ( nodeId ,
i ) ; if
...
```

```
{2323753332,345256745}
```

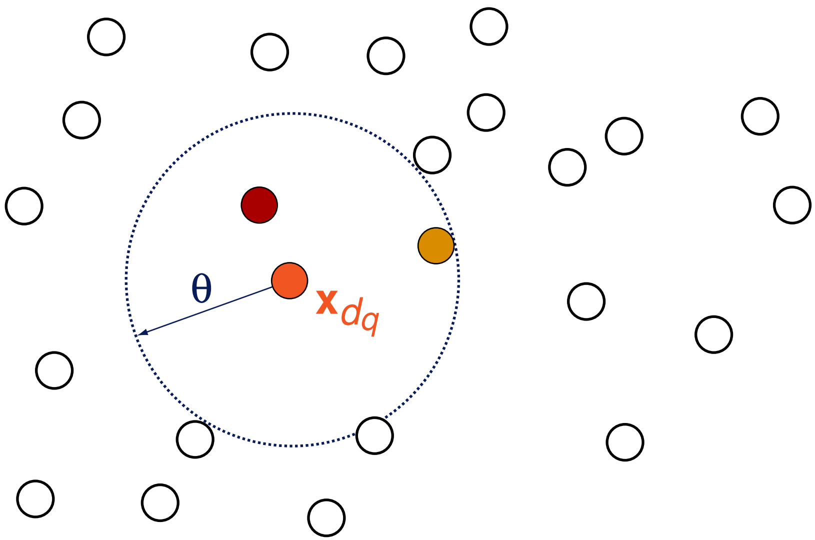
Rationale:

- 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

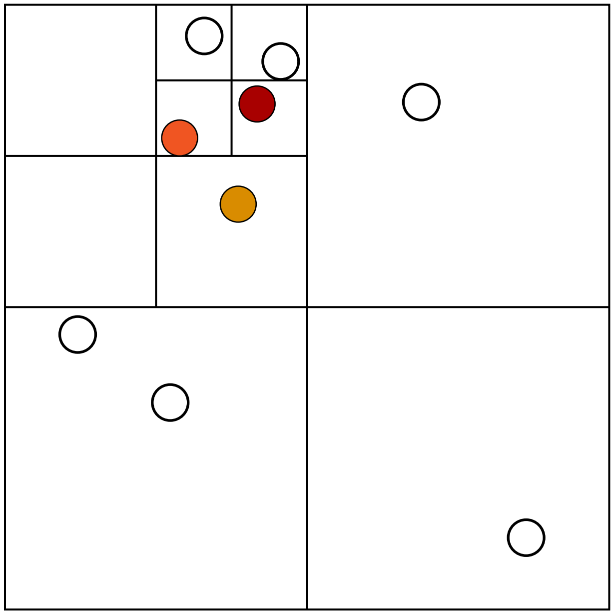


Applications:

- elimination of duplicates / near duplicates
- identification of versioned and plagiarized documents
- retrieval of similar documents
- identification of source code plagiarism

Hash-based Search: Motivation

Nearest Neighbor Search



Indexing with space partitioning methods:

- Quad-tree.

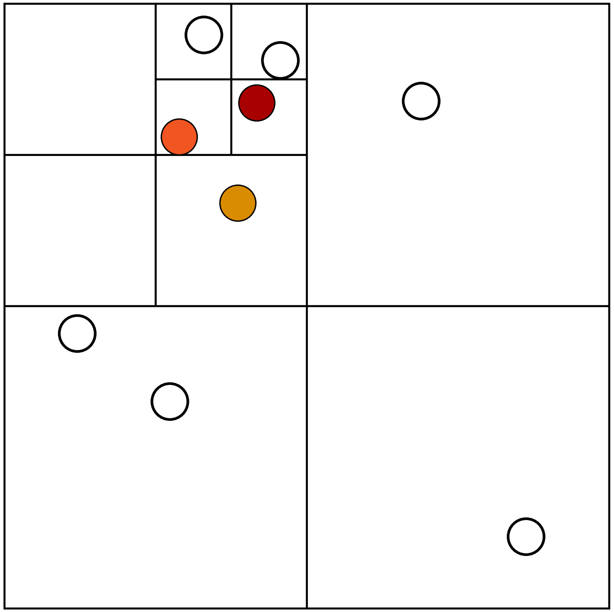
  Split the space recursively into sub-squares until only a few points left.

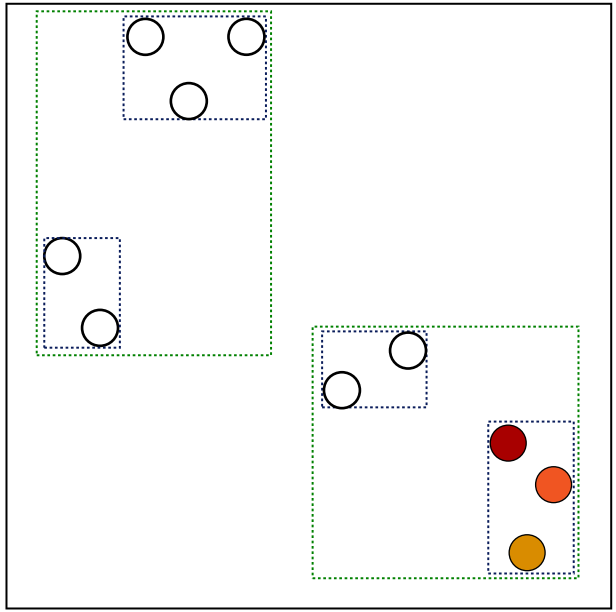
  Space exponential in dimension; time exponential in dimension.

- 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

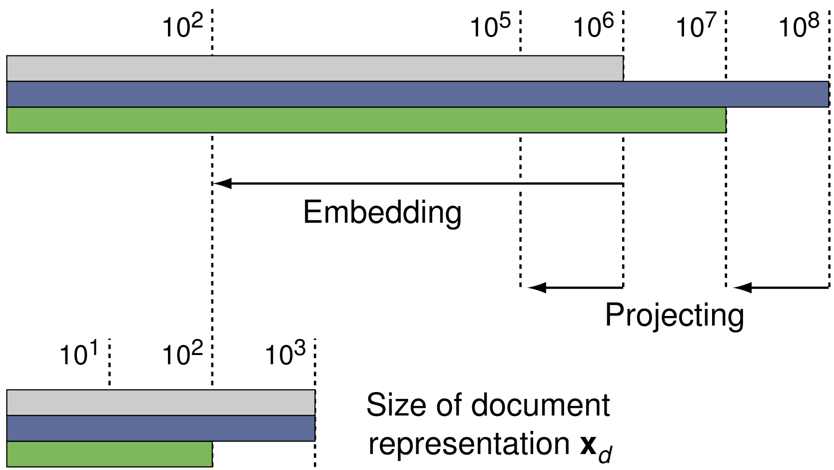
Document Representation and Search

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:

Dictionary	Number of dimensions
1-gram space	3 921 588
4-gram space	274 101 016
8-gram space	373 795 734
Shingling space	75 659 644



Hash-based Search: Motivation

Document Representation and Search

Given the representation $\mathbf{x}_{d_q}$ of a query document and a collection D.

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

$$
\begin{pmatrix} 0.02 \\ 0.0 \\ 0.01 \\ 0.0 \\ 0.0 \\ \vdots \\ 0.0 \\ 0.02 \\ 0.07 \\ 0.0 \end{pmatrix}
\begin{pmatrix} 0.1 \\ 0.2 \\ 0.0 \\ 0.1 \\ 0.2 \\ \vdots \\ 0.1 \\ 0.3 \\ 0.0 \\ 0.0 \end{pmatrix}
\begin{pmatrix} 0.0 \\ 0.1 \\ 0.0 \\ 0.04 \\ 0.0 \\ \vdots \\ 0.0 \\ 0.04 \\ 0.0 \\ 0.03 \end{pmatrix}
\quad \cdots \quad
\begin{pmatrix} 0.07 \\ 0.0 \\ 0.0 \\ 0.1 \\ 0.0 \\ \vdots \\ 0.01 \\ 0.02 \\ 0.03 \\ 0.0 \end{pmatrix}
\begin{pmatrix} 0.0 \\ 0.0 \\ 0.05 \\ 0.1 \\ 0.0 \\ \vdots \\ 0.08 \\ 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}
\begin{pmatrix} 0.02 \\ 0.0 \\ 0.01 \\ 0.0 \\ 0.0 \\ \vdots \\ 0.0 \\ 0.06 \\ 0.09 \\ 0.0 \end{pmatrix}
\begin{pmatrix} 0.0 \\ 0.01 \\ 0.06 \\ 0.0 \\ 0.01 \\ \vdots \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.03 \end{pmatrix}
\begin{pmatrix} 0.04 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.05 \\ \vdots \\ 0.01 \\ 0.02 \\ 0.03 \\ 0.0 \end{pmatrix}
\begin{pmatrix} 0.0 \\ 0.0 \\ 0.01 \\ 0.0 \\ 0.0 \\ \vdots \\ 0.0 \\ 0.02 \\ 0.06 \\ 0.0 \end{pmatrix}
\begin{pmatrix} 0.1 \\ 0.0 \\ 0.09 \\ 0.0 \\ 0.0 \\ \vdots \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.05 \end{pmatrix}
$$

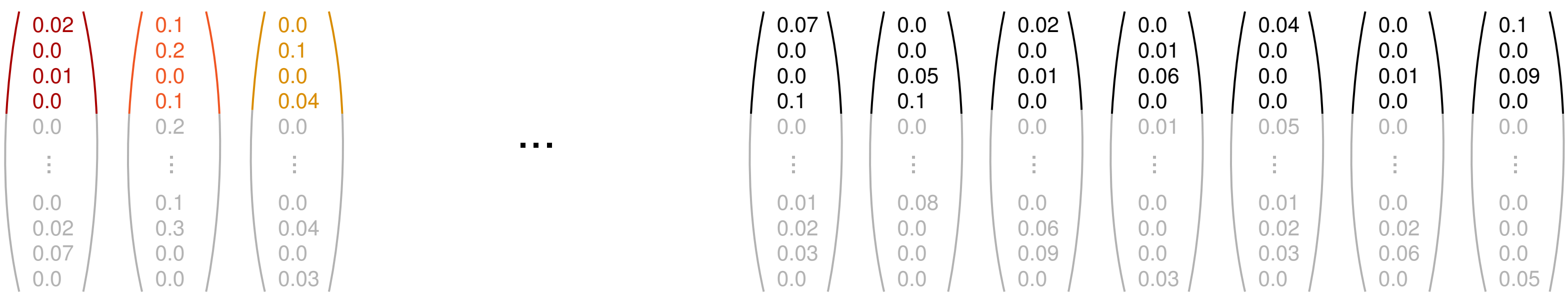
Hash-based Search: Motivation

Document Representation and Search

Given the representation $\mathbf{x}_{d_q}$ 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 $\varphi > 0.5$)



Hash-based Search: Motivation

Document Representation and Search

Given the representation $\mathbf{x}_{d_q}$ of a query document and a collection D.

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

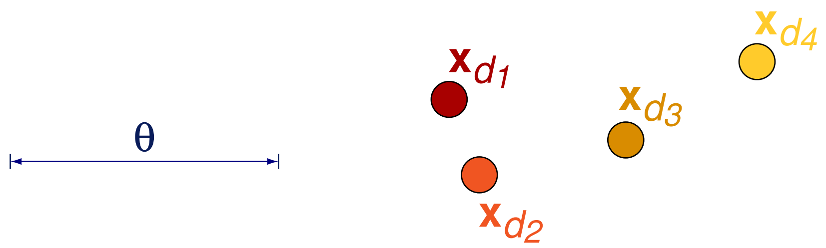
- Linear comparison under some compact representation
  ➜ Acceptable similarity ranking (85% recall at $\varphi > 0.5$)

- Comparison in constant time with a similarity-sensitive hash function h_φ
  ➜ Binary decision wrt. threshold θ (similar if $\varphi > \theta$ / not similar if $\varphi \leq \theta$)

124298	456723	546781		342509	129842	972653	921345	546719	564214	519461
0.02	0.1	0.0		0.07	0.0	0.02	0.0	0.04	0.0	0.1
0.0	0.2	0.1		0.0	0.0	0.0	0.01	0.0	0.0	0.0
0.01	0.0	0.0		0.0	0.05	0.01	0.06	0.0	0.01	0.09
0.0	0.1	0.04		0.1	0.1	0.0	0.0	0.0	0.0	0.0
0.0	0.2	0.0		0.0	0.0	0.0	0.01	0.05	0.0	0.0
⋮	⋮	⋮	...	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0.0	0.1	0.0		0.01	0.08	0.0	0.0	0.01	0.0	0.0
0.02	0.3	0.04		0.02	0.0	0.06	0.0	0.02	0.02	0.0
0.07	0.0	0.0		0.03	0.0	0.09	0.0	0.03	0.06	0.0
0.0	0.0	0.03		0.0	0.0	0.0	0.03	0.0	0.0	0.05

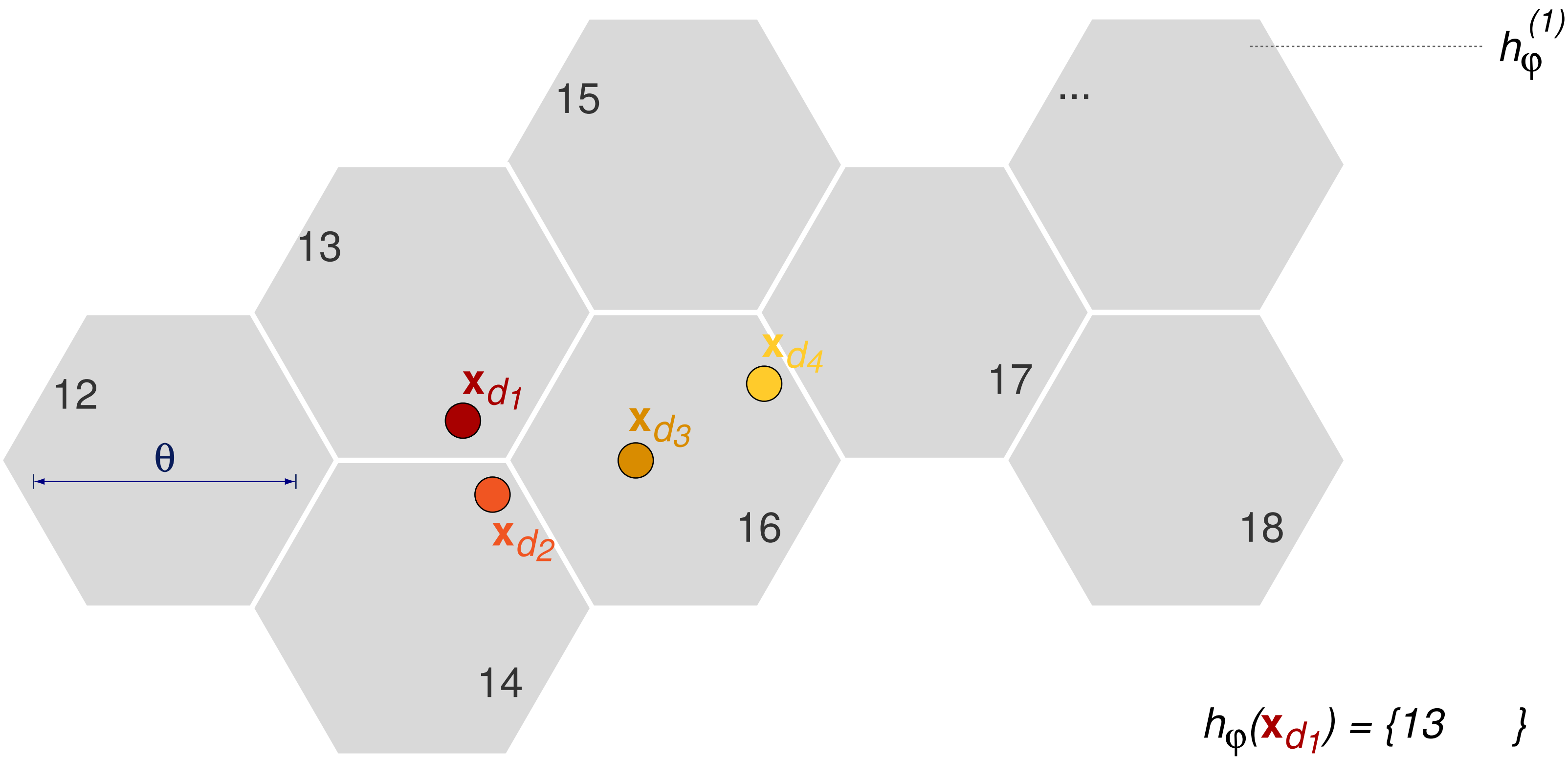
Hash-based Search: Motivation

Hash-based Search is a Space Partitioning Method



Hash-based Search: Motivation

Hash-based Search is a Space Partitioning Method



$h_\varphi(\mathbf{x}_{d_1}) = \{13 \quad \}$

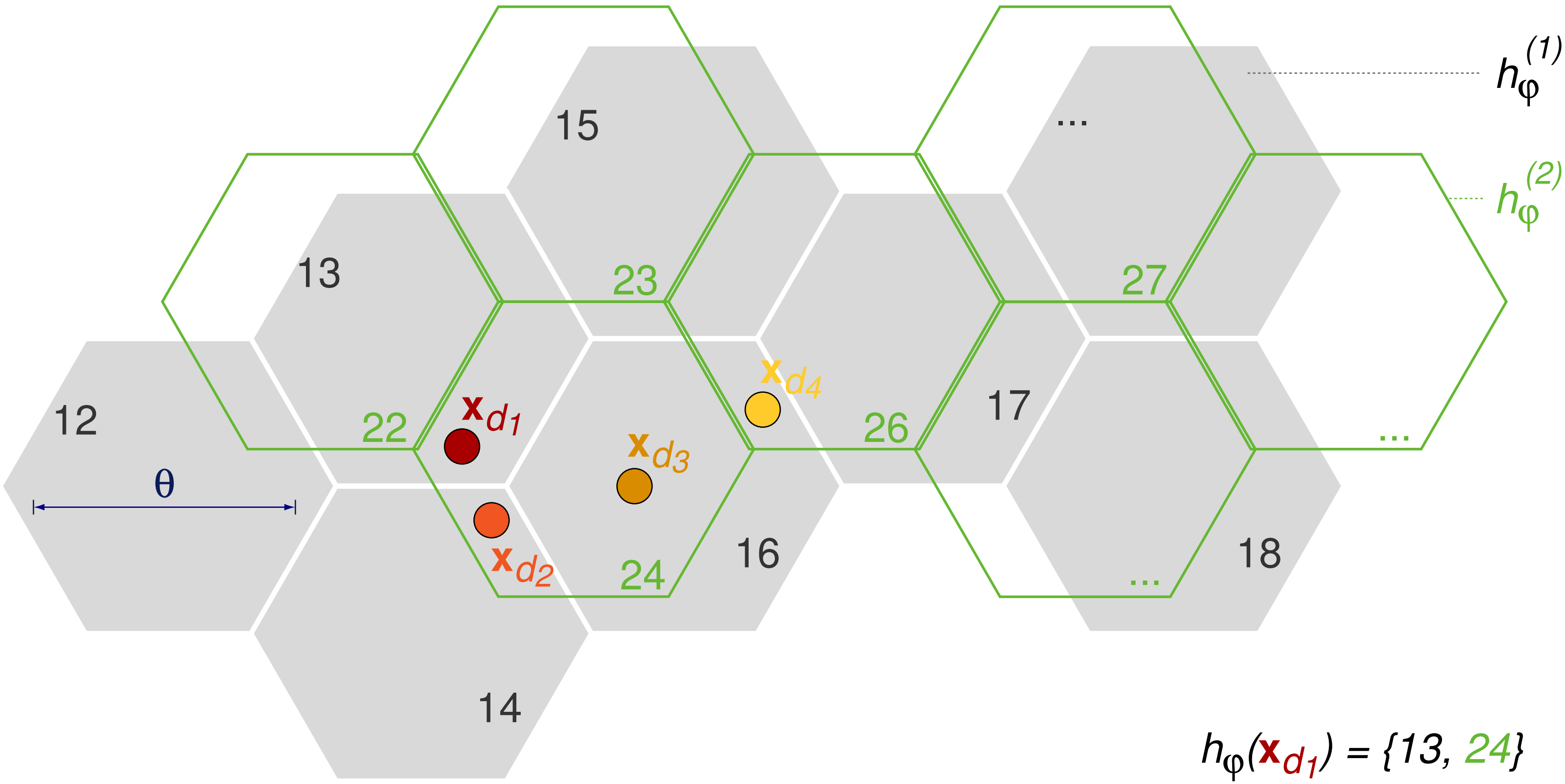
$h_\varphi(\mathbf{x}_{d_2}) = \{14 \quad \}$

$h_\varphi(\mathbf{x}_{d_3}) = \{16 \quad \}$

$h_\varphi(\mathbf{x}_{d_4}) = \{16 \quad \}$

Hash-based Search: Motivation

Hash-based Search is a Space Partitioning Method



$h_\varphi(\mathbf{x}_{d_1}) = \{13, 24\}$

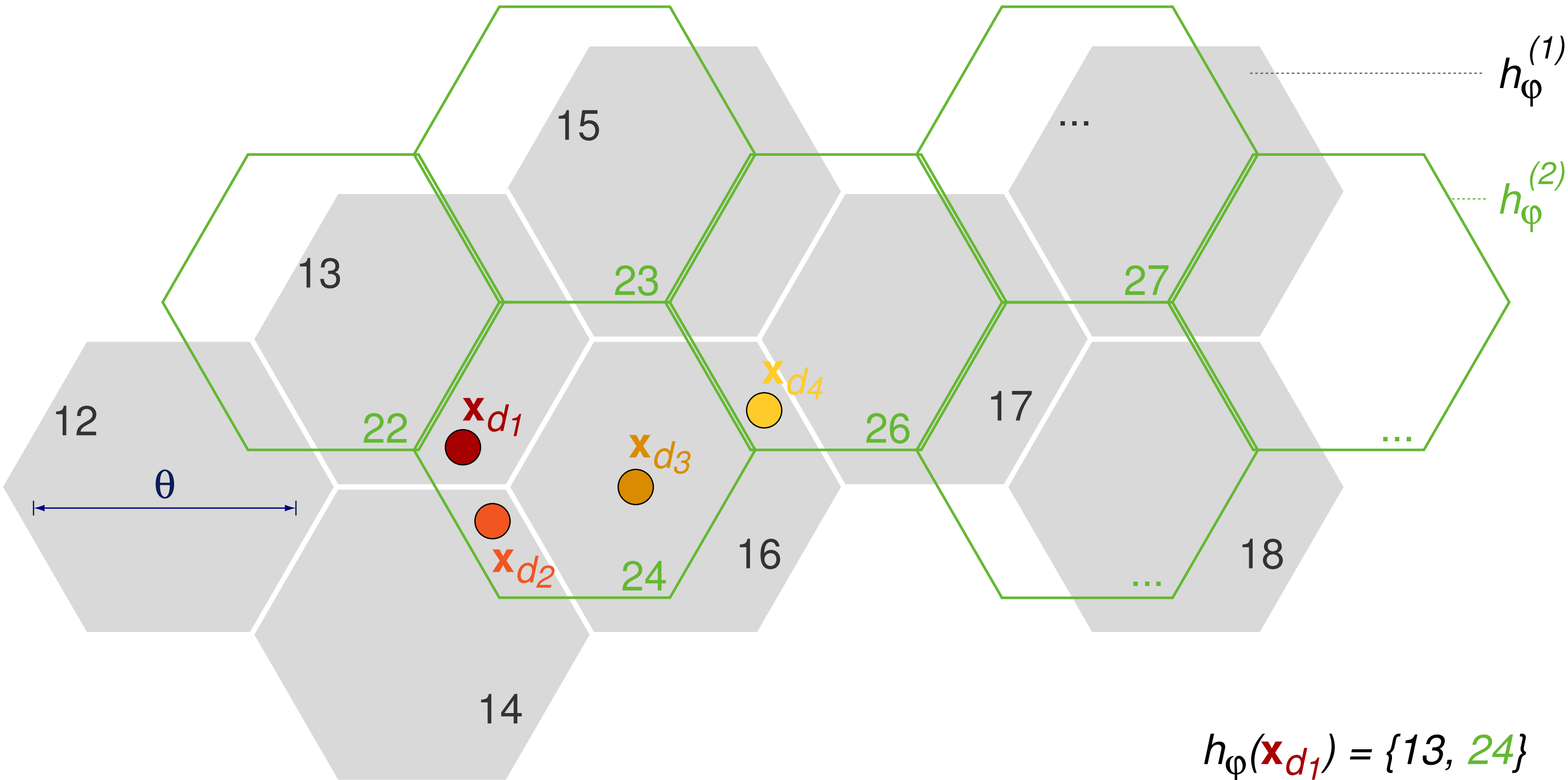
$h_\varphi(\mathbf{x}_{d_2}) = \{14, 24\}$

$h_\varphi(\mathbf{x}_{d_3}) = \{16, 24\}$

$h_\varphi(\mathbf{x}_{d_4}) = \{16, 26\}$

Hash-based Search: Motivation

Hash-based Search is a Space Partitioning Method



$h_\varphi(\mathbf{x}_{d_1}) = \{13, 24\}$

$h_\varphi(\mathbf{x}_{d_2}) = \{14, 24\}$

$h_\varphi(\mathbf{x}_{d_3}) = \{16, 24\}$

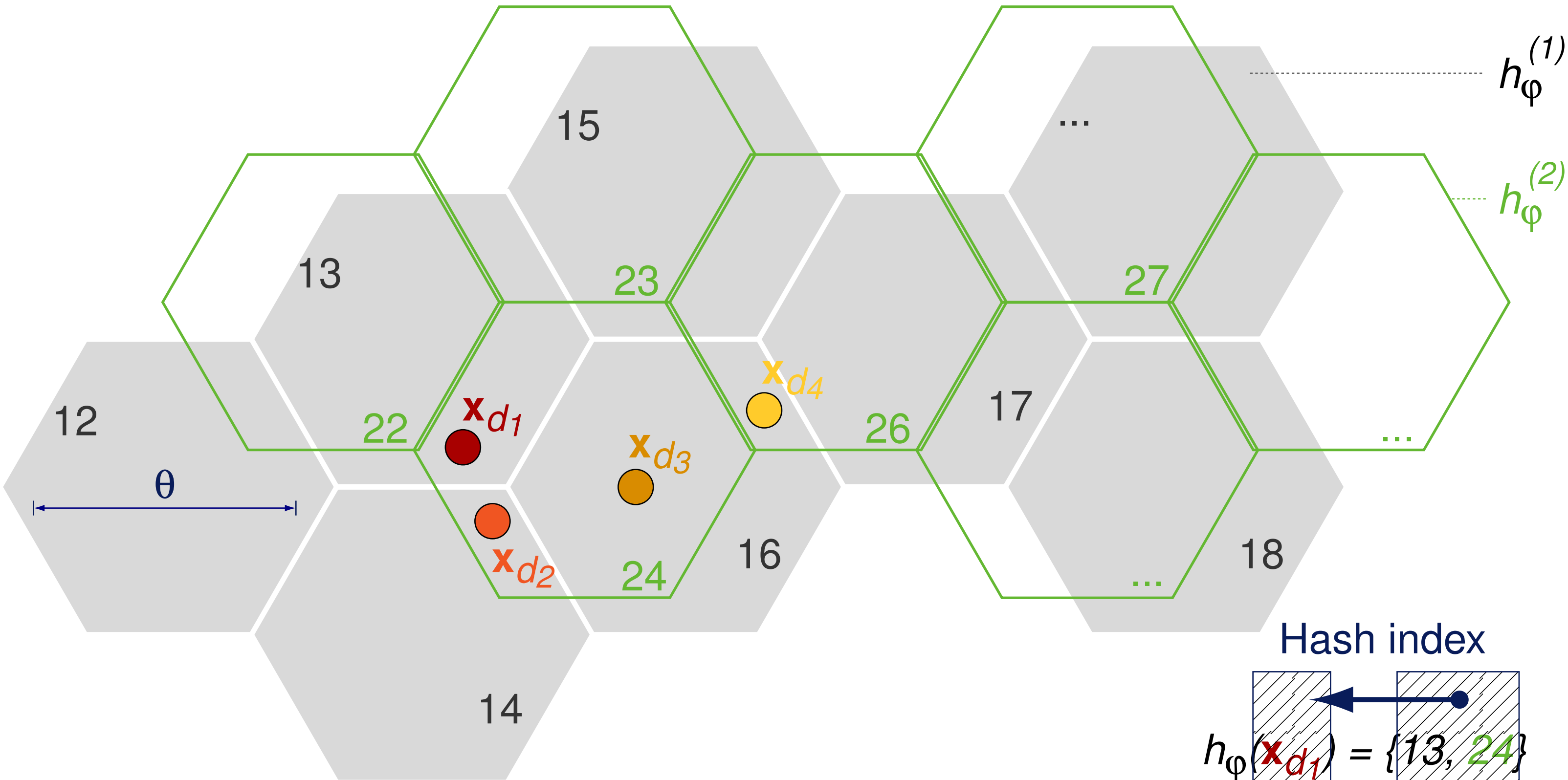
$h_\varphi(\mathbf{x}_{d_4}) = \{16, 26\}$

Similarity collision condition:

$$(\; h^*_\varphi(\mathbf{x}_{d_1}) \cap h^*_\varphi(\mathbf{x}_{d_2}) \;) \neq \emptyset \quad \Leftrightarrow \quad \varphi(\mathbf{x}_{d_1}, \mathbf{x}_{d_2}) > \theta$$

Hash-based Search: Motivation

Hash-based Search is a Space Partitioning Method



Hash index

$h_\varphi(\mathbf{x}_{d_1}) = \{13, 24\}$

$h_\varphi(\mathbf{x}_{d_2}) = \{14, 24\}$

$h_\varphi(\mathbf{x}_{d_3}) = \{16, 24\}$

$h_\varphi(\mathbf{x}_{d_4}) = \{16, 26\}$

Similarity collision condition:

$$\left(h^*_\varphi(\mathbf{x}_{d_1}) \cap h^*_\varphi(\mathbf{x}_{d_2}) \right) \neq \emptyset \quad \Leftrightarrow \quad \varphi(\mathbf{x}_{d_1}, \mathbf{x}_{d_2}) > \theta$$

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_φ.

- Equal codes under h_φ indicate similar objects with a high probability.

  Precision: $h_\varphi(\mathbf{x}_{d_1}) \cap h_\varphi(\mathbf{x}_{d_2}) \neq \emptyset \quad \Rightarrow \quad P(\varphi(\mathbf{x}_{d_1}, \mathbf{x}_{d_2}) > \theta)$ is high

- h_φ maps similar objects on equal codes with a high probability.

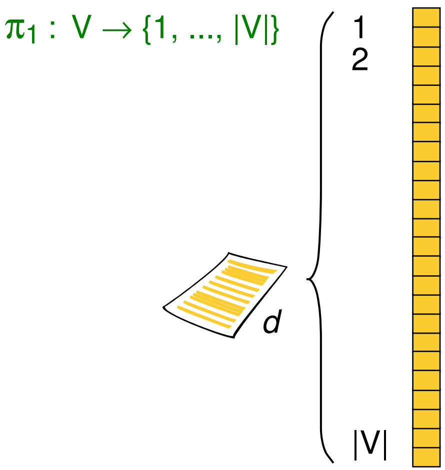
  Recall: $\varphi(\mathbf{x}_{d_1}, \mathbf{x}_{d_2}) > \theta \quad \Rightarrow \quad P(h_\varphi(\mathbf{x}_{d_1}) \cap h_\varphi(\mathbf{x}_{d_2}) \neq \emptyset)$ is high

- h_φ must be multi-valued if D is partly unknown.
- A perfectly similarity-sensitive hash function h_φ^* may exist for each D.

Hash-based Search

Construction Principles for h_φ: Shingling [Broder 2000]

Embedding → Quantization → Encoding

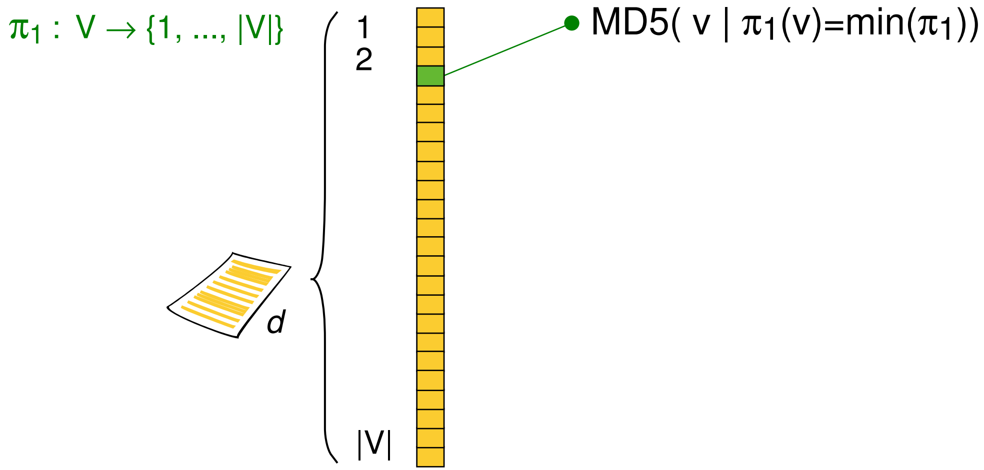


Synchronized random projection

Hash-based Search

Construction Principles for h_φ: Shingling [Broder 2000]

Embedding → **Quantization** → Encoding

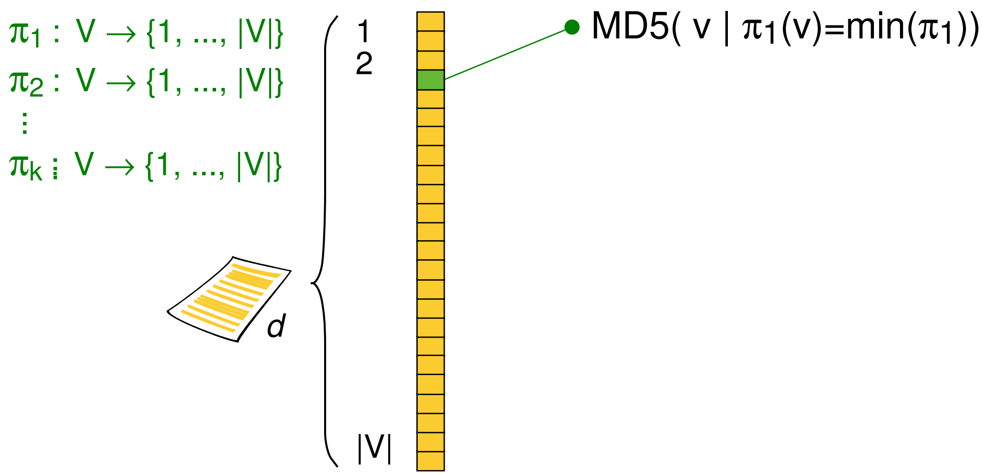


Synchronized random projection

Hash-based Search

Construction Principles for h_φ: Shingling [Broder 2000]

Embedding → Quantization → Encoding

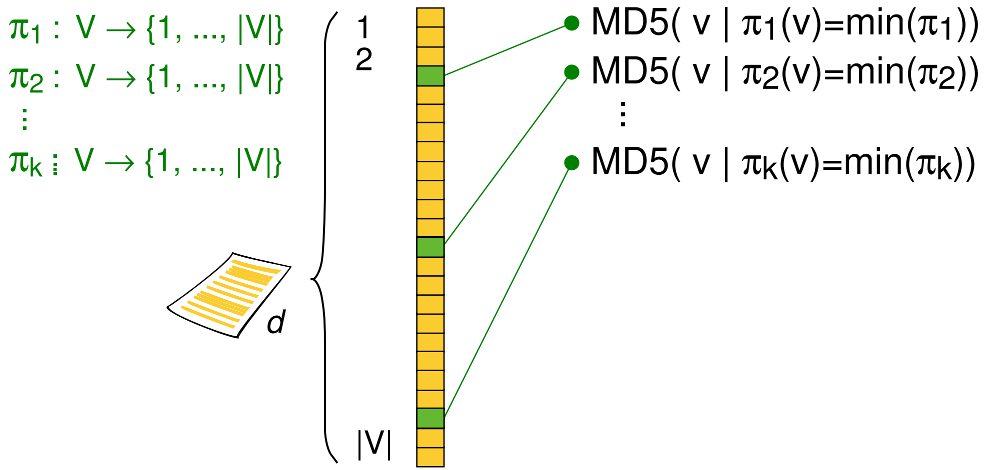


Synchronized random projection

Hash-based Search

Construction Principles for h_φ: Shingling [Broder 2000]

Embedding → **Quantization** → Encoding



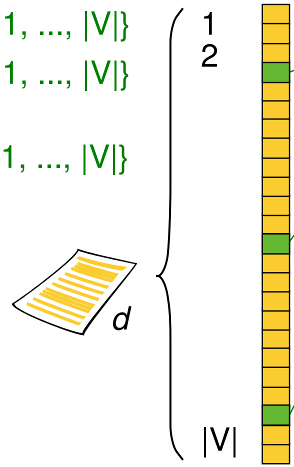
Synchronized random projection

Hash-based Search

Construction Principles for h_φ: Shingling [Broder 2000]

Embedding → Quantization → Encoding

$\pi_1 : V \rightarrow \{1, ..., |V|\}$
$\pi_2 : V \rightarrow \{1, ..., |V|\}$
⋮
$\pi_k : V \rightarrow \{1, ..., |V|\}$



- MD5(v | π_1(v)=min(π_1))
- MD5(v | π_2(v)=min(π_2))
  ⋮
- MD5(v | π_k(v)=min(π_k))

Projection and quantization of MD5 hashes.

Synchronized random projection

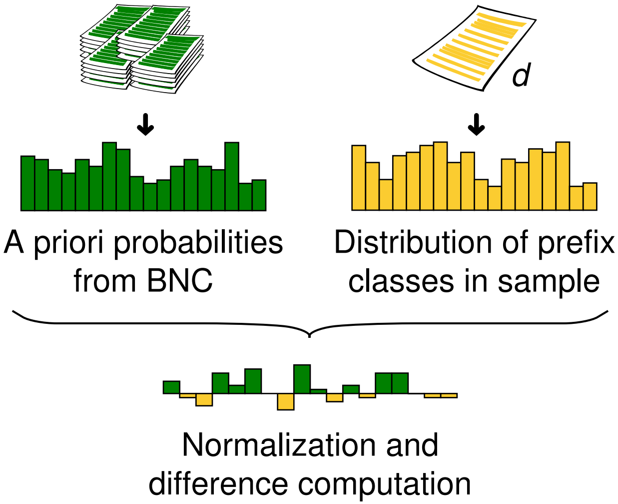
"Super-shingling"

→ Fingerprint = {2643256, 325567} = $h_\varphi(\mathbf{x}_d)$

Hash-based Search

Construction Principles for h_φ: Fuzzy-Fingerprinting

Embedding → Quantization → Encoding

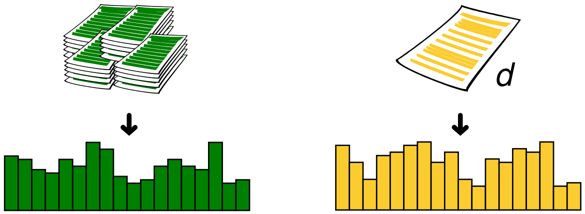


- Documents from the British National Corpus

Hash-based Search

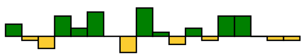
Construction Principles for h_φ: Fuzzy-Fingerprinting

Embedding → **Quantization** → Encoding

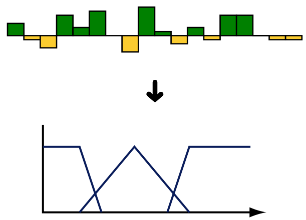


A priori probabilities from BNC

Distribution of prefix classes in sample



Normalization and difference computation



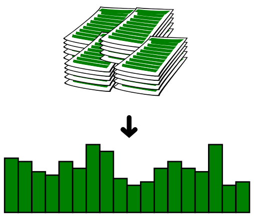
Fuzzification

- Documents from the British National Corpus

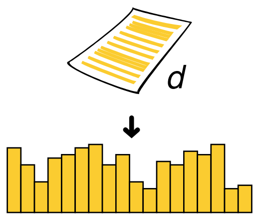
Hash-based Search

Construction Principles for h_φ: Fuzzy-Fingerprinting

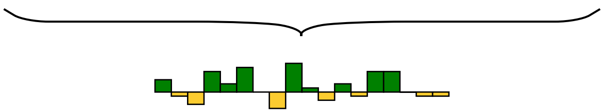
Embedding ➜ Quantization ➜ **Encoding**



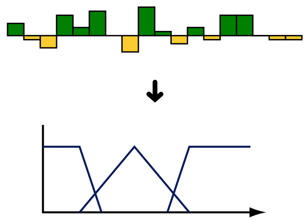
A priori probabilities from BNC



Distribution of prefix classes in sample



Normalization and difference computation



Fuzzification

$$h_\varphi^{(\rho)}(\mathbf{x}_d) = \sum_{i=1}^{k} \rho(y_i) \cdot r^{i-1}$$

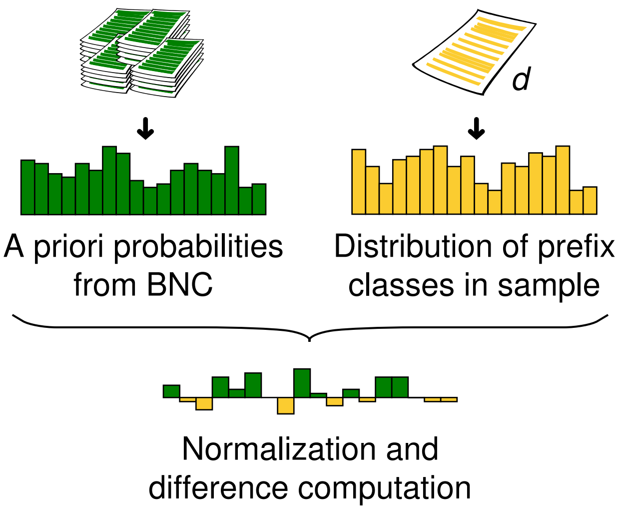
- Documents from the British National Corpus

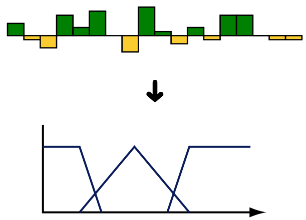
➜ Fingerprint = {2643256,

Hash-based Search

Construction Principles for h_φ: Fuzzy-Fingerprinting

Embedding ➜ **Quantization** ➜ Encoding





Fuzzification

$$h_\varphi^{(\rho)}(\mathbf{x}_d) = \sum_{i=1}^{k} \rho(y_i) \cdot r^{i-1}$$

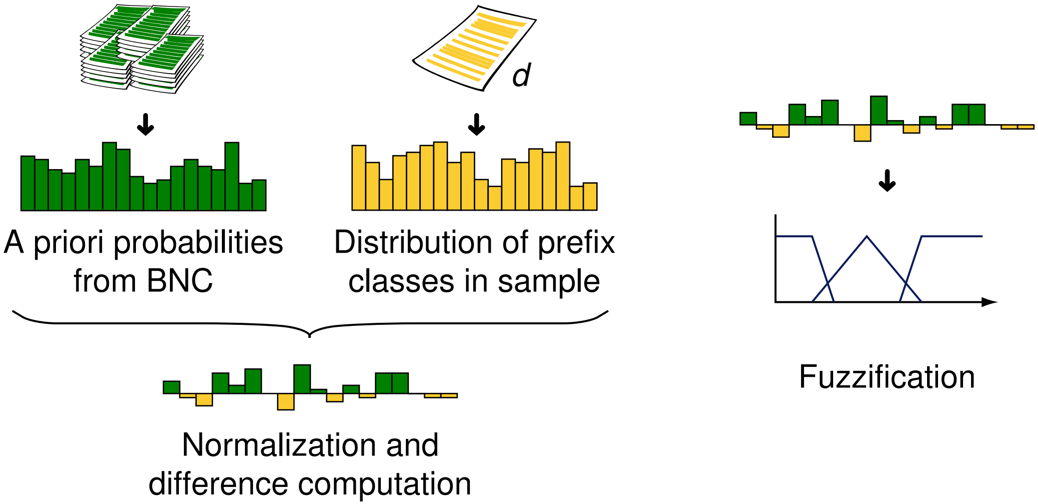
- Documents from the British National Corpus

➜ Fingerprint = {2643256,

Hash-based Search

Construction Principles for h_φ: Fuzzy-Fingerprinting

Embedding → Quantization → **Encoding**



$$h_\varphi^{(\rho)}(\mathbf{x}_d) = \sum_{i=1}^{k} \rho(y_i) \cdot r^{i-1}$$

- Documents from the British National Corpus

➜ Fingerprint = {2643256, 325567} = $h_\varphi(\mathbf{x}_d)$

Hash-based Search

Properties of h_φ

Code length controls precision.

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

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

is increased.

Hash-based Search

Properties of h_φ

Code length controls precision.

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

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

is increased.

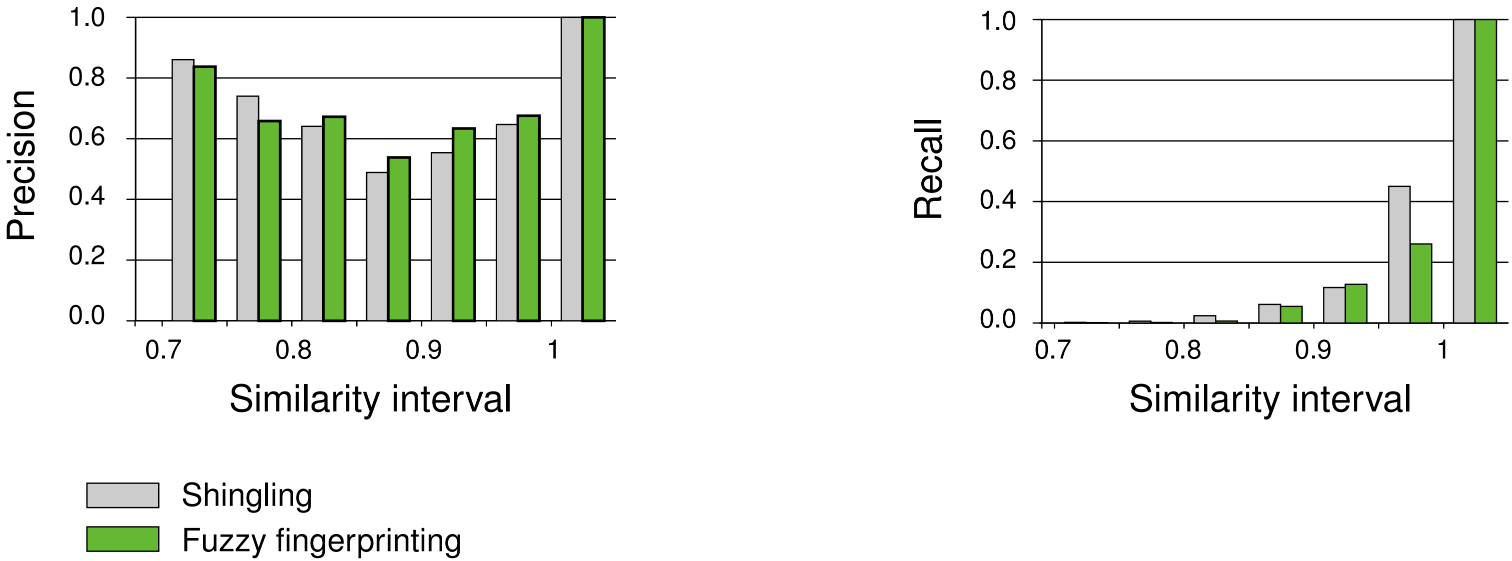
Retrieval Models for Source Code

Fingerprint-based Models

Corpus: as before

Experiment (plot below): 200 queries against fingerprinted corpus

Baseline: greedy string tiling



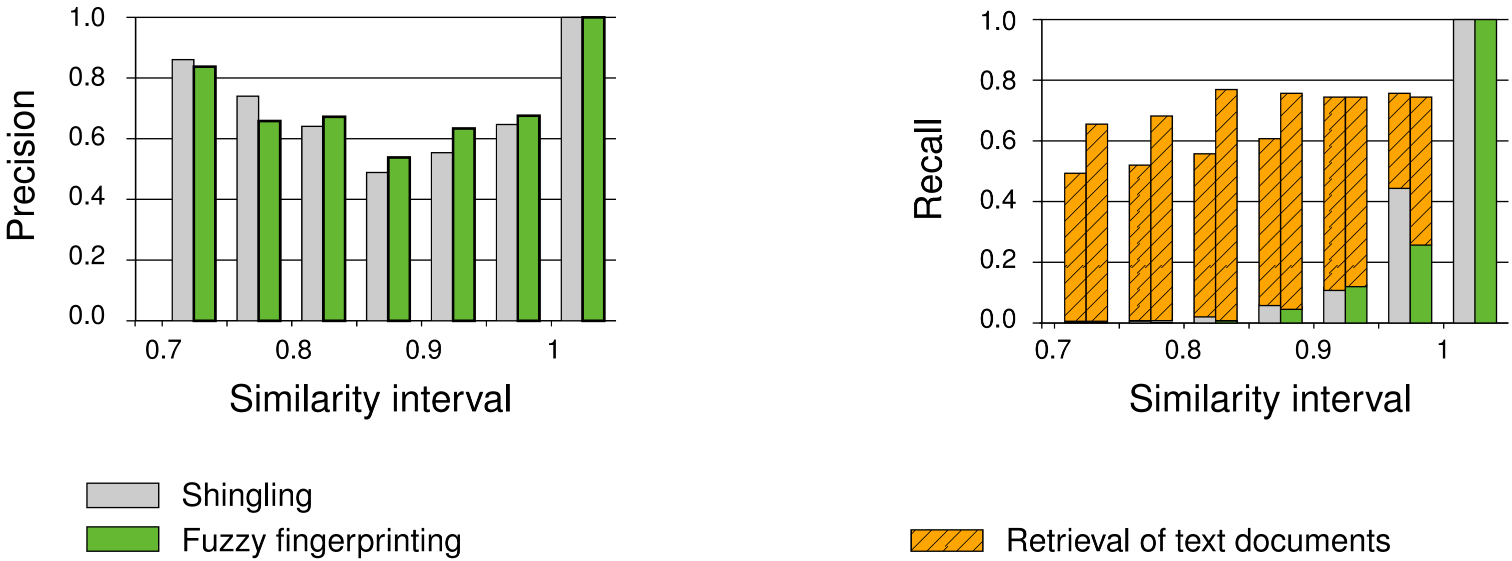
Retrieval Models for Source Code

Fingerprint-based Models

Corpus: as before

Experiment (plot below): 200 queries against fingerprinted corpus

Baseline: greedy string tiling



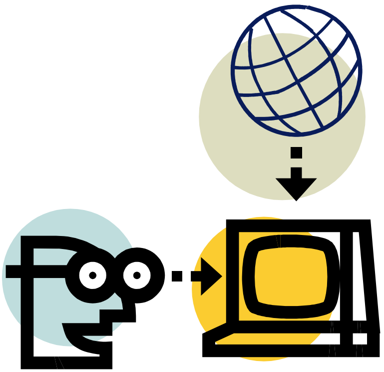
Summary

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
   - φ computation in $O(|\mathbf{d}|^2)$ instead of in $O(|\mathbf{d}|^3)$

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!