Baumeister, Atzmüller (Eds.): Proceedings of the LWA 2008
University of Würzburg, Technical Report 448, pp. 5-12

Retrieval-Technologien für die Plagiaterkennung in Programmen

Fabian Loose, Steffen Becker, Martin Potthast und Benno Stein
Bauhaus-Universität Weimar
99421 Weimar
<vorname>.<nachname>@medien.uni-weimar.de

Abstract

Plagiaterkennung in Programmen (Quellcode) funktioniert analog zu der in Texten: gegeben ist der Quellcode eines Programms d_q sowie eine Kollektion D von Programmquellen. Die Retrieval-Aufgabe besteht darin, in d_q alle Codeabschnitte zu identifizieren, die aus Dokumenten in D übernommen wurden.

Im vorliegenden Papier werden Parallelen und Unterschiede zwischen der Plagiaterkennung in Texten und der in Computerprogrammen aufgezeigt, ein neues Maß zum Ähnlichkeitsvergleich kurzer Code-Abschnitte vorgestellt und erstmalig Fingerprinting als Technologie für effizientes Retrieval aus großen Codekollektionen ($|D| \approx 80.000$) demonstriert. In den von uns durchgeführten Experimenten werden kurze Codeabschnitte aus D, die eine hohe Ähnlichkeit zu Abschnitten aus d_q aufweisen, mit einer Precision von 0.45 bei einem Recall von 0.51 in konstanter Zeit gefunden.

1 Einleitung

Ein Quellcodedokument ist ein Computerprogramm oder ein Teil davon, geschrieben in einer bestimmten Programmiersprache. Ähnlich wie der Autor eines Textes schreibt ein Programmierer den Quellcode seines Programms unter Zuhilfenahme externer Ressourcen, wie zum Beispiel dem World Wide Web. Recherchiert wird unter anderem nach Algorithmen, Programmbibliotheken und Quellcode. Das Web ist in dieser Hinsicht eine reichhaltige Quelle, nicht zuletzt wegen der zahlreichen Open-Source-Projekte, die ihren Quellcode vollständig veröffentlichen [siehe 1].

Es ist üblich, dass ein Programmierer fremden Code in seinen eigenen integriert, um die Entwicklung seines Programms zu beschleunigen. Das geschieht nach dem Motto „das Rad nicht noch einmal zu erfinden", da für häufig auftretende Progammierprobleme robuste Lösungen existieren. Die Benutzung fremden Codes für eigene Zwecke ist jedoch nicht ohne Risiko, denn Quellcode ist urheberrechtlich geschützt. Darüber hinaus wird der Code oftmals nur unter bestimmten Lizenzbedingungen veröffentlicht oder die Innovation, die zur algorithmischen Lösung eines Problems geführt hat, ist patentiert. In diesem Zusammenhang bezeichnen wir die Verwendung von fremdem Code bei Verletzung des Urheberrechts als Plagiat.

Je größer ein Programmierprojekt ist und je mehr Entwickler daran beteiligt sind, desto schwerer ist es, den Überblick über die Verwendung von Programmcode Dritter zu behalten. Das wird zusätzlich erschwert, wenn die Entwicklung eines Programms einem externen Dienstleister überlassen wird (Stichwort: Outsourcing). Auf der einen Seite ergibt sich für den Herausgeber eines Programms daraus das Risiko, für eventuelle Verfehlungen der Entwickler rechtlich belangt zu werden; auf der anderen Seite haben Rechteinhaber ein Interesse daran, Plagiate aufzudecken.

In der vorliegenden Arbeit geben wir einen umfassenden Einblick in die automatische Plagiaterkennung in Quellcode: wir vergleichen sie mit der Plagiaterkennung in Text [siehe 2], stellen ein neues Code-Retrieval-Modell für kurze Codeabschnitte vor, führen erstmals die Anwendung von Fingerprinting auf Quellcode für das Retrieval aus großen Quellcodekollektionen vor und schließen mit einer experimentellen Evaluierung in realistischer Größenordnung.

Im nachfolgenden Unterabschnitt wird die vorliegende Retrieval-Aufgabe diskutiert und ein Überblick über aktuelle Forschungen zur Erkennung von Text- und Quellcodeplagiaten gegeben. Abschnitt 2 gibt einen Überblick über bekannte Retrieval-Modelle für Quellcode aus der Perspektive des Information-Retrieval, Abschnitt 3 gibt eine kurze Einführung in zwei Verfahren zum Fingerprinting und in Abschnitt 4 werden die durchgeführten Experimente beschrieben sowie die Ergebnisse diskutiert.

1.1 Retrieval-Aufgabe und Forschungsüberblick

Die Frage, ob ein Dokument d_q plagiierte Passagen enthält, formulieren wir nach Stein et al. [2007] wie folgt als Retrieval-Aufgabe: Gegeben sei eine Kollektion von Dokumenten D, finde die Passagen s_q aus d_q, für die ein Dokument $d_x \in D$ existiert, das eine Passage s_x enthält, deren Ähnlichkeit $\varphi_{\mathcal{R}}$ zu s_q unter dem Retrieval-Modell $\mathcal{R}$ hoch ist.

Ein Vergleich von d_q mit allen Dokumenten aus D ist nicht effizient, da D üblicherweise groß ist – im Fall von Textdokumenten käme das ganze WWW in Frage. Daher werden Indextechnologien eingesetzt, die das Retrieval von Dokumenten bzw. Passagen aus D erleichtern. Zwei verschiedene Ansätze werden verfolgt, nämlich die Verwendung bestehender Schlüsselwortindexe großer Such-

[1] Vertikale Suchmaschinen wie z.B. Krugle (www.krugle.com) und Google Code (code.google.com) unterstützen Programmierer bei der Suche.

[2] Bei einem Textplagiat beansprucht ein Autor die Urheberschaft für einen Text, der ganz oder in Teilen aus nicht referenzierten Quellen übernommen wurde.

maschinen und die auf die Plagiaterkennung spezialisierte Indizierung von Dokument-Fingerprints (siehe Abbildung 1).

Schlüsselwortindexe werden mit Hilfe von Verfahren zur fokussierten Suche für die Plagiaterkennung in Texten nutzbar gemacht (Si et al. [1997]; Fullam and Park [2002]). Eine kleine Anzahl von Kandidatendokumenten wird aus D extrahiert, die aufgrund ihrer inhaltlichen Ähnlichkeit zu d_q als wahrscheinliche Quellen für Plagiate in Frage kommen. Anschließend werden die Passagen aus d_q mit jeder Passage der extrahierten Dokumente verglichen.

Beim Hashing-basierten Retrieval wird zunächst ein neuer Index für D erzeugt, in dem jedes Dokument als sogenannter Fingerprint repräsentiert wird (Stein [2007]; Potthast [2007]; Broder [2000]; Hoad and Zobel [2003]). Der Fingerprint eines Dokuments ist eine kleine Menge Hashcodes, die so berechnet wird, dass mit großer Wahrscheinlichkeit ähnlichen Dokumenten dieselben Hashcodes zugeordnet werden. Auf diese Weise genügt es, den Fingerprint für d_q zu berechnen und daraufhin mit wenigen Index-Lookups alle ähnlichen Dokumente aus D zu extrahieren. Werden Fingerprints für alle Passagen in D indiziert, lassen sich unmittelbar alle verdächtigen Passagen in d_q identifizieren.

Bei der intrinsischen Analyse werden Plagiate aufgrund von Brüchen im Schreibstil in d_q identifiziert, der mit Hilfe einer Reihe von Merkmalen wie zum Beispiel der durchschnittlichen Satzlänge oder der Anzahl von Funktionsworten pro Satz etc. quantifiziert werden kann (Meyer zu Eissen and Stein [2006]). Weicht der Schreibstil einer Passage aus d_q statistisch signifikant vom durchschnittlichen Schreibstil des gesamten Dokuments ab, so liegt der Verdacht nahe, dass die Passage plagiiert wurde.

Die bisherige Forschung zur Plagiaterkennung in Quellcode befasst sich ausschließlich mit einem speziellen Anwendungsfall: die Plagiaterkennung in kleinen Kollektionen studentischer Programmierübungen. In diesem Anwendungsfall ist das Retrieval aus größeren Quellcodekollektionen unnötig, so dass sich die meisten Arbeiten nur darauf konzentrieren, Retrieval-Modelle für den detaillierten Vergleich von Quellcodedokumenten zu entwickeln. Der nächste Abschnitt gibt einen Überblick über diese Modelle und verweist auf die einschlägigen Arbeiten. Indexbasiertes Retrieval für Quellcode wird einzig in der Arbeit von Burrows et al. [2006] betrachtet. Hier wird eine Art Schlüsselwortindex eingesetzt, um, ähnlich wie im oben genannten heuristischen Retrieval, zu d_q ähnliche Code-Dokumente aus D zu extrahieren. Fingerprint-Indexe sind diesem Ansatz sowohl vom notwendigen Speicheraufwand als auch vom Retrieval-Aufwand her überlegen. Eine auf Stilmerkmalen basierende intrinsische Plagiaterkennung wurde für Quellcode bislang nicht in Betracht gezogen.

2 Retrieval-Modelle für Quellcode

Im Information-Retrieval wird ein reales Dokument d durch ein Modell $\mathbf{d}$ repräsentiert. Für Dokumentrepräsentationen wird außerdem ein Ähnlichkeitsmaß φ definiert, das die inhaltliche Ähnlichkeit zweier Dokumente quantifiziert. Dokumentrepräsentation und Ähnlichkeitsmaß sind Teil des Retrieval-Modells $\mathcal{R}$.

Die Dokumentrepräsentation soll den Inhalt des realen Dokuments in einer für die vorliegende Retrieval-Aufgabe geeigneten Form abbilden. Das beinhaltet ein Abwägen zwischen einer möglichst exakten Repräsentation auf der einen Seite und geringer Komplexität, die Repräsentation zu erzeugen und Ähnlichkeiten zu berechnen auf der anderen. Eine Repräsentationsform zu finden, die eine hohe Retrieval-Qualität bei geringem Aufwand ermöglicht, ist eine der wichtigsten Aufgaben im Information-Retrieval.

Tabelle 1 gibt einen Überblick über die bislang für eine Plagiaterkennung in Quellcode vorgeschlagenen Retrieval-Modelle. Um eine Einschätzung einzelner Modelle zu ermöglichen, erläutern wir für jedes Modell, welche Anforderungen an die Kompilierbarkeit an ein Programm d gestellt werden, und mit welcher Laufzeitkomplexität die Ähnlichkeit zwischen zwei Code-Dokumenten berechnet werden kann.

Die Kompilierbarkeit von d wird anhand der aus dem Compilerbau bekannten Konzepte bewertet, die zur Beschreibung der Übersetzung eines Dokuments aus seiner Programmiersprache in die Sprache der Zielplattform benutzt werden. Es gibt drei aufeinander aufbauende Stufen, die lexikalische, die syntaktische und die semantische Analyse. Bei der lexikalischen Analyse wird der Quellcode in Token unterteilt. Es handelt sich dabei um das Vokabular der Programmiersprache, wobei Schlüsselworte, Sonderzeichen oder frei benannte Variablen möglich sind. Jedes Token ist von einem bestimmten Typ; Beispiele für Token sind in Tabelle 2 gelistet. Das Ergebnis dieses Analyseschritts ist ein Token-String. In der syntaktischen Analyse wird überprüft, ob der tokenisierte Quellcode wohlgeformt ist, d.h. ob der Code konform zur Grammatik der Programmiersprache ist. Das Ergebnis dieses Analyseschritts ist ein Syntaxbaum. Im letzten Analyseschritt, der semantischen Analyse, wird die statische Semantik des Codes geprüft, also z.B. ob die Typen von Variablen mit denen der zugeordneten Werte übereinstimmen. Das Resultat ist ein attributierter Syntaxbaum, der Informationen über die statische Semantik enthält, und der als Grundlage zur Übersetzung des Quellcodes in Maschinensprache dient.

Die Laufzeitangaben in der Tabelle beruhen größtenteils auf Worst-Case-Angaben im referenzierten Papier. Spezialisierte Heuristiken erlauben hier zum Teil deutliche Verbesserungen.

Um den Inhalt eines Quellcodedokuments zu repräsentieren genügt es nicht, das Dokument als eine Menge von Token zu betrachten. Das steht im Gegensatz zu Textdo-

Tabelle 2: Abstrakte Code-Token und ihre Häufigkeit.

Token-Typ	Beispiel	Häufigkeit (%)
APPLY	`System.out.print('!!');`	32.66
VARDEF	`int i; String text;`	12.12
ASSIGN	`i = i+5; i++; i += 5;`	8.82
BEGINIF	`if (test) {`	6.19
ELSE	`} else {`	1.68
ENDIF	`} // end of 'if'`	5.55
NEWCLASS	`test = new A();`	5.27
BEGINFOR	`for (i=0; i<n; i++) {`	0.93
ENDFOR	`} // end of 'for'`	0.93
BEGINSWITCH	`switch (expr) {`	0.13
ENDSWITCH	`} // end of 'switch'`	0.13
CASE	`case 5:, default:`	1.04
BEGINWHILE	`while (condition) {`	0.36
ENDWHILE	`} // end of 'while'`	0.36
BEGINDO	`do {`	0.02
ENDDO	`} while(condition);`	0.02
(21 weitere in Prechelt et al. [2000])		

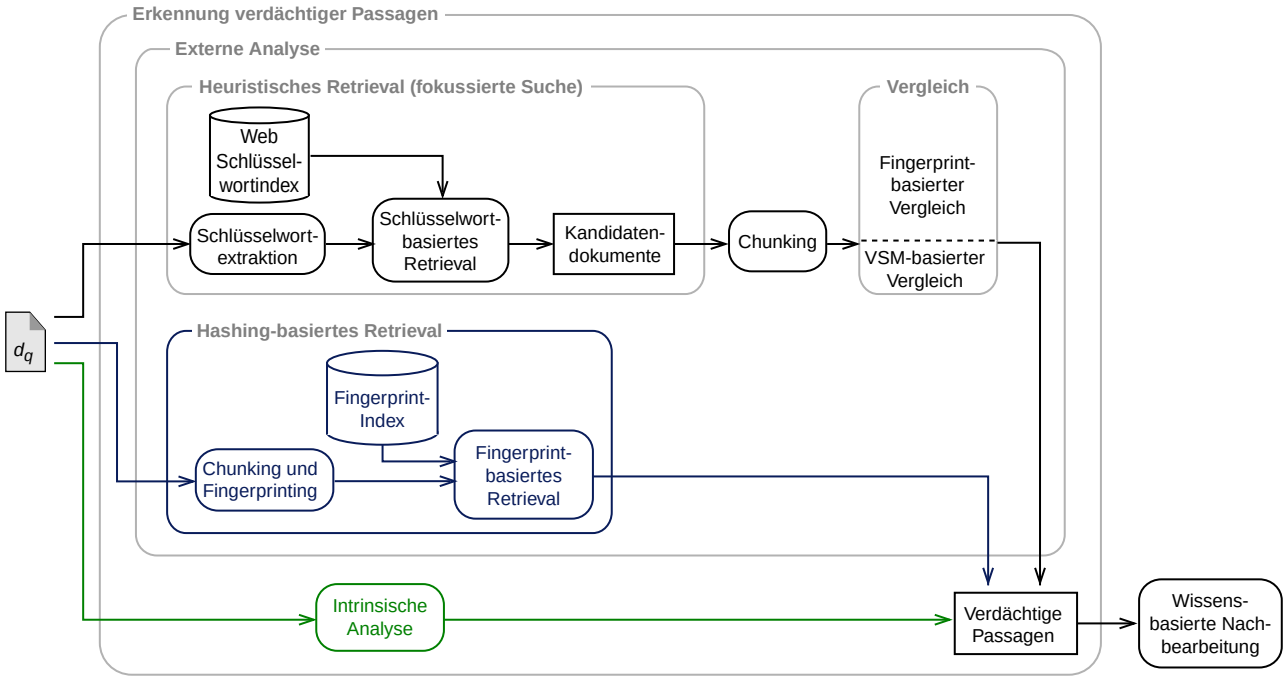

Abbildung 1: Gegenüberstellung verschiedener Ansätze zur Plagiaterkennung. Grundsätzlich ist dieses Vorgehen sowohl für Textdokumente als auch für Programmquellen denkbar, jedoch variiert der Aufwand zur Lösung der Einzelaufgaben bei Text und Code.

kumenten, bei denen die Bag-of-Words-Repräsentation in vielen Fällen hinreichend ist. Bei Quellcode ist das Vokabular stark eingeschränkt, so dass ein typisches Dokument schon fast das gesamte Vokabular der Sprache enthalten kann. Daher werten selbst die einfachsten Quellcoderepräsentationen Informationen über die Struktur des Quellcodes aus. Wir unterscheiden drei Gruppen von Code-Retrieval-Modellen: (1) attributbasierte Vektormodelle, (2) strukturbasierte String-Modelle und (3) strukturbasierte Graphenmodelle.

Attributbasierte Vektormodelle. Ein Dokument d wird als Vektor $\mathbf{d}$ repräsentiert, dessen Dimensionen Token-n-Gramme sind, $n < 5$. Eine weitere Form attributbasierter Vektormodelle berechnet für d eine Reihe von Softwaremetriken, die dazu dienen, die Komplexität eines Programms, den Kontrollfluss oder den Programmierstil zu quantifizieren. Nach Verco and Wise [1996] ist letzteres Modell jedoch untauglich zur Quantifizierung von Ähnlichkeit. Stattdessen könnte es sich aber in der Plagiaterkennung für die intrinsische Analyse eignen.

Strukturbasierte String-Modelle. Diese Modelle wählen den Token-String, der das Ergebnis der lexikalischen Analyse beim Kompiliervorgang ist, als Repräsentation für ein Dokument d. Dabei werden die konkreten Token in d durch ihre Typbezeichnung ersetzt (vgl. Tabelle 2).

Token-Strings sind eine weit verbreitete Repräsentationsform für Quellcode und eine Reihe von Ähnlichkeitsmaßen wurden für sie vorgeschlagen. Dabei beruhen die Maße Shared-Information, Greedy-String-Tiling und Local-Alignment jeweils auf derselben Annahme, dass inhaltlich ähnliche Token-Strings längere Sub-Strings gemein haben. Das trifft für hinreichend lange Token-Strings im Allgemeinen zu, doch je kürzer die verglichenen Token-Strings sind, desto seltener ist das tatsächlich der Fall. Einschübe im Code oder umsortierte Befehle sind im Hinblick auf Plagiate bei ansonsten inhaltlich gleichen Code-Schnipseln oft zu beobachten. Die vorgenannten Ähnlichkeitsmaße degenerieren hier, da die Ähnlichkeitsberechnung nicht mehr feingranular genug ist.

Für kurze Quellcodedokumente schlagen wir daher ein neues Ähnlichkeitsmaß vor, dass auf der Berechnung der Longest-Common-Subsequence zweier Token-Strings beruht. Im Gegensatz zum längsten gemeinsamen Sub-String können bei der längsten gemeinsamen Teilfolge zweier Strings ungleiche Stellen übersprungen werden. Beispielsweise ist für die Token-Strings $\mathbf{s}_q$ = ”ABCD” und $\mathbf{s}_x$ = ”ACED” der längste gemeinsame Sub-String wahlweise ”A”, ”B”, ”C” oder ”D”, während die längste gemeinsame Teilfolge ”ACD” ist. Letzteres kommt im Falle von Token-Strings einer realistischeren Einschätzung der Ähnlichkeit des zugrunde liegenden Quellcodes näher als Sub-Strings. Die Ähnlichkeit zweier Token-Strings berechnen wir auf Grundlage dieser Idee wie folgt:

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

wobei $\text{lcs}(\mathbf{s}_q, \mathbf{s}_x)$ die längste gemeinsame Teilfolge von $\mathbf{s}_q$ und $\mathbf{s}_x$ bezeichnet.

Strukturbasierte Graphenmodelle. Vertreter dieser Retrieval-Modelle repräsentieren ein Dokument d als Syntaxbaum oder als attributierten Syntaxbaum. Folglich werden größere Anforderungen an die Kompilierbarkeit von d gestellt. Im Verhältnis zu den oben genannten Repräsentationen werden Dokumente hier deutlich exakter repräsentiert, was sich auch auf den Aufwand der Ähnlichkeitsberechnung auswirkt. Eine Ausnahme diesbezüglich ist das Modell von Baxter et al. [1998], bei dem eine auf Hashing basierende Strategie zur Identifikation ähnlicher Teilbäume in den Syntaxbäumen zweier Dokumente eingesetzt wird.

Wir gehen davon aus, dass in der Plagiaterkennung dem detaillierten Vergleich von Quellcodedokumenten ein heuristisches Retrieval voraus geht (vgl. Abbildung 1). Daher sind für diese Aufgabe Retrieval-Modelle zu präferieren, die nur geringe Anforderungen an die Kompilierbar-

Tabelle 1: Retrieval-Modelle für Quellcode.

Code-Retrieval-Modell $\mathcal{R}$		**Kompilierbarkeit**	**Laufzeit**	**Referenz**
Dokumentrepräsentation $\mathbf{d}$	Ähnlichkeitsmaß φ	von d	von φ	
Attributbasierte Vektormodelle			$\lvert\mathbf{d}\rvert$ = Dimensionalität von $\mathbf{d}$	
Softwaremetriken	Kosinusähnlichkeit	–	$O(\lvert\mathbf{d}\rvert)$	Ottenstein [1976], Grier [1981]
n-Gramme des Token-Strings	Jaccard-Koeffizient oder Kosinusähnlichkeit	lexikalisch	$O(\lvert\mathbf{d}\rvert)$	Clough et al. [2002]
Winnowing: Auswahl von n-Grammen des Token-Strings	Jaccard-Koeffizient oder Kosinusähnlichkeit	lexikalisch	$O(\lvert\mathbf{d}\rvert)$	Schleimer et al. [2003]
Strukturbasierte String-Modelle			$\lvert\mathbf{d}\rvert$ = Tokenlänge von $\mathbf{d}$	
Token-String	Shared-Information: Kompressionsrate zweier Token-Strings zusammen.	lexikalisch	$O(\lvert\mathbf{d}\rvert^2)$	Chen et al. [2004]
Token-String	Greedy-String-Tiling: Summe der längsten gemeinsamen Sub-Strings.	lexikalisch	$O(\lvert\mathbf{d}\rvert^3)$	Prechelt et al. [2000]
Token-String	Local-Alignment: Summe der längsten gemeinsamen Sub-Strings, die ungleiche Stellen besitzen dürfen.	lexikalisch	$O(\lvert\mathbf{d}\rvert^2)$	Burrows et al. [2006]
Token-String	Longest-Common-Subsequence: Längste gemeinsame Teilfolge von Token.	lexikalisch	$O(\lvert\mathbf{d}\rvert^2)$	siehe Abschnitt 2
Strukturbasierte Graphenmodelle			$\lvert\mathbf{d}\rvert$ = Knotenanzahl in $\mathbf{d}$	
Abstract-Syntax-Trees	Hashing-basierte Suche nach längsten gleichen Teilbäumen.	syntaktisch	$O(\lvert\mathbf{d}\rvert)$	Baxter et al. [1998]
Conceptual-Graphs	Spezialisierte Suche nach isomorphen Teilgraphen.	semantisch	$O(\lvert\mathbf{d}\rvert^3)$	Mishne and de Rijke [2004]
Program-Dependence-Graphs	Suche nach isomorphen Teilgraphen.	semantisch	NP-vollständig	Liu et al. [2006]

keit der Dokumente stellen. Der Grund dafür ist, dass im Web Quellcode oftmals in Form von Code-Schnipseln zu finden ist, die unvollständig bezüglich der Grammatik der Programmiersprache sind, und dass die benötigten Bibliotheken zur semantischen Analyse nicht vorliegen. Es ist zu beachten, dass bezüglich des heuristischen Retrievals von Quellcode unter Zuhilfenahme existierender (vertikaler) Suchmaschinen bislang keinerlei Forschung existiert.

In einer geschlossenen Retrieval-Situation, in der alle zu vergleichenden Dokumente im voraus bekannt sind (z.B. beim Vergleich studentischer Programmierübungen), kommen nur das Graphenmodell von Baxter et al. [1998] oder das von uns vorgeschlagene Token-String-Modell in Frage. Ersteres bietet eine hohe Retrieval-Qualität bei einer guten Vergleichslaufzeit, während letzteres vor allem auf den Vergleich kurzer Code-Abschnitte spezialisiert ist.

3 Fingerprinting für Quellcode-Retrieval

Mit Fingerprinting wird ein Verfahren bezeichnet, bei dem ähnlichkeitssensitives Hashing eingesetzt wird, um ähnliche Dokumente zu identifizieren ohne sie direkt miteinander zu vergleichen.

Nach Stein [2007] ist der Ausgangspunkt dafür ein Retrieval-Modell $\mathcal{R}$ und eine speziell konstruierte Hashing-Funktion $h_\varphi^{(\rho)} : \mathbf{D} \rightarrow \mathbf{N}$, die die Menge der Dokumentrepräsentationen $\mathbf{D}$ unter $\mathcal{R}$ auf die natürlichen Zahlen abbildet. Dabei kann $h_\varphi^{(\rho)}$ mit einem Parameter ρ variiert werden, der abhängig vom eingesetzen Hashing-Verfahren ist.

Mit einer Schar $h_\varphi^{(\Pi)}$ von Hashing-Funktionen, parametrisiert mit einer Menge unterschiedlicher Parameter Π, lässt sich die Ähnlichkeitsfunktion φ aus $\mathcal{R}$ wie folgt nachbilden:

$$h_\varphi^{(\Pi)}(\mathbf{d}_q) \cap h_\varphi^{(\Pi)}(\mathbf{d}_x) \neq \emptyset \quad \Leftrightarrow \quad \varphi(\mathbf{d}_q, \mathbf{d}_x) \geq 1 - \varepsilon,$$

wobei $h_\varphi^{(\Pi)}(\mathbf{d}_q)$ die Menge der Hashcodes von $\mathbf{d}_q$ ist, die alle Hashcodes $h_\varphi^{(\rho)}(\mathbf{d}_q)$ mit $\rho \in \Pi$ enthält. ε bezeichnet einen Ähnlichkeitsradius mit $0 < \varepsilon \ll 1$.

$h_\varphi^{(\Pi)}(\mathbf{d}_q)$ wird als Fingerprint von $\mathbf{d}_q$ bezeichnet. Die Indizierung von Fingerprints geschieht mit Hilfe eines Inverted-File-Index, der Hashcodes auf Postlisten von Dokumenten abbildet, deren Fingerprints den jeweiligen Hashcode teilen. Folglich können, ausgehend vom Fingerprint $h_\varphi^{(\Pi)}(\mathbf{d}_q)$, alle zu d_q ähnlichen Dokumente in D in konstanter Zeit gefunden werden.

Im Allgemeinen berechnen ähnlichkeitssensitive Hashing-Funktionen $h_\varphi^{(\rho)}$ den Hashcode für ein Dokument in drei Schritten (Stein and Potthast [2007]):

(1) *Dimensionsreduktion* $\mathbf{d} \Rightarrow \mathbf{d}'$. Die Reduzierung der Dimensionalität der Dokumentrepräsentationen $\mathbf{d}$ unter $\mathcal{R}$ dient in erster Linie der Reduzierung der möglichen Freiheitsgrade, da ähnlichkeitssensitives Hashing ein Verfahren der Raumpartitionierung ist, und

Tabelle 3: Quellcode-Fingerprinting mit (Super-)Shingling und Fuzzy-Fingerprinting.

Hashing-Funktion $h_\varphi^{(\rho)}$	**(Super-)Shingling**	**Fuzzy-Fingerprinting**
Code-Retrieval-Modell $\mathcal{R}$	$\mathbf{d}$: n-Gramme des Token-Strings φ: Jaccard-Koeffizient	$\mathbf{d}$: n-Gramme des Token-Strings mit Gewichtung φ: Kosinusähnlichkeit
Parameter ρ	Menge von Zufallspermutationen $\{\pi_1, \ldots, \pi_k\}$, wobei $\pi_i : V \to \{1, \ldots, \lvert V\rvert\}$ jedem n-Gramm aus V eine Position aus $\{1, \ldots, \lvert V\rvert\}$ in der Permutation zuordnet.	Fuzzyfizierungsfunktion $\sigma : \mathbf{R} \to \{0, \ldots, r-1\}$, die die reellen Zahlen $\mathbf{R}$ auf $\{0, \ldots, r-1\}$ abbildet. Die Bildmenge steht für die r linguistischen Variablen des zugrundeliegenden Fuzzy-Schemas.
(1) Dimensionsreduktion	*Shingling.* Selektion von $\mathbf{d}' \subset \mathbf{d}$, so dass für alle n-Gramme $v \in \mathbf{d}'$ gilt: Es existiert ein π_i in ρ und $\pi_i(v) = \min_{x \in \mathbf{d}}(\pi_i(x))$. Jedes n-Gramm in $\mathbf{d}'$ minimiert also mindestens eine der Zufallspermutationen bezüglich der n-Gramme in $\mathbf{d}$. Auf diese Weise wird die Dimensionalität – die Größe des Vokabulars V – typischerweise um eine Größenordnung reduziert.	Normalisierung von $\mathbf{d}$ mit den a-Priori-Wahrscheinlichkeiten jedes Token-n-Gramms (vgl. Tabelle 2), ermittelt auf Grundlage einer großen Dokumentkollektion. Reformulierung von $\mathbf{d}$ in ein niedrigdimensionales $\mathbf{d}'$ (weniger als 100 Dimensionen). Jede Dimension von $\mathbf{d}'$ erhält als Gewicht die Summe der Gewichte einer Teilmenge der Dimensionen des hochdimensionalen, normalisierten $\mathbf{d}$. Die Teilmengen werden im Voraus zufällig oder mit Domänenwissen gebildet.
(2) Quantisierung	Die n-Gramme aus $\mathbf{d}'$ werden mit Standard-Hashing-Funktionen zu unterschiedlichen Hashcodes verrechnet, die die Menge $\mathbf{d}''$ bilden. Diese Menge kann bereits als Fingerprint verwendet werden. Sie ist in der Regel mit mehr als 10 Elementen aber noch zu umfangreich, um effizientes Fingerprinting zu ermöglichen.	Fuzzyfizierung der Vektorkomponenten von $\mathbf{d}'$ mit der Fuzzyfizierungsfunktion σ. Der reellwertige Vektor $\mathbf{d}'$ wird so zum ganzzahligen Vektor $\mathbf{d}''$. Die im Fuzzy-Schema definierten linguistischen Variablen sind z.B. „kleine Abweichung", „mittlere Abweichung" und „große Abweichung" vom Erwartungswert.
(3) Kodierung	*Supershingling.* Die Hashcodes $\mathbf{d}''$ werden sortiert, und anschließend der Reihe nach, getrennt durch Leerzeichen, zu einem String verknüpft. Dieser String wird abermals in n-Gramme unterteilt, und anschließend werden die Schritte 1 und 2, parametrisiert mit einer Zufallspermutation π noch einmal ausgeführt. Das Ergebnis des 2. Schritts ist der Hashcode $h_\varphi^{(\rho)}(\mathbf{d})$.	Kodierung des Hashcodes $h_\varphi^{(\rho)}(\mathbf{d})$ nach folgender Formel: $$h_\varphi^{(\rho)}(\mathbf{d}) = \sum_{i=0}^{\lvert\mathbf{d}''\rvert} \mathbf{d}''[i] \cdot r^i,$$ wobei $\mathbf{d}''[i]$ die i-te Vektorkomponente von $\mathbf{d}''$ ist.
Referenz	Broder [2000]	Stein [2005]

die Anzahl aneinandergrenzender Raumregionen exponentiell mit der Dimensionalität steigt.

Es gibt zwei Möglichkeiten zur Dimensionsreduktion: Projektion und Einbettung. Bei der Projektion werden eine Reihe von Dimensionen selektiert, die in der niedrigdimensionalen Repräsentation verbleiben, die übrigen Dimensionen werden verworfen. Bei der Einbettung werden die Dimensionen des niedrigdimensionalen Raumes aus denen des hochdimensionalen Raumes neu berechnet.

(2) *Quantisierung* $\mathbf{d}' \Rightarrow \mathbf{d}''$. Hier wird das niedrigdimensionale $\mathbf{d}'$ einer bestimmten Raumregion zugeordnet, die durch den Vektor $\mathbf{d}''$ beschrieben wird, indem zum Beispiel reellwertige Komponenten diskretisiert werden. Im Idealfall werden hinreichend ähnliche Vektoren derselben Raumregion zugeordnet.

(3) *Kodierung* $\mathbf{d}'' \to \mathbf{N}$. Wird eine durch $\mathbf{d}''$ beschriebene Raumregion beobachtet, kann sie mit einer Kodierungsvorschrift auf eine natürliche Zahl abgebildet werden. Diese Zahl dient als Hashcode $h_\varphi^{(\rho)}(\mathbf{d})$.

Fingerprinting wird in der Plagiaterkennung dazu verwendet, verdächtige Passagen in Bezug auf ein Dokument d_q aus einer Menge von Dokumenten D zu extrahieren, wobei der Fingerint-Index die Fingerprints aller Passagen s_x aus D indiziert.

Es haben sich für Textdokumente in den Experimenten von Potthast and Stein [2008] zwei Hashing-Verfahren als erfolgreich erwiesen: (Super-)Shingling und Fuzzy-Fingerprinting. In Tabelle 3 wird jedes Verfahren nach dem oben vorgestellten Schema erläutert. Beide können auf ein Code-Retrieval-Modell angewendet werden.

4 Evaluierung

In unseren Experimenten testen wir das in Abschitt 2 vorgeschlagene Code-Retrieval-Modell basierend auf Token-Strings und Longest-Common-Subsequence und vergleichen es mit einem anderen Code-Retrieval-Modell. Weiterhin vergleichen wir die Retrieval-Qualität der in Abschnitt 3 vorgeschlagenen Fingerprinting-Verfahren Supershingling und Fuzzy-Fingerprinting beim Retrieval von Code-Abschnitten aus einer großen Kollektion von Quellcodedokumenten. Die Retrieval-Qualität eines Verfahrens messen wir anhand seiner Precision und seines Recalls. Ersteres misst für ein Retrieval-Ergebnis den Anteil relevanter Dokumente an allen gefundenen Dokumenten, letzteres misst den Anteil der gefundenen relevanten Dokumente an allen relevanten Dokumenten.

Testkorpus. Grundlage für unseren Testkorpus ist der Quellcode des Open-Source-Projekts „Java New Operating

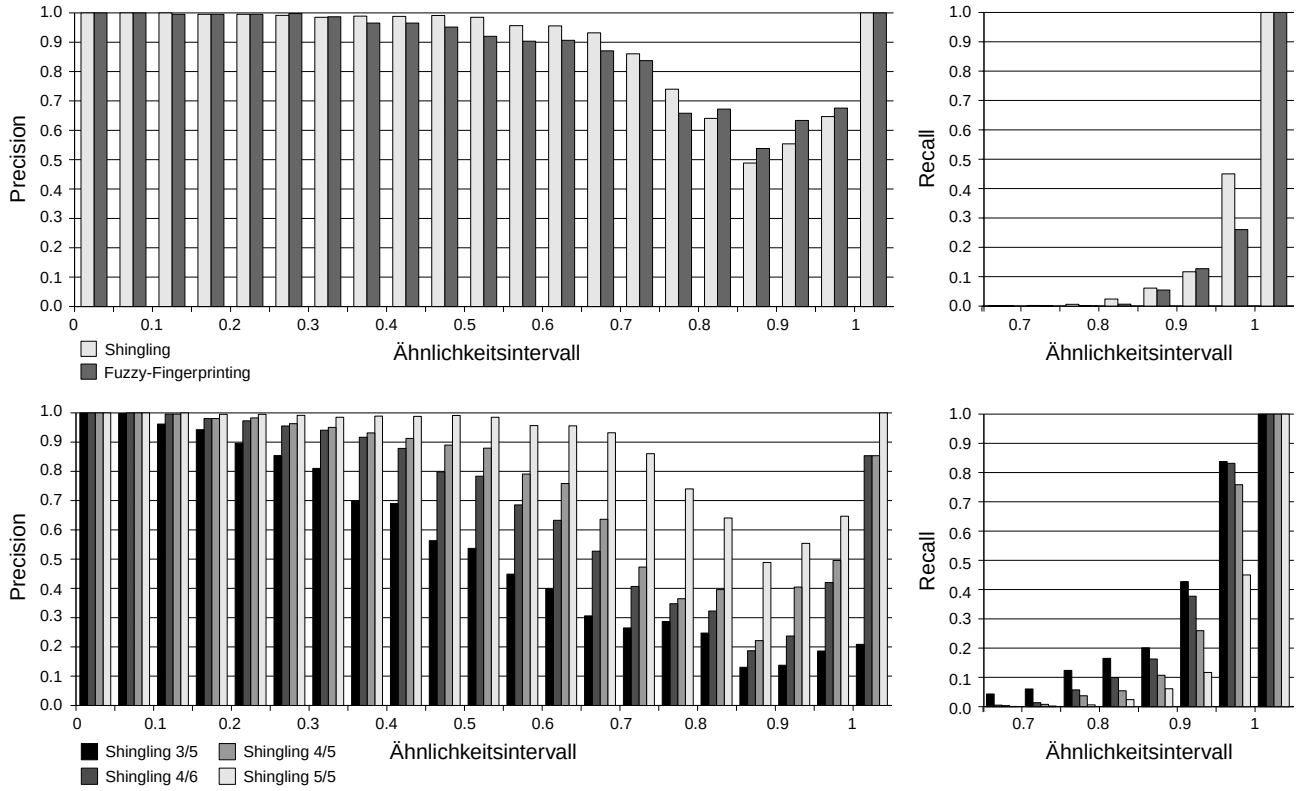

Abbildung 2: Oben: Precision und Recall von Shingling und Fuzzy-Fingerprinting abhängig von Ähnlichkeitsintervallen. Unten: Precision und Recall von Shingling-Varianten abhängig von Ähnlichkeitsintervallen.

System Design Effort (JNode)" [siehe 3]. Aus dem frei zugänglichen Quellcode haben wir 18 aufeinander folgende Release-Versionen des gesamten Projekts heruntergeladen. Insgesamt haben wir so 80 091 Quellcodedokumente gesammelt. Auf diese Weise simulieren wir Fälle von Code-Plagiaten, da aufeinander folgende Versionen derselben Klasse mit hoher Wahrscheinlichkeit Ähnlichkeiten aufweisen, der Code aber von Release zu Release umstrukturiert und weiterentwickelt wurde. Im Hinblick auf den Anwendungsfall der Plagiaterkennung betrachten wir Code-Abschnitte in der Länge von Methoden. Der Korpus enthält insgesamt 121 215 Methoden, wobei diejenigen, deren Token-String weniger als 12 Token lang ist, unberücksichtigt blieben, da es sich hier hauptsächlich um Get-/Set-Methoden oder ähnliches handelt.

[3] Web-Seite: http://www.jnode.org

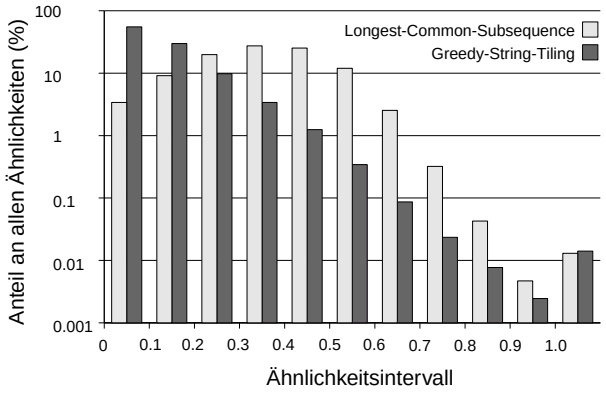

Abbildung 3: Ähnlichkeitsverteilung von 50 000 zufällig gezogenen (i.i.d.) Paaren von Methoden. Gegenübergestellt sind die Ähnlichkeitsmaße Longest-Common-Subsequence und Greedy-String-Tiling.

Vergleich von Code-Retrieval-Modellen. Im ersten Experiment haben wir die Eignung des von uns vorgeschlagenen Ähnlichkeitsmaß Longest-Common-Subsequence evaluiert. Wir haben es mit dem bislang am häufigsten eingesetzten Ähnlichkeitsmaß Greedy-String-Tiling verglichen. Dazu wurden zufällig 50 000 Paare von Methoden aus dem Testkorpus gewählt und die Ähnlichkeit mit jeweils beiden Modellen berechnet. Abbildung 3 zeigt die Verteilung der beobachteten Ähnlichkeiten abhängig von Ähnlichkeitsintervallen. Ähnlichkeiten des Maßes Longest-Common-Subsequence sind im Durchschnitt größer als die von Greedy-String-Tiling, die erwartete Ähnlichkeit zweier Methoden liegt bei etwa 0.3. Das entspricht unserer Auffassung von Code-Ähnlichkeit, da sich zwei Methoden, die hinreichend lang sind, niemals völlig voneinander unterscheiden (z.B. werden Schleifen und Konditionalblöcke oft verwendet). Daher erlangen auch bedeutend mehr Methodenpaare eine hohe Ähnlichkeit von mehr als 0.7. Einige Stichproben wurden manuell ausgewertet mit dem Ergebnis, dass die Ähnlichkeitsbemessung mit Longest-Common-Subsequence vorgezogen wurde. Dieser Eindruck ist jedoch nicht ohne Weiteres verallgemeinerbar; für einen größer angelegten Vergleich fehlt ein Korpus mit von Menschenhand bewerteten Ähnlichkeiten.

Vergleich von Shingling und Fuzzy-Fingerprinting. Die oberen Plots in Abbildung 2 zeigen Precision und Recall von Shingling und Fuzzy-Fingerprinting abhängig von Ähnlichkeitsintervallen. Für einen fairen Vergleich wurden beide Verfahren so parametrisiert, dass der Fingerprint für eine Methode genau einen Hashcode enthält (siehe Tabelle 4). Die Precision wurde auf Grundlage eines Fingerprint-

Tabelle 4: Parametrisierung der Fingerprinting-Verfahren für das Code-Retrieval, so dass ein Fingerprint genau einen Hashcode enthält.

Verfahren	**Parameter** ρ
Shingling	5 Zufallspermutationen π
Supershingling	Eine Zufallspermutation
Fuzzy-Fingerprinting	Fuzzyfizierungsfunktion σ mit $r = 3$ linguistischen Variablen: „negative Abweichnung", „keine Abweichung" und „positive Abweichung" vom Erwartungswert, wobei ersteres und letzteres eine Abweichung größer als 0.01 bedeutet.
	Einbettung in 20-dimensionalen Raum

Indexes für den gesamten Testkorpus ermittelt, an den 200 Anfragen gestellt wurden. Der Recall wurde mit Hilfe von mindestens 50 Paaren von Dokumenten pro Ähnlichkeitsintervall gemessen, deren Fingerprints berechnet und verglichen wurden.

Die Precision und der Recall beider Verfahren ist annähernd gleich. Einzig der Recall von Shingling ist bei Ähnlichkeiten zwischen 0.95 und 1 dem von Fuzzy-Fingerprinting überlegen. Wir nehmen an, dass Shingling hier einen Vorteil hat, da es einen größeren Anteil an Strukturinformationen des Quellcodes mit einbezieht als Fuzzy-Fingerprinting. Zum Vergleich: In den Experimenten von Potthast and Stein [2008] auf Textdokumenten liefern beide Verfahren gleich gute Ergebnisse.

Shingling-Varianten. Die unteren Plots in Abbildung 2 zeigen Precision und Recall von vier Varianten von Shingling: Die Variation besteht darin, dass kein Supershingling angewendet wird, sondern das Ergebnis des Quantisierungsschritts direkt als Fingerprint verwendet wird. Für kurze Code-Abschnitte ist das sinnvoll, da nicht genug n-Gramme in der hochdimensionalen Repräsentation vorhanden sind, um Shingling oder Supershingling mehrfach auszuführen. Der Fingerprint für eine Methode besteht hier aus 5 bzw. 6 Hashcodes, die mit Zufallspermutationen ausgewählt wurden. Zwei Methoden wurden als ähnlich angesehen, wenn mindestens c/5 der Hashcodes ihrer Fingerprints gleich waren, wobei $c \in \{3, 4, 5\}$. Shingling 5/5 entspricht in diesem Fall einem Fingerprint, der mit Supershingling berechnet wurde, so wie im vorherigen Experiment.

Es ist zu beobachten, dass die Shingling-Varianten eine deutliche Steigerung des Recalls bei in etwa halb so großer Precision ermöglichen, wobei Shingling 4/5 den besten Kompromiss bietet. Im zusammengefassten Ähnlichkeitsintervall $[0.9, 1)$ beträgt die gemittelte Precision 0.45 und der gemittelte Recall 0.51.

5 Zusammenfassung

In der vorliegenden Arbeit werden die Grundlagen zur Plagiaterkennung in Quellcode erarbeitet, die analog zu der in Texten abläuft. Frühere Arbeiten haben sich in dieser Hinsicht nur mit einem eingeschränkten Spezialfall befasst, nämlich der Plagiaterkennung in einer kleinen, geschlossenen Kollektion studentischer Programmierübungen. Der hier vorgestellte Ansatz betrachtet dagegen die Erkennung von Plagiaten in einer offenen Retrieval-Situation. Daraus ergeben sich eine Reihe von bislang nicht betrachteten Problemstellungen, wie zum Beispiel das heuristische Retrieval von Quellcode, die Analyse von Änderungen im Programmierstil zur Plagiaterkennung, oder das Retrieval der Lizenz zu einem Code-Abschnitt auf einer Web-Seite.

Der größte Unterschied zwischen der Plagiaterkennung in Text und der in Quellcode besteht im zur Ähnlichkeitsmessung eingesetzten Retrieval-Modell. In Texten genügt ein vergleichsweise einfaches Bag-of-Word-Modell, wohingegen Code-Retrieval-Modelle auf der Struktur der Token im Quellcode aufbauen und komplexe Ähnlichkeitsmaße dafür benötigen. Umgekehrt erleichtert die starke Strukturierung von Quellcode Aufgaben wie das Unterteilen von Dokumenten in Passagen (Chunking), das bei Textdokumenten eine große Herausforderung darstellt. Darüber hinaus gibt es kaum Unterschiede im generellen Vorgehen zur Plagiaterkennung.

Unsere Forschungsbeiträge sind ein neues Code-Retrieval-Modell, das insbesondere zur Messung der Ähnlichkeit kurzer Code-Abschnitte geeignet ist, und die erstmalige Verwendung von Fingerprinting für das Retrieval von plagiierten Code-Abschnitten aus einer großen Kollektion. Die experimentelle Auswertung zeigt hier eine vielversprechende Retrieval-Qualität. Dennoch kann die Problemstellung der Plagiaterkennung in Quellcode noch nicht als gelöst betrachtet werden.

Literatur

Ira Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In *Proc. International Conference on Software Maintenance*. IEEE Computer Society Press, 1998.

Andrei Z. Broder. Identifying and filtering near-duplicate documents. In *COM'00: Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, pages 1–10, London, UK, 2000. Springer-Verlag. ISBN 3-540-67633-3.

Steven Burrows, S. M. M. Tahaghoghi, and Justin Zobel. Efficient plagiarism detection for large code repositories. *Software–Practice and Experience*, 37: 151–175, September 2006.

Chen, Francia, Li, McKinnon, and Seker. Shared information and program plagiarism detection. *IEEETIT: IEEE Transactions on Information Theory*, 50, 2004. URL `http://monod.uwaterloo.ca/papers/04sid.pdf`.

Paul Clough, Robert Gaizauskas, Scott S.L. Piao, and Yorick Wilks. Measuring text reuse. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 152–159, 2002.

K. Fullam and J. Park. Improvements for scalable and accurate plagiarism detection in digital documents. `https://webspace.utexas.edu/fullamkk/pdf/DataMiningReport.pdf`, 2002.

Sam Grier. A tool that detects plagiarism in pascal programs. In *SIGCSE '81: Proceedings of the twelfth SIGCSE technical symposium on Computer science education*, pages 15–20, New York, NY, USA, 1981. ACM. ISBN 0-89791-036-2. doi: http://doi.acm.org/10.1145/800037.800954.

Timothy C. Hoad and Justin Zobel. Methods for Identifying Versioned and Plagiarised Documents. *American Society for Information Science and Technology*, 54(3):203–215, 2003.

Chao Liu, Chen Chen, Jiawei Han, and Philip S. Yu. GPLAG: detection of software plagiarism by program dependence graph analysis. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 872–881. ACM, 2006. ISBN 1-59593-339-5. URL `http://www.ews.uiuc.edu/~chaoliu/papers/kdd06liu.pdf`.

Sven Meyer zu Eissen and Benno Stein. Intrinsic plagiarism detection. In Mounia Lalmas, Andy MacFarlane, Stefan M. Rüger, Anastasios Tombros, Theodora Tsikrika, and Alexei Yavlinsky, editors, *Proceedings of the European Conference on Information Retrieval (ECIR 2006)*, volume 3936 of *Lecture Notes in Computer Science*, pages 565–569. Springer, 2006. ISBN 3-540-33347-9.

Gilad Mishne and Maarten de Rijke. Source code retrieval using conceptual similarity. In *Proceedings RIAO 2004*, pages 539–554, 2004. URL `http://staff.science.uva.nl/~gilad/pubs/riao2004.pdf`.

K. J. Ottenstein. An algorithmic approach to the detection and prevention of plagiarism. *SIGCSE Bull.*, 8(4): 30–41, 1976. ISSN 0097-8418. doi: http://doi.acm.org/10.1145/382222.382462.

Martin Potthast. Wikipedia in the pocket - indexing technology for near-duplicate detection and high similarity search. In Charles Clarke, Norbert Fuhr, Noriko Kando, Wessel Kraaij, and Arjen de Vries, editors, *30th Annual International ACM SIGIR Conference*, pages 909–909. ACM, July 2007. ISBN 987-1-59593-597-7.

Martin Potthast and Benno Stein. New Issues in Near-duplicate Detection. In Christine Preisach, Hans Burkhardt, Lars Schmidt-Thieme, and Reinhold Decker, editors, *Data Analysis, Machine Learning and Applications*, pages 601–609, Berlin Heidelberg New York, 2008. Springer. ISBN 978-3-540-78239-1.

Lutz Prechelt, Guido Malpohl, and Michael Philippsen. JPlag: Finding Plagiarisms among a Set of Programs. Technical Report 2000-1, University of Karlsruhe, Computer Science Department, 2000.

Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-634-X.

Antonio Si, Hong Va Leong, and Rynson W. H. Lau. Check: a document plagiarism detection system. In *SAC '97: Proceedings of the 1997 ACM symposium on Applied computing*, pages 70–77, New York, NY, USA, 1997. ACM Press. ISBN 0-89791-850-9. doi: http://doi.acm.org/10.1145/331697.335176.

Benno Stein. Fuzzy-Fingerprints for Text-Based Information Retrieval. In Klaus Tochtermann and Hermann Maurer, editors, *Proceedings of the 5th International Conference on Knowledge Management (I-KNOW 05), Graz*, Journal of Universal Computer Science, pages 572–579. Know-Center, July 2005.

Benno Stein. Principles of hash-based text retrieval. In Charles Clarke, Norbert Fuhr, Noriko Kando, Wessel Kraaij, and Arjen de Vries, editors, *30th Annual International ACM SIGIR Conference*, pages 527–534. ACM, July 2007. ISBN 987-1-59593-597-7.

Benno Stein and Martin Potthast. Construction of Compact Retrieval Models. In Sándor Dominich and Ferenc Kiss, editors, *Studies in Theory of Information Retrieval*, pages 85–93. Foundation for Information Society, October 2007. ISBN 978-963-06-3237-9.

Benno Stein, Sven Meyer zu Eissen, and Martin Potthast. Strategies for retrieving plagiarized documents. In Charles Clarke, Norbert Fuhr, Noriko Kando, Wessel Kraaij, and Arjen de Vries, editors, *30th Annual International ACM SIGIR Conference*, pages 825–826. ACM, July 2007. ISBN 987-1-59593-597-7.

Kristina L. Verco and Michael J. Wise. Software for detecting suspected plagiarism: comparing structure and attribute-counting systems. In *ACSE '96: Proceedings of the 1st Australasian conference on Computer science education*, pages 81–88, New York, NY, USA, 1996. ACM. ISBN 0-89791-845-2. doi: http://doi.acm.org/10.1145/369585.369598.