Candidate Document Retrieval for Web-Scale Text Reuse Detection*

Matthias Hagen and Benno Stein

Faculty of Media
Bauhaus-Universität Weimar, Germany
<first name>.<last name>@uni-weimar.de

Abstract Given a document d, the task of text reuse detection is to find those passages in d which in identical or paraphrased form also appear in other documents. To solve this problem at web-scale, keywords representing d's topics have to be combined to web queries. The retrieved web documents can then be delivered to a text reuse detection system for an in-depth analysis. We focus on the query formulation problem as the crucial first step in the detection process and present a new query formulation strategy that achieves convincing results: compared to a maximal termset query formulation strategy [10, 14], which is the most sensible non-heuristic baseline, we save on average 70% of the queries in realistic experiments. With respect to the candidate documents' quality, our heuristic retrieves documents that are, on average, more similar to the given document than the results of previously published query formulation strategies [4, 8].

1 Introduction

The problem considered in this paper appears as an important sub-task of automatic text reuse detection. A text reuse detection system aims at finding passages within a given document which, in a similar form, are also contained in another document. The goal is not only to identify simple one-to-one copies but also cases of paraphrased text reuse. Note that plagiarism detection represents a special case whereas text reuse detection addresses a broader spectrum that also covers problems like information spread analysis (e.g., where are news stories reused?).

Usually, automatic detection systems find potential reuse passages via face-to-face comparisons of the given document against a set of "promising" documents. While for small document collections it is feasible to perform a complete comparison against every document, this is obviously not possible when the collection is large. The idea then is to compare only against documents that cover a topic similar to the given document, with the rationale that such documents are more likely to be the source (or "sink") of text reuse. A straightforward approach to find documents on similar topics is to extract keywords or longer components like head noun phrases from the given document and to retrieve other documents also containing these keywords.

Our contribution to this problem is a strategy of how to query a web search engine using the extracted keywords. However, we do not deal with the complete task of

* Extended version of an ECDL 2010 poster paper [10].

text reuse detection. We tackle the essential pre-computation step that finds promising candidate documents for the in-depth analysis (for which we in turn assume that state-of-the-art text reuse or plagiarism detection techniques are used [4, 9, 13, 16]). We focus on *web querying* to identify potential candidates since the web became the typical place of text reuse. However, a detection system usually is not given arbitrary access to a web search engine's index; moreover it has incomplete or even no knowledge about the engine's underlying retrieval model, implementation details, and the like. A web search engine appears as a black box and queries are not for free but entail costs—at the very least some non-negligible amount of time is consumed, and monetary charges come into play for larger contingents of queries.[1]

1.1 User over Ranking

The number of documents a detection system can consider for an in-depth text reuse analysis is constrained by a processing capacity k, which in turn depends on the desired answer time, the processing time per document, and machine usage cost. If the entire set of extracted keywords (typically about 10) from a given document is submitted as a single web query, this query will probably be *overspecific* (i.e., hardly returning more than a handful of documents) and thus wasting processing capacity. On the other hand, queries containing only few of the extracted keywords are likely to be *underspecific* (i.e., having very long result lists) and discard valuable information: from overlong result lists only a fraction, typically the top-ranked results, can be processed by the detection system. Notice that such queries put the burden of selecting the most promising text reuse candidates on the search engine's ranking algorithm; potential text reuse cases that are not among the top results will be missed. Hence, a set of promising web queries should avoid underspecificity and, combined, cover all extracted keywords in order to ensure a high similarity to the given document's topic. Altogether, we argue that the probability to find potential text reuse cases by exploring k results becomes maximum if the combined result list length of the set of promising queries is in the order of magnitude of the processing capacity k of the detection system. This is an implication of the recent User-over-Ranking hypothesis, which states that queries have a higher probability of satisfying a user's information need if they return about as many results as the user can consider [17]. Figure 1 illustrates the outlined connections.

Under the User-over-Ranking hypothesis the treated query formulation sub-problem of text reuse detection is defined as follows:

CAPACITY CONSTRAINED QUERY FORMULATION

Given: (1) Set W of keywords.
(2) Query interface for web search engine $\mathcal{S}$.
(3) Upper bound k on the number of desired documents.

Task: Find a family $\mathcal{Q} \subseteq 2^W$ of queries together returning at most k documents, while containing all keywords from W.

[1] E.g., \$0.40–\$0.80 per 1000 Yahoo! BOSS queries, `http://www.ysearchblog.com/2011/02/08/latest-on-boss/` (accessed April 16th, 2011).

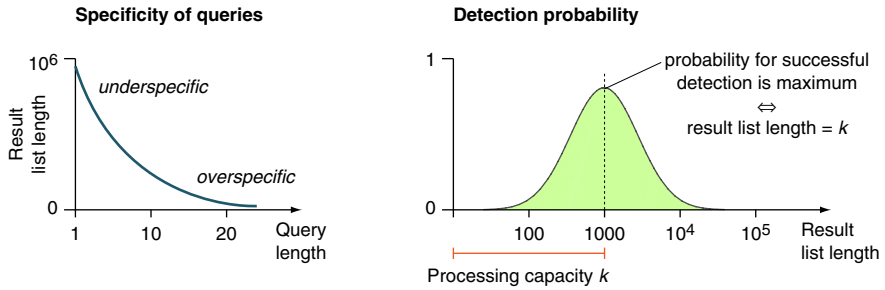

Figure 1. Left: a query with few terms or many results is likely to be underspecific, queries with many terms or few results tend to be overspecific. Right: under the User-over-Ranking hypothesis a combined result list length of the detection system's capacity k maximizes the probability of finding potential text reuse cases [17].

1.2 Related Work

One of the earliest approaches on formulating queries respecting a bound on the number of returned results is the maximal termset query formulation of Pôssas et al. [14]. We use an adapted version of maximal termset query formulation as our baseline. With respect to runtime, our new system clearly improves upon the maximal termset baseline while retrieving basically the same candidate documents.

A recent paper by Bendersky and Croft also deals with the scenario of text reuse detection on the web [4]. However, Bendersky and Croft's problem setting is different from ours: they focus on single sentences and not on complete documents as input, and hence their querying strategy is quite elementary. Our setting is also more general in another respect: the passages from the document for which a reuse analysis is requested have not to be known a priori. Nevertheless, in our experiments we compare our query formulation strategy to Bendersky and Croft's querying approach.

In our setting it would be desirable to use the given document as a query itself ("query by document"). Yang et al. [19] focus on such a scenario in the context of analyzing blog posts. But, similar to us, they also try to derive a keyword query that reflects the document's (blog post's) content. Their approach extracts keyphrases from the document, but formulates only a single query from them. Since this would waste capacity in our setting and since their approach of manually selecting the number of "good" keywords for each document is not applicable in a fully automatic system, we do to not include Yang et al.'s approach in the experimental comparison.

A more applicable setting which is also related to ours is Dasdan et al.'s work on finding similar documents by using only a search engine interface [8]. Although Dasdan et al. focus on a search engine coverage problem (resolve whether a search engine's index contains a given document or some variant of it), their approach of finding similar documents using keyword interfaces is basically equivalent to our setting. Dasdan et al. propose two querying strategies and experimentally show that their approaches indeed find similar documents. In our experiments we also compare their strategies to our heuristic.

A very promising idea for our setting would be to predict a given query's performance before submitting it to a search engine [6, 7, 11, 12]. However, the evaluation

of the best performing predictors needs access to knowledge that is not available at user site in a standard web search scenario. For example, the simplified query clarity predictor [12] needs the total keyword frequencies for the whole corpus—web search engines just return an estimation of the number of documents in the corpus that contain the keyword. The query scope predictor [12] needs the number of documents in the index—most web search providers stopped publishing it. Furthermore, there are studies suggesting to take care when interpreting the published evaluations of established predictors [15] such that we decided not to use quality prediction in our approaches.

2 Notation and Basic Definitions

Starting point of the query formulation process is a set $W = \{w_1, \ldots, w_n\}$ of keywords; allowing the w_i to be longer components like head noun phrases makes no difference. Subsets $Q \subseteq W$ can be submitted as web queries, with the notion that phrases are included in quotation marks. An engine's reply to a query consists of the beginning of an exhaustive, ranked list L_Q of snippets and URLs of documents relevant for Q, and an estimation ℓ_Q for the result list length $|L_Q|$. The task of CAPACITY CONSTRAINED QUERY FORMULATION is to find a family $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ of queries $Q_i \subseteq W$ having the following properties:

1. $\mathcal{Q}$ is *simple* in the sense that $Q_i \not\subseteq Q_j$ for any $Q_i, Q_j \in \mathcal{Q}$, with $i \neq j$. This avoids redundancy in the queries and the results.
2. Combined, $\mathcal{Q}$'s queries don't return more than k results. This respects the detection system's processing capacity.
3. Combined, $\mathcal{Q}$'s queries cover W's keywords if possible: ideally $\bigcup_{Q \in \mathcal{Q}} Q = W$. This ensures that the resulting documents cover all the topics contained in W.

With respect to the capacity k, we introduce a per-query upper bound $\ell_{\max}$ with the notion that a query Q is promising iff $\ell_Q \leq \ell_{\max}$ (i.e., it returns at most $\ell_{\max}$ results). How exactly this upper bound is derived will be explained in Section 3. A per-query lower bound $\ell_{\min}$ serves convenience purposes of ruling out queries with very few results (e.g., $\ell_{\min} = 1$ means that queries returning an empty result list are not tolerated). Applying both bounds, a query $Q \in \mathcal{Q}$ has to satisfy $\ell_{\min} \leq \ell_Q \leq \ell_{\max}$. Adopting notation from Bar-Yossef and Gurevich [2], we say that for $\ell_Q < \ell_{\min}$ the query Q is *underflowing*, whereas for $\ell_Q > \ell_{\max}$ it is *overflowing*. Queries that are neither under- nor overflowing are *valid*. A valid query Q is *minimal* iff dropping some keyword from Q results in an overflowing query.

As a solution to CAPACITY CONSTRAINED QUERY FORMULATION we suggest the family $\mathcal{Q}_{\text{lo}}$ of all minimal valid queries for an appropriate value of $\ell_{\max}$. Obviously, $\mathcal{Q}_{\text{lo}}$ is simple and depends on the value of $\ell_{\max}$. Our approach will adaptively determine values for $\ell_{\max}$, derive the corresponding $\mathcal{Q}_{\text{lo}}$ and, according to the combined number of $\mathcal{Q}_{\text{lo}}$'s results (more or less than k), output this $\mathcal{Q}_{\text{lo}}$ or re-iterate by setting $\ell_{\max}$ to a more appropriate value. Hence, an appropriate $\mathcal{Q}_{\text{lo}}$ respects the first two constraints of being simple and not returning more than k results.

That $\mathcal{Q}_{\text{lo}}$ also is a good choice with respect to the third constraint of covering as many keywords of W as possible can be seen as follows. We say that a query Q *covers*

all its keywords. Analogously, a family $\mathcal{Q}$ of queries covers all keywords in $\bigcup_{Q \in \mathcal{Q}} Q$. Note that there are situations where it is not possible to cover W with a family of valid queries (e.g., when a single keyword itself is underflowing). A keyword $w \in W$ is *coverable* iff there is a valid query $Q \subseteq W$ with $w \in Q$.

Lemma 1. *Let $\mathcal{Q}$ be a family of valid queries covering the coverable keywords from a keyword set W for given $\ell_{\min}$ and $\ell_{\max}$. For every $Q \in \mathcal{Q}$ we have: there is a sub-family $\mathcal{Q}'_{\text{lo}} \subseteq \mathcal{Q}_{\text{lo}}$ such that $Q = \bigcup_{Q' \in \mathcal{Q}'_{\text{lo}}} Q'$.*

Proof. Assume we have $Q \in \mathcal{Q}$ but $Q \neq \bigcup_{Q' \in \mathcal{Q}'_{\text{lo}}} Q'$ for any $\mathcal{Q}'_{\text{lo}} \subseteq \mathcal{Q}_{\text{lo}}$. Since $\mathcal{Q}$ is a family of valid queries, Q must be valid. Assume that Q contains only coverable keywords from W. Now consider the family $\mathcal{Q}'$ of the $2^{|Q|} - 1$ subqueries of Q excluding the empty query. Let $\mathcal{Q}'' \subseteq \mathcal{Q}'$ be the sub-family of valid queries. Note that $\mathcal{Q}''$ is not empty since it contains Q. From $\mathcal{Q}''$ we remove all queries that are proper supersets of queries in $\mathcal{Q}''$ and obtain the family $\tilde{\mathcal{Q}}$ of minimal valid subqueries of Q. Note that $\tilde{\mathcal{Q}}$ is not empty since $\mathcal{Q}''$ is not empty and that $Q = \bigcup_{\tilde{Q} \in \tilde{\mathcal{Q}}} \tilde{Q}$. Since $\tilde{\mathcal{Q}}$ contains minimal valid queries only, we have $\tilde{\mathcal{Q}} \subseteq \mathcal{Q}_{\text{lo}}$; a contradiction to our assumption. Hence, Q contains a non-coverable keyword $w \in W$. Since w is not coverable by a valid query, Q cannot be valid. A contradiction again. □

Corollary 1. *For a given set W of keywords and given $\ell_{\min}$ and $\ell_{\max}$, the respective family $\mathcal{Q}_{\text{lo}}$ covers the coverable keywords. Furthermore, $\mathcal{Q}_{\text{lo}}$ contains the with respect to set inclusion minimal queries covering the coverable keywords.*

In the process of finding an appropriate $\mathcal{Q}_{\text{lo}}$ on input W, we count the overall number $cost$ of queries that are submitted to the search engine. The underlying assumption is that a system is faster when it submits less queries.

3 Baseline: Maximal Termset Query Formulation

As a baseline query formulation process, we adapt the maximal termset approach by Pôssas et al. [14]. We refrain from using GENMAX as a subroutine to enlarge promising keyword subsets (as proposed in [14]) but choose the classic Apriori algorithm instead, which also comes from the field of frequent itemset mining [1]. Apriori is considered as one of the top 10 data mining algorithms [18]; it traverses the search space of all possible queries in a level-wise manner. A basic pseudo-code listing of Apriori for fixed lower and upper bounds $\ell_{\min}$ and $\ell_{\max}$ is given as Algorithm 1. How the algorithm handles the adaptive adjustment to detect a reasonable $\ell_{\max}$ is explained below, after introducing the basic Apriori framework.

Apriori first checks which of the initial keywords itself are overflowing or valid. The overflowing keywords form the first level of candidate queries (variable C_1 in line 2) that can be further expanded. A second pre-check ensures that these remaining keywords from the first candidate level altogether are not overflowing (line 3). Otherwise no valid queries can be formulated from them. After the pre-checks, Apriori combines candidate queries (lines 5 to 11) in a level-wise manner. It is straightforward to show that Algorithm 1 finally outputs the desired $\mathcal{Q}_{\text{lo}}$ for given $\ell_{\min}$ and $\ell_{\max}$; just notice

Algorithm 1 The Apriori algorithm for query formulation

Input: a set W of keywords, $\ell_{\min}$, and $\ell_{\max}$
Output: the family $\mathcal{Q}_{\text{lo}}$

1: $\mathcal{Q} \leftarrow \{\{w\} : w \in W \text{ and } \{w\} \text{ is valid}\}$
2: $C_1 \leftarrow \{\{w\} : w \in W \text{ and } \{w\} \text{ overflows}\}$
3: **if** $\bigcup_{\{w\} \in C_1} \{w\}$ overflows **then stop and output** $\mathcal{Q}$
4: $i \leftarrow 1$
5: **while** $C_i \neq \emptyset$ **do**
6: **for all** $Q, Q' \in C_i, |Q \cap Q'| = i - 1$ **do**
7: $Q_{\text{cand}} \leftarrow Q \cup Q'$
8: **if** $Q_{\text{cand}} \setminus \{w\} \in C_i$ for all $w \in Q_{\text{cand}}$ **then**
9: **if** Q_{cand} overflows **then** $C_{i+1} \leftarrow C_{i+1} \cup \{Q_{\text{cand}}\}$
10: **if** Q_{cand} is valid **then** $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{Q_{\text{cand}}\}$
11: $i \leftarrow i + 1$
12: **output** $\mathcal{Q}$

that whenever a query becomes valid it is directly added to the output (thus being minimal) and that, due to the exhaustive search character, no minimal valid query will be missed. A query's validity (lines 1, 2, 3, 9, and 10) is checked via submission to the web search engine. We use the engine's estimations ℓ_Q, although they often overestimate the correct result list lengths. However, they usually respect monotony (queries containing additional keywords have smaller ℓ-value) and the shorter the result list, the more accurate the estimations.

We adopt Algorithm 1 as our baseline—and even tighten this baseline by applying the following Lemma as a means to reduce the number of web queries Apriori submits.

Lemma 2. *Let $Q_1, Q_2, Q_3 \subseteq W$ be queries. Assuming the ℓ-estimations to be reasonable we have* $\ell_{Q_1 \cup Q_2 \cup Q_3} \geq \ell_{Q_1 \cup Q_2} + \ell_{Q_1 \cup Q_3} - \ell_{Q_1}$.

Proof.
$$\begin{aligned} \ell_{Q_1 \cup Q_2 \cup Q_3} &= |L_{Q_1 \cup Q_2} \cap L_{Q_1 \cup Q_3}| = |L_{Q_1 \cup Q_2} \setminus (L_{Q_1 \cup Q_2} \setminus L_{Q_1 \cup Q_3})| \\ &\geq |L_{Q_1 \cup Q_2} \setminus (L_{Q_1} \setminus L_{Q_1 \cup Q_3})| \geq |L_{Q_1 \cup Q_2}| - |L_{Q_1} \setminus L_{Q_1 \cup Q_3}| \\ &= |L_{Q_1 \cup Q_2}| - (|L_{Q_1}| - |L_{Q_1 \cup Q_3}|) = \ell_{Q_1 \cup Q_2} + \ell_{Q_1 \cup Q_3} - \ell_{Q_1} \quad \square \end{aligned}$$

Let Q and Q' be the queries that are merged to get Q_{cand} (line 7 of Apriori). Lemma 2 then is applied as follows: $Q_1 \cup Q_2 = Q$, $Q_1 \cup Q_3 = Q'$, and $Q_1 = Q \cap Q'$. The rationale is: if the Lemma 2 estimation $\ell_{Q_{\text{cand}}} \geq \ell_Q + \ell_{Q'} - \ell_{Q \cap Q'}$ is larger than $\ell_{\max}$, we do not have to submit Q_{cand} to an engine but can add it to the current candidate set C_{i+1} immediately.

A remaining problem is to adaptively set $\ell_{\max}$. The algorithm starts with $\ell_{\max} = k$, computes $\mathcal{Q}_{\text{lo}}$ using Apriori and checks the number of results returned by $\mathcal{Q}_{\text{lo}}$. Usually this will be too many since $\mathcal{Q}_{\text{lo}}$ contains more than one query. The algorithm then sets $\ell_{\max} = \lfloor \ell_{\max}/2 \rfloor$ and computes the corresponding $\mathcal{Q}_{\text{lo}}$. A naïve approach would restart the entire Apriori computation from scratch and repeat all steps from the previous run, resulting in a bad overall practical performance. However, a nice feature of Apriori is that it can be easily modified to continue computation on the reached state of the previous $\ell_{\max}$ setting such that re-computations and re-submissions of web queries are

avoided. If at one intermediate step the algorithm notices that the current $\mathcal{Q}_{\text{lo}}$ returns not more but approximately k results (we set the bound to at least 90%), it stops and outputs the current $\mathcal{Q}_{\text{lo}}$. If eventually too few results are returned, the algorithm sets $\ell_{\max} = \lfloor 3/2 \cdot \ell_{\max} \rfloor$. Altogether, this implements a kind of binary search for a good value of $\ell_{\max}$. Note that whenever the algorithm enlarges $\ell_{\max}$ for the first time, all of the currently needed queries have already been examined during the previous step such that no further queries have to be submitted.

4 Heuristic Search Strategy

Preliminary tests revealed that the savings due to Lemma 2 in the Apriori algorithm are often negligible: the sum $\ell_Q + \ell_{Q'}$ usually is too small compared to $\ell_{Q \cap Q'}$ and hence the query $Q_{\text{cand}} = Q \cup Q'$ has to be submitted. Nevertheless, the performance of the baseline Apriori framework can be significantly improved. We propose a heuristic that mimics Apriori's workflow in the second of the following two phases. The first phase of our heuristic can be seen as a pre-processing step although it submits exactly the same queries as Apriori does on the first two levels (while $i < 2$). However, for a keyword set W co-occurrence information obtained from the estimations of the first and second Apriori levels are stored in a matrix M in form of the—here called—*yield factors* $\gamma(w, w') = \ell_{\{w,w'\}}/\ell_{\{w\}}$. A yield factor $\gamma(w, w')$ multiplied by $\ell_{\{w\}}$ gives the yield of web results when the keyword w' is added to the query $\{w\}$. Note that the yield factors are not symmetric (i.e., usually $\gamma(w', w) \neq \gamma(w, w')$) such that M also is not symmetric.

The second phase of our heuristic then starts an Apriori-like candidate generation on the third level (queries containing three keywords). Hence, our technique does not save queries on the first two levels compared to our baseline Algorithm 1 but from Level 3 on the heuristic uses the yield factors to internally estimate a query and only submit it as a web query if necessary. Assume we are on some level $i \geq 3$ and that all processed queries Q from lower levels have a stored value est_Q, indicating an estimation of the length of their result lists, and a value age_Q, indicating the elapsed Apriori levels from the last time a subset of Q was submitted as a web query. Hence, for $age_Q = 0$ we have $est_Q = \ell_Q$. Let the current candidate query Q_{cand} be obtained by merging Q and Q' (line 7 of Apriori). Before submitting a web query, we now internally compute $est_{Q_{\text{cand}}}$ as follows. Let $age_Q \leq age_{Q'}$ and $Q' \setminus Q = \{w'\}$. We set $est_{Q_{\text{cand}}} = est_Q \cdot \text{avg}\{\gamma(w, w') : w \in Q\}$, where avg denotes the mean value. Submitting Q_{cand} as a web query and storing the engine's $\ell_{Q_{\text{cand}}}$ as $est_{Q_{\text{cand}}}$ is done iff $est_{Q_{\text{cand}}} < adj \cdot \ell_{\max}$ for a given adjustment factor adj. If however $est_{Q_{\text{cand}}} \geq adj \cdot \ell_{\max}$ we do not submit a web query but store the internally derived $est_{Q_{\text{cand}}}$ and set $age_{Q_{\text{cand}}} = age_Q + 1$.

The rationale for using the factor adj in the above inequalities is as follows. An experimental in-depth analysis revealed that $\ell_Q \geq est_Q$ holds for most queries Q, though there are rare cases where Q is valid or underflowing while $est_Q > \ell_{\max}$ (i.e., even the tendency of the internal estimation is wrong). For this reason, the informed heuristic does not blindly follow the internal estimations but only trusts them when $est_{Q_{\text{cand}}} \geq adj \cdot \ell_{\max}$ for an adjustment factor adj. The rationale is that as long as the internal estimations are sufficiently above the validity bound $\ell_{\max}$, the probability for a wrong validity check based on the internal estimation is negligible. Only when the

internal estimation $est_{Q_{\text{cand}}}$ is close to or below the validity bound $\ell_{\max}$, the current query is submitted to the search engine in order to "adjust" the internal estimation with the search engine's $\ell_{Q_{\text{cand}}}$. Larger values of adj enlarge the adjustment range and thus guarantee to catch more of the rare cases where Q is valid but $est_Q > \ell_{\max}$. However, this comes with a larger amount of submitted web queries. Moreover, only huge values of adj can guarantee the heuristic to return the same family $\mathcal{Q}_{\text{lo}}$ as the baseline. We compare different realistic settings of adj, and the fine-tuning shows good conformity of the output with the baseline's $\mathcal{Q}_{\text{lo}}$ while saving lots of queries (cf. Section 5).

5 Experimental Analysis

In a first experiment, we compare the uninformed Apriori baseline to our yield factor informed heuristic with respect to the number of submitted queries. In a second experiment, we then compare our heuristic to Bendersky and Croft's and Dasdan et al.'s query formulation strategies [4, 8] with respect to the quality of the retrieved documents. The experimental setting for both experiments is inspired by the observation that scientific publications often follow an evolutionary process from a technical report / workshop / poster / or short paper level to a full conference paper and sometimes to a journal paper. Although the different versions of the same publication have a potentially different and more complete presentation at more mature levels, they still deal with the same topic—such that we assume a significant amount of text reuse among them. To model the described scenario, we crawled computer science papers from major conferences and journals available on the web and tried to find a previous version for each. The document pairs were manually checked to ensure that they really are different versions of the same paper; we obtain 257 such verified pairs, all written in English. We verified that both versions are retrievable using the Bing API that we use in our experiments.

5.1 Number of Submitted Queries

For each of the 257 document pairs we extract a number of keywords from the more mature paper (e.g., conference vs. workshop) and then formulate queries using these keywords. For the keyword extraction we use an implementation of the head noun extractor [3]. We set $\ell_{\min} = 1$ to foreclose queries with no results. Furthermore, we set $k = 1000$ since current state-of-the-art automatic plagiarism detection techniques against a collection of 1000 potential source documents run in about 10 minutes [9], which we consider as a reasonable answer time for most text reuse detection scenarios. For each keyword set extracted from a document of our test collection, we run the Apriori baseline and our heuristic with the first 4, 5, ..., 10 extracted keywords against the Bing API during November 07–20, 2010.

Table 1 contains the results of this experiment. Different settings of the heuristic's adjustment factor correspond to different rows. Note that especially for small numbers of extracted keywords even the complete query containing all keywords is often overflowing. Because $\mathcal{Q}_{\text{lo}}$ cannot be computed in such cases and the first 1000 results of the complete query should be used instead, we filter out the corresponding documents and derive the statistics just for the remaining ones. For those inputs where the computation of $\mathcal{Q}_{\text{lo}}$ is possible, all four approaches always find a $\mathcal{Q}_{\text{lo}}$. We report the average

Table 1. Results of the number-of-queries experiment.

	Number of extracted keywords							
	3	4	5	6	7	8	9	10
Number of documents where								
Complete query overflows	238	207	177	146	124	102	93	81
$\mathcal{Q}_{\text{lo}}$ computation possible	19	50	80	111	133	155	164	176
Average cost (number of submitted queries)								
Heuristic, $adj = 1$	4.91	6.69	9.35	13.30	20.20	32.58	53.13	95.86
Heuristic, $adj = 3$	5.81	7.88	10.85	16.48	26.16	43.44	70.77	125.56
Heuristic, $adj = 5$	5.91	8.73	13.55	20.41	33.30	53.13	87.70	159.16
Uninformed baseline	6.09	10.65	19.08	34.60	61.66	106.19	178.98	302.87
Average cost ratio (basis: uninformed baseline)								
Heuristic, $adj = 1$	**0.80**	**0.63**	**0.49**	**0.38**	**0.33**	**0.31**	**0.30**	**0.32**
Heuristic, $adj = 3$	0.95	0.74	0.57	0.48	0.42	0.41	0.40	0.41
Heuristic, $adj = 5$	0.97	0.82	0.71	0.59	0.54	0.50	0.49	0.53
Average $\lvert\mathcal{Q}_{\text{lo}}\rvert$								
Heuristic, $adj = 1$	1.33	1.85	2.69	3.76	5.25	7.44	10.65	14.72
Heuristic, $adj = 3$	1.32	1.84	2.69	3.75	5.28	7.50	10.88	14.85
Heuristic, $adj = 5$	1.33	1.85	2.72	3.83	5.38	7.61	11.01	14.97
Uninformed baseline	1.33	1.87	2.75	3.88	5.49	7.78	11.12	15.18
Average number of retrievable result URLs (without duplicates)								
Heuristic, $adj = 1$	71	104	157	221	315	428	555	679
Heuristic, $adj = 3$	68	101	158	223	314	422	553	682
Heuristic, $adj = 5$	70	99	161	228	325	439	562	686
Uninformed baseline	69	98	162	231	328	445	569	690
Average ratio of common result URLs with baseline								
Heuristic, $adj = 1$	0.95	0.92	0.92	0.93	0.93	0.94	0.94	0.95
Heuristic, $adj = 3$	0.92	0.97	0.98	0.98	0.97	0.98	0.95	0.97
Heuristic, $adj = 5$	0.98	0.98	0.99	0.99	0.98	0.99	0.98	0.98

number $cost$ of web queries the approaches submitted to obtain the output $\mathcal{Q}_{\text{lo}}$ and the average ratio of submitted queries compared to the baseline (smaller $cost$ and smaller ratio indicate better approaches). Note that using very few keywords results in fewer retrievable documents using the $\mathcal{Q}_{\text{lo}}$ queries. We also observed that we needed at least 6 keywords to guarantee the retrieval of the original document and its previous version among $\mathcal{Q}_{\text{lo}}$'s web results. Hence, we suggest to use about 10 extracted keywords to obtain a meaningful set of documents from our approaches.

With respect to the runtime, the possible savings in the number of submitted queries are substantial compared to the baseline. For 7 or more keywords our heuristics save 70% of the queries. For all approaches the internal computation time to formulate the queries is never larger than several hundred milliseconds, while a typical web query against the API takes about 300ms–550ms. Hence, the fastest algorithm always is the one that submits the fewest queries. With respect to the quality of the heuristics' $\mathcal{Q}_{\text{lo}}$, we compare the average size of the generated $\mathcal{Q}_{\text{lo}}$ and the ratio of retrieved result URLs common with the baseline's results. The small differences are due to some rare overestimations using the internal expectations that hide some of the queries the baseline finds.

It can be observed that larger values of the adjustment factor adj are able to compensate for more of these overestimations. Additional spot checks show that the difference of the baseline's results to the heuristic with $adj = 1$ is rather small, such that for larger keyword sets the by far better running time should be favored. For keyword sets of size 10 the fastest heuristic with $adj = 1$ computes $\mathcal{Q}_{\text{lo}}$ in about 38 seconds compared to 50 seconds for $adj = 3$, 64 seconds for $adj = 5$, and about 2 minutes for the baseline. This is a saving of 70%. Hence, a near real time text reuse detection service can safely extract 10 keywords.

5.2 Candidate Document Quality

The first experiment shows the heuristic with $adj = 1$ to outperform the other approaches with respect to runtime (while retrieving basically the same set of documents). On the same corpus, we now compare this variant to other previously published query formulation strategies with respect to the quality of the retrieved documents. Important competitors in this regard are Bendersky and Croft's and Dasdan et al.'s query formulation approaches [4, 8].

Bendersky and Croft submit $2n - 1$ queries for a set of n keywords [4]; the first n queries are submitted to obtain the search engine estimations ℓ_w for each keyword w. The keywords are ordered by descending ℓ-value and submitted as a single query containing all n keywords. The approach then iteratively drops the last keyword and submits the resulting queries until 1000 documents are retrieved.

Dasdan et al. describe two approaches [8] that are slightly different right from the keyword extraction. Their first approach (LFT) extracts from a document the terms that are least frequent on the web, using a dictionary with web frequencies like 1-grams from the Google 5-gram corpus [5]. The strategy submits 10 queries: the first one contains the 10 least frequent keywords, the second one contains the next 10 least frequent keywords, and so forth. Dasdan et al.'s second approach (RST) builds 10 queries each of which containing a random sequence of 10 words from the given document. Note that this approach aims to their original problem setting of search engine coverage analysis, where the task is to find near-duplicates of a document in a search engine index. However, such strategies of using a random string from a document are also often suggested as an "intelligent" strategy to manually detect plagiarism. Hence, we decided to employ RST in our candidate document quality experiment. We adjust LFT and RST to only retrieve the top 100 results of each of the constructed queries to ensure for $k = 1000$.

Bendersky and Croft's approach uses the same 10 extracted keywords as our heuristic. Note that for the 81 documents from our collection, where the 10 keywords as one query already overflow, our heuristic and Bendersky and Croft's approach retrieve the same result documents, since both use the first 1000 documents from the all-keywords query. To detect the keywords for LFT, we indexed the Google 1-grams in a big hash table and for our 257 documents in a pre-processing detected the 100 keywords with lowest frequencies. As for the RST approach, a pre-processing sampled 10 random sequences of 10 consecutive words from the documents. Our heuristic obviously submits more queries (65.96 on average: just one query for the 81 documents where $\mathcal{Q}_{\text{lo}}$ is not possible and an average of 95.86 for the other 176 documents) than Bendersky and Croft's approach (10.45 queries on average; note that 19 is the worst case but often

Table 2. Average cosine similarity (tf weights) of the retrieved documents to the given document.

Measure	Approach			
	Our heuristic	[4]	LFT	RST
10 most similar documents	0.55	0.55	**0.56**	**0.56**
100 most similar documents	**0.39**	0.37	0.35	0.29
all retrieved documents	**0.29**	0.25	0.22	0.21

1000 results are retrieved earlier, for 81 documents even with the first query) or LFT and RST (10 queries). With respect to the quality of the retrieved documents, we first check whether the different approaches have the two paper versions among their results. For our heuristic, for Bendersky and Croft's approach as well as for LFT this is always the case. However, RST missed the previous version 8 times. This is probably due to the fact that random sequences help to find nearly-identical versions of a document but that scientists also often rewrite a paper in different versions. Since RST was primarily developed to find near-duplicates, the few misses are no surprise.

The aim of our approach is not only to retrieve the documents from our corpus but also to retrieve similar documents as good candidates: the assumption for text reuse detection is that more similar documents are more likely to contain the assumed text reuse. Hence, we downloaded the results the different approaches retrieve and compare the approaches with respect to similarity of the retrieved documents. As similarity measure we use cosine similarity with tf weights. Table 2 contains the results of this experiment. With respect to the retrieved 10 most similar documents, all approaches are somehow on par. However, this behavior changes significantly when one checks the average similarity for the 100 most similar or even all retrieved documents: then our approach outperforms the other approaches. The gap to Bendersky and Croft's approach is only due to the given documents for which $\mathcal{Q}_{\text{lo}}$ could be computed, since on the other documents our heuristic and Bendersky and Croft's approach return exactly the same documents (the top 1000 results of the query containing all keywords). The gap to LFT and RST is probably due to the slightly different use case as LFT and RST were mainly designed to retrieve a few near-duplicate instances of a given document. That goal is achieved as shown by Dasdan et al.'s experiments [8] and the slightly worse performance in our experiments is probably mainly due to the different scenario that our experiments address.

6 Conclusion and Outlook

We developed a new strategy to formulate promising queries from a given set of keywords. In our scenario a text reuse detection system "plays" against a retrieval system (the web search engine) in order to find promising queries that help to detect text reuse in or from a given document. Our formalization forms the ground for both to define the problem CAPACITY CONSTRAINED QUERY FORMULATION and to develop a heuristic search strategy that tackles a query-cost-oriented optimization variant. The analysis of our heuristic shows (1) that it drastically outperforms a maximal termset query formulation baseline system, and (2) that it finds candidate documents which are more similar to the original document than other approaches for related problems. If, however, a method is needed that returns few very similar web results for a given document

(like it is the case in Dasdan et al.'s scenario), using our heuristic would probably be an overhead since it requires more queries. But, if the aim is to retrieve a larger collection of documents all of which are similar to a given document (like it is the case in the text reuse detection scenario), the rather small 20 second overhead of our method can be regarded as a worthwhile investment for a better average similarity. A straightforward extension of the above similarity experiment is an analysis of the retrieved documents with state-of-the-art text reuse or plagiarism detection techniques [4, 9, 13, 16], and to compare the candidate document sets with respect to the number of found text reuse cases (and not just the similarity). However, this is beyond the scope of this paper: having shown the potential of our heuristic for the retrieval step, we leave the examination of text reuse cases as an interesting task for future work.

Bibliography

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. *Proc. of VLDB'94*, pages 487–499.
[2] Z. Bar-Yossef and M. Gurevich. Random sampling from a search engine's index. *JACM*, 55(5), 2008.
[3] K. Barker and N. Cornacchia. Using noun phrase heads to extract document keyphrases. *Proc. AI 2000*, pages 40–52.
[4] M. Bendersky and W. B. Croft. Finding text reuse on the web. *Proc. of WSDM 2009*, pages 262–271.
[5] T. Brants and A. Franz. Web 1T 5-gram Version 1. LDC2006T13, 2006.
[6] D. Carmel, E. Yom-Tov, A. Darlow, and D. Pelleg. What makes a query difficult? *Proc. of SIGIR 2006*, pages 390–397.
[7] S. Cronen-Townsend, Y. Zhou, and W. B. Croft. Predicting query performance. *Proc. of SIGIR 2002*, pages 299–306.
[8] A. Dasdan, P. D'Alberto, S. Kolay, and C. Drome. Automatic retrieval of similar content using search engine query interface. *Proc. of CIKM 2009*, pages 701–710.
[9] C. Grozea, C. Gehl, and M. Popescu. ENCOPLOT: Pairwise Sequence Matching in Linear Time Applied to Plagiarism Detection. *Proc. of PAN 09*, pages 10–18.
[10] M. Hagen and B. Stein. Capacity-constrained Query Formulation. *Proc. of ECDL 2010 (posters)*, pages 384–388.
[11] C. Hauff, D. Hiemstra, and F. de Jong. A survey of pre-retrieval query performance predictors. *Proc. of CIKM 2008*, pages 1419–1420.
[12] B. He and I. Ounis. Inferring query performance using pre-retrieval predictors. *Proc. of SPIRE 2004*, pages 43–54.
[13] J. Kasprzak and M. Brandejs. Improving the Reliability of the Plagiarism Detection System: Lab Report for PAN at CLEF 2010. *Proc. of PAN10*.
[14] B. Pôssas, N. Ziviani, B. A. Ribeiro-Neto, and W. Meira Jr. Maximal termsets as a query structuring mechanism. *Proc. of CIKM 2005*, pages 287–288.
[15] F. Scholer and S. Garcia. A case for improved evaluation of query difficulty prediction. *Proc. of SIGIR 2009*, pages 640–641.
[16] J. Seo and W. B. Croft. Local text reuse detection. *Proc. of SIGIR 2008*, pages 571–578.
[17] B. Stein and M. Hagen. Introducing the User-over-Ranking Hypothesis. *Proc. of ECIR 2011*, pages 503–509.
[18] X. Wu and V. Kumar. *The Top Ten Algorithms in Data Mining*. CRC Press, 2009.
[19] Y. Yang, N. Bansal, W. Dakka, P. G. Ipeirotis, N. Koudas, and D. Papadias. Query by document. *Proc. of WSDM 2009*, pages 34–43.