Capacity-constrained Query Formulation

Matthias Hagen and Benno Maria Stein

Faculty of Media
Bauhaus University Weimar, Germany

Abstract Given a set of keyphrases, we analyze how Web queries with these phrases can be formed that, taken altogether, return a specified number of hits. The use case of this problem is a plagiarism detection system that searches the Web for potentially plagiarized passages in a given suspicious document. For the query formulation problem we develop a heuristic search strategy based on co-occurrence probabilities. Compared to the maximal termset strategy [3], which can be considered as the most sensible non-heuristic baseline, our expected savings are on average 50% when queries for 9 or 10 phrases are to be constructed.

1 Introduction

The problem considered in this paper appears as an important sub-task of automatic text plagiarism detection. Plagiarized passages in a suspicious document can be found via direct comparisons against potential source documents. Todays typical source of plagiarism is the Web, which obviously contains too many documents for direct comparisons. The straightforward solution is to extract keyphrases from the suspicious document and to retrieve a tractable number of documents containing these phrases. These documents are considered as the best potential sources of plagiarism since they probably cover similar topics. Our contribution is a strategy for finding a family of "promising" Web queries whose combined results will be used for direct comparisons. The paper in hand does not deal with the complete plagiarism detection task; its focus is on the Web query pre-computation step.

The number of source documents a detection system can consider for direct comparisons is constrained by some processing capacity k. If all the extracted keyphrases (usually about 10) from the suspicious document are submitted as one single Web query, probably too few documents are returned with respect to k. Similarly, queries containing only few of the extracted phrases are likely to yield a huge number of hits; from these only a fraction, typically the Web search engine's top-ranked results, could be processed by the detection system. We argue that the probability to find potential plagiarism sources becomes maximum if the combined result list length of the promising queries is in the order of magnitude of the processing capacity k. We term this argument *the-user-knows-better hypothesis* or, more formally, *user-over-ranking hypothesis*: the detection system as the "user" of the search engine simply processes all of the promising queries' combined results, this way avoiding any search engine ranking issues that cannot be influenced.

Under the user-over-ranking hypothesis the CAPACITY CONSTRAINED QUERY FORMULATION problem analyzed in this paper is defined as follows. Given is (1) a set W of keyphrases, (2) a Web search engine's query interface, and (3) an upper bound k on the number of desired documents. The task is to find a family $\mathcal{Q} \subseteq 2^W$ of queries, together returning at most k documents and containing all the phrases of W, if possible. Obviously, a series of queries must be submitted to the search engine for finding $\mathcal{Q}$, and we focus on the following optimization problem from the detection system's perspective: What strategy

Table 1. Keyphrase-document-relationships for the example scenario.

Keyphrase	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}
w_1	•	•		•	•					•
w_2		•				•			•	
w_3	•	•	•	•	•	•	•		•	
w_4	•			•		•	•	•		•
w_5	•		•		•	•	•	•	•	

minimizes the average number of submitted queries? Two previous papers analyze related query formulation problems: Shapiro and Taksa [4] suggest the rather simple open end query formulation approach, for generating queries that each return at most an upper bound number of hits. Unfortunately, it is straightforward to construct situations in which the approach fails although adequate queries exist. A more involved maximal termset query formulation method is proposed by Pôssas et al. [3]; we use an adapted version as our baseline.

2 Basic Definitions and the Baseline Method

Any subset $Q \subseteq W$ can be submitted as a Web query, with the notion that phrases are included in quotation marks. An engine's reply contains an estimation l_Q for the total number of results matching the query. Our task is to find a simple family $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$; *simple* means that $Q_i \not\subseteq Q_j$ for any $i \neq j$. Altogether $\mathcal{Q}$'s queries should not yield more than k results. From k we will derive an upper bound $l_{\max}$ with the notion that a single query Q is promising iff $l_Q \leq l_{\max}$. Another lower bound $l_{\min}$ is introduced for convenience reasons. We say that for $l_Q < l_{\min}$ the query Q is *underflowing*, whereas for $l_Q > l_{\max}$ it is *overflowing*. Queries that are neither under- nor overflowing are *valid*. A valid query Q is *minimal* iff omitting any phrase will result in an overflowing query. We propose the family $\mathcal{Q}_{\text{lo}}$ of all the minimal valid queries as a solution to CAPACITY CONSTRAINED QUERY FORMULATION. $\mathcal{Q}_{\text{lo}}$ is simple and covers all phrases that are contained in any valid query. During the computation we count the overall number $cost$ of submitted Web queries.

Consider the following example scenario: Given are 10 indexed documents $d_1, \ldots, d_{10}$ and the set $W = \{w_1, \ldots, w_5\}$ with the keyphrase-document-relationships shown in Table 1. Note that, submitted as a query, the set W itself will not result in any hit. Figure 1 shows a part of the hypercube of the possible 2^5 queries; the valid queries for $l_{\min} = 3$ and $l_{\max} = 4$ are shown highlighted. The query $\{w_3, w_5\}$ is overflowing (six hits) whereas $\{w_1, w_5\}$ is underflowing (two hits). The family $\mathcal{Q}_{\text{lo}} = \{\{w_1, w_3\}, \{w_1, w_4\}, \{w_2\}, \{w_3, w_4\}, \{w_4, w_5\}\}$ corresponds to the lower border in Figure 1; it will return all documents except d_3.

As a baseline we adapt the maximal termset approach by Pôssas et al. [3], but we do not use GENMAX as a subroutine to enlarge promising keyphrase subsets. Instead, we adopt

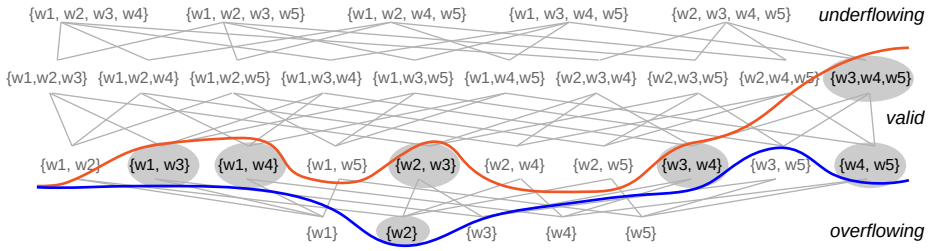

Figure 1. Hypercube of possible queries in the example scenario.

Input: W, $l_{\min}$, $l_{\max}$ **Output:** $\mathcal{Q}_{\text{lo}}$

if W is not overflowing **then**
  $\mathcal{Q} \leftarrow \{\{w\} : w \in W \text{ and } \{w\} \text{ is valid}\}$
  $C_1 \leftarrow \{\{w\} : w \in W \text{ and } \{w\} \text{ is overflowing}\}$
  $i \leftarrow 1$
  while $C_i \neq \emptyset$ **do**
    for all $Q, Q' \in C_i, |Q \cap Q'| = i - 1$ **do**
      $Q_{\text{cand}} \leftarrow Q \cup Q'$
      if $Q_{\text{cand}} \setminus \{w\} \in C_i$ for all $w \in Q_{\text{cand}}$ **then**
        if Q_{cand} is valid **then** $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{Q_{\text{cand}}\}$
        if Q_{cand} overflows **then** $C_{i+1} \leftarrow C_{i+1} \cup \{Q_{\text{cand}}\}$
    $i \leftarrow i + 1$
  output $\mathcal{Q}$

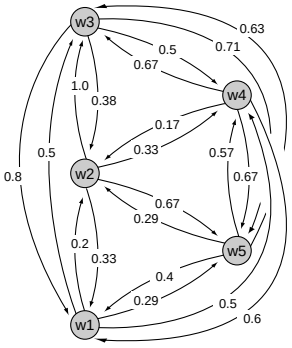

Figure 2. Left: Apriori algorithm. Right: co-occurrence graph of Table 1's example scenario.

the classic Apriori algorithm that also stems from the field of frequent itemset mining [1] (cf. Figure 2 (left) for a basic pseudo-code listing). Apriori traverses the search space of all possible queries (cf. Figure 1) in a level-wise manner. Whenever the validity of a query Q has to be checked Apriori submits it to the Web search engine and obtains l_Q. The problem now is to find an appropriate $l_{\max}$. We start with $l_{\max} \leftarrow k$, compute $\mathcal{Q}_{\text{lo}}$ using Apriori and count the results all queries in $\mathcal{Q}_{\text{lo}}$ return. Usually this will be too many and we then use a binary search for an appropriate $l_{\max}$ by halving the value as long as the computed $\mathcal{Q}_{\text{lo}}$ returns too many results. If at one intermediate step a $\mathcal{Q}_{\text{lo}}$ returns approximately k results (we set the bound to at least 90%), the computation stops and outputs this $\mathcal{Q}_{\text{lo}}$. If eventually too few results are returned we enlarge $l_{\max}$ according to the binary search paradigm. Note that whenever we enlarge $l_{\max}$ for the first time, all of the remaining evaluations have already been done during the previous step such that no further queries have to be submitted.

3 Outline of the Heuristic Search Strategy

To improve the performance of the baseline with respect to the number of submitted queries we propose a heuristic that mimics Apriori's workflow but tries to avoid submission of Web queries. In a pre-processing step the heuristic derives a directed edge- and vertex-weighted co-occurrence graph G_W. The graph contains a vertex v_w for each keyphrase $w \in W$. The weight of v_w is set to $l_{\{w\}}$. An edge $e = v_w \rightarrow v_{w'}$ from v_w to $v_{w'}$ gets as weight the yield factor $\gamma(e) = l_{\{w,w'\}}/l_{\{w\}}$. Semantics: the yield factor multiplied by the weight of v_w gives the yield of Web hits when w' is added to the query $\{w\}$. Note that the yield factor is reminiscent of the co-occurrence probability for the keyphrases w and w'; G_W is reminiscent of a mutual information graph (cf. Figure 2 (right)). Obtaining G_W during pre-processing involves the same computations and Web queries that Apriori processes during the first two levels (queries with at most two keyphrases).

After the pre-processing step the heuristic starts an Apriori-like candidate generation on the third level (queries containing three phrases). Hence our technique does not save queries on the first two levels compared to Apriori, and no overall savings are achievable for initial keyphrase sets of size three. However, from Level 3 on G_W is used to assess a query before submitting it as a Web query. Assume we are on some level $i \geq 2$ (queries with i keyphrases). All processed queries Q from lower levels have a stored value est_Q indicating an estimation of the length of their result lists. Let the current candidate query Q_{cand} be obtained by merging queries Q and Q' from level $i-1$. Before submitting Q_{cand} as a Web query (like the baseline would do) the estimation $est_{Q_{\text{cand}}} = est_Q \cdot \text{avg}\{\gamma(v_w \rightarrow$

Table 2. Experimental results.

	Number of keyphrases:	4	5	6	7	8	9	10	15
1	Complete query overflows	647	535	444	351	274	229	212	18
2	Remaining documents	465	567	668	752	838	883	900	757
3	Avg. $cost$ heuristic	10.33	16.33	26.25	39.93	62.11	98.73	150.37	1 379.33
4	Avg. $cost$ baseline	11.09	19.66	36.33	64.92	117.05	207.20	342.95	3 020.34
5	**Micro-averaged cost ratio**	**0.93**	**0.83**	**0.72**	**0.62**	**0.53**	**0.48**	**0.44**	**0.46**

$v_{w'}) : w \in Q\}$ is computed. Submitting Q_{cand} as a Web query and obtaining the engine's $l_{Q_{\mathrm{cand}}}$ is done iff the estimation $est_{Q_{\mathrm{cand}}}$ is in the order of $l_{\max}$. Otherwise, no Web query is submitted; $est_{Q_{\mathrm{cand}}}$ is remembered. This heuristic is not guaranteed to output the same family $\mathcal{Q}_{\mathrm{lo}}$ as the baseline. However, experiments show good conformity of the output with the baseline's $\mathcal{Q}_{\mathrm{lo}}$ while saving a significant number of queries at the same time (see below).

4 Experimental Analysis and Conclusion

We experimentally compare our heuristic and the baseline as follows: for a given document we extract a number of keyphrases and then formulate queries using these phrases. Keyword extraction is managed by the head noun extractor [2]. Our document collection was obtained by crawling papers on computer science from major conferences and journals. We also added some books. From the established corpus we removed the documents for which we were not able to extract 10 reasonable keyphrases. Our test collection was formed by the 1112 remaining documents. We set the bounds $k = 1000$ and $l_{\min} = 1$. For each document of the test collection we had 7 runs of the baseline and our heuristic with 4, 5, ..., 10 extracted keyphrases. Another run was done on the 775 documents of our collection from which 15 reasonable keyphrases could be extracted. As Web search engine we used the Microsoft Bing API. A typical Web query took about 300–550ms.

Table 2 shows the results of our experiments. For small keyphrase sets the complete query with all phrases often overflows (cf. first row). We filtered out such keyphrase sets and derived the statistics (rows 3 to 5) for the remaining documents (number given in second row). In rows 3 and 4 we state the average number $cost$ of Web queries the approaches submitted. The average ratio of submitted queries of the heuristic over the baseline is given in row 5. The possible savings are substantial: Even for 7 phrases our heuristic saves 30–40% of the queries and for 9 phrases possible savings reach 50%. Altogether, our results suggest that a near real-time plagiarism detection service with processing capacity $k = 1000$ should try to extract 9 or 10 keyphrases as then the heuristic computes $\mathcal{Q}_{\mathrm{lo}}$ in about 1 minute.

Bibliography

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. of VLDB 1994*, pages 487–499.

[2] K. Barker and N. Cornacchia. Using noun phrase heads to extract document keyphrases. In *Proc. of AI 2000*, pages 40–52.

[3] B. Pôssas, N. Ziviani, B. A. Ribeiro-Neto, and W. Meira Jr. Maximal termsets as a query structuring mechanism. In *Proc. of CIKM 2005*, pages 287–288.

[4] J. Shapiro and I. Taksa. Constructing web search queries from the user's information need expressed in a natural language. In *Proc. of SAC 2003*, pages 1157–1162.