Search by Screenshots for Universal Article Clipping in Mobile Apps

KAZUTOSHI UMEMOTO, Kyoto University
RUIHUA SONG, Microsoft Research Asia
JIAN-YUN NIE, University of Montreal
XING XIE, Microsoft Research Asia
KATSUMI TANAKA, Kyoto University
YONG RUI, Microsoft Research Asia

To address the difficulty in clipping articles from various mobile applications (apps), we propose a novel framework called UniClip, which allows a user to snap a screen of an article to save the whole article in one place. The key task of the framework is *search by screenshots*, which has three challenges: (1) how to represent a screenshot; (2) how to formulate queries for effective article retrieval; and (3) how to identify the article from search results. We solve these by (1) segmenting a screenshot into structural units called blocks, (2) formulating effective search queries by considering the role of each block, and (3) aggregating the search result lists of multiple queries. To improve efficiency, we also extend our approach with learning-to-rank techniques so that we can find the desired article with only one query. Experimental results show that our approach achieves high retrieval performance ($F_1 = 0.868$), which outperforms baselines based on keyword extraction and chunking methods. Learning-to-rank models improve our approach without learning by about 6%. A user study conducted to investigate the usability of UniClip reveals that ours is preferred by 21 out of 22 participants for its simplicity and effectiveness.

CCS Concepts: • **Information systems** → **Retrieval on mobile devices**; • **Human-centered computing** → *Interaction design*; User studies;

Additional Key Words and Phrases: Universal clipping, search by screenshots, article clipping, mobile apps

ACM Reference Format:
Kazutoshi Umemoto, Ruihua Song, Jian-Yun Nie, Xing Xie, Katsumi Tanaka, and Yong Rui. 2017. Search by screenshots for universal article clipping in mobile apps. ACM Trans. Inf. Syst. 35, 4, Article 34 (June 2017), 29 pages.
DOI: http://dx.doi.org/10.1145/3091107

1. INTRODUCTION

The use of mobile devices, for example, smartphones and tablets, has surpassed PCs since 2010 and continues to grow, according to a report[1] from the Internet Trends Conference. In 2013, the number of smartphones shipped is more than three times as

[1]http://www.kpcb.com/internet-trends.

This work was supported in part by JSPS Grants-in-Aid for Scientific Research (No. 17K12787).
Authors' addresses: K. Umemoto (current address), University of Tokyo, National Institute of Information and Communications Technology, Tokyo 1538505, Japan; email: umemoto@tkl.iis.u-tokyo.ac.jp; R. Song (corresponding author) and X. Xie, Microsoft Research Asia, Beijing 100080, China; emails: {rsong, xingx}@microsoft.com; J.-Y. Nie, University of Montreal, Quebec H3C 3J7, Canada; email: nie@iro.umontreal.ca; K. Tanaka, Kyoto University, Kyoto 6068501, Japan; email: tanaka.katsumi.85e@st.kyoto-u.ac.jp; Y. Rui (current address), Lenovo, NC 27560, United States; email: yongrui@lenovo.com.

DOI: http://dx.doi.org/10.1145/3091107

Fig. 1. How UniClip works for users.

many as that of PCs. This report also shows that the fraction of page views via mobile devices has been rapidly increasing year by year (14% in 2013 to 25% in 2014).

Different from PCs, the major way of reading on mobile devices is not browser-centered, but application-driven. Information providers develop their own mobile applications (so-called apps) and publish a large number of interesting articles through apps. Users install apps for different purposes. It is common that a user uses several apps to access information. This causes a problem that what a user reads or likes is scattered in different apps or buried by never-ending updating streams. As people have only limited time to read the full text of articles at a time, there are increasing demands for developing effective and user-friendly tools to save articles in one place from different apps.

Some note-taking apps, like OneNote, EverNote, and Pocket, have been developed to solve the problem. However, these solutions are far from satisfactory for several reasons. The note-taking apps usually depend on "Share," "Copy," or "Copy the link" interfaces to receive contents from other apps. However, the decisions on whether to support sharing with an app or what to share or copy is fully controlled by app developers. It is difficult for note-taking apps to partner with all reading related apps. For example, Facebook Paper and Klout do not support any note-taking apps. Instead, they prefer sharing articles with social networks. Some apps, such as Toutiao, only allow users to copy a bookmark of an article, including title, a link, or an image. Once the link expires, the users will lose their saved articles eventually. Furthermore, in a normal reading view, apps often hide the "Share" or "Copy" buttons. Thus, users have to take several actions to share or copy an article.

To address the difficulty in clipping articles from diverse apps in a universal way, we propose a service called UniClip (a shorthand for Universal Clipping). As shown in Figure 1, UniClip allows users to capture an article by taking a screenshot when they are reading in any app. The screenshot could be any part of the article that contains some text content. We observe that most articles, in particular popular ones, in mobile apps have copies on the Web. Thus, UniClip will automatically discover the article URL by leveraging Web search, extract the main contents from the URL by leveraging existing work [Zheng et al. 2007], and save them to the cloud storage, so that users can read the article later on.

The core research problem with UniClip is searching for a whole article from its partial screenshot, which we call *search by screenshots*. Search by screenshots has three main challenges: (1) how to represent an article screenshot for finding important structural units, (2) how to formulate effective search queries for retrieving the article, and (3) how to identify the article URL from search results. We propose methods to solve these challenges. As calling a search engine is costly and has transaction quota, we raise a more challenging task: can we use only one query to retrieve the full article? In this article, we also propose a machine learning based approach to these challenges. Our approach (1) applies a Conditional Random Field (CRF) model [Lafferty et al. 2001] for estimating the role of each block segmented by structural features, (2) selects

the most promising query from candidate queries generated from blocks by leveraging a learning-to-rank model [Liu 2009], and (3) reranks search results through the use of a learning-to-rank model to select the most likely one.

We make the following contributions in this article.

—We formulate the *search by screenshots* task, which receives a screenshot of a partial article as input and outputs the whole article found on the Web, which can enhance the potential of screenshots.
—We propose methods to solve the challenges of search by screenshots. Our approach segments a screenshot into structural units called blocks, formulates effective search queries by taking into account the role of each block, and aggregates the search result lists of multiple queries. In addition, we extend our approach to improve efficiency so that we can find the desired article with only one query. Our extended approach utilizes learning-to-rank techniques to select the most promising query and to find the most likely URL from a single search result list.
—We evaluate each component of our approach and the overall performance on purpose-built datasets. Results show that when we use all the generated queries, our approach achieves high retrieval performance ($F_1 = 0.868$) and outperforms state-of-the-art baselines that formulate keyword- and chunk-based search queries. We also confirm that when only a single query is affordable due to efficiency, our extended approach based on learning-to-rank improves ours without learning by about 6% in terms of the F_1 measure.
—We implement UniClip as an Android app and investigate its usability through a user study. Study participants are satisfied with the quality of retrieved results for 69.5% of the articles they clip. Furthermore, our clipping app is preferred by 21 out of 22 participants for its simplicity and effectiveness and is rated significantly higher than conventional clipping methods.

The rest of this article is structured as follows. Section 2 describes existing work related to ours. Section 3 formalizes the search by screenshots task and describes our approach to solving this task. Sections 4 and 5 report experiments and a user study that we conduct, respectively. We conclude the article with discussion of future work in Section 6.

2. RELATED WORK

Research areas related to this work are as follows: (1) finding similar or duplicate documents, (2) handling long queries, and (3) leveraging images as search queries. This section overviews each of these areas.

2.1. Search for Similar Documents

Some researchers have addressed a research problem called *query by document*. The objective of query by document is to retrieve documents similar to a given document. One of the biggest challenges in query by document is how to select representative words or phrases from the document. Yang et al. [2009] propose methods that assign scores to phrases extracted from a given document by calculating tf-idf [Büttcher et al. 2010] and mutual information [Church and Hanks 1990]. Their method then selects top-N (specified by users) phrases to formulate a search query for retrieving similar documents. Weng et al. [2011] point out that queries formulated by this approach would lose the semantic information of a given document. They attempt to solve this problem by decomposing a document into both a compact vector and document-specific keywords; the former is used for finding related documents efficiently, while the latter is for reranking related documents. They adopt locality sensitive hashing [Andoni and Indyk 2008] to index the compact vectors for quickly finding a set of related documents and rerank documents by document-specific words.

The objective of query by document and that of search by screenshots look similar but are actually different from one another. Our goal is to find a not similar but exactly the same article as a given screenshot. Thus, we need to consider a different query formulation method from theirs, for example, keyword and chunk queries. We show in Section 4.4.3 that queries formulated by our method are more effective for search by screenshots than those generated by state-of-the-art keyword extraction [Mihalcea and Tarau 2004] and chunking [Koeling 2000] methods.

Another research area related to our work is *(near-) duplicate detection* [Cho et al. 2000; Chowdhury et al. 2002; Deng and Rafiei 2006; Henzinger 2006], which is aimed at finding documents that have the same contents (except for slight differences such as ads and timestamps). Applications of duplicate detection especially for Web search include filtering out duplicate documents in an indexed document collection from search results and preventing crawlers from fetching documents linked by duplicate documents. One of the most recent approaches has been proposed by Manku et al. [2007]. Their method calculates documents' fingerprints through locality sensitive hashing [Charikar 2002] and finds near-duplicate documents by solving the hamming distance problem with fingerprint collection.

While the aim of duplicate detection is similar to ours, methods proposed for the former assume that the full text of documents to be compared is available. However, the input of search by screenshots is a screenshot that contains part of an article. In addition, we would like to avoid fetching the full text of Web pages in search results in terms of the processing time of our approach. Therefore, we decide to leverage learning-to-rank methods to efficiently find the URL of exactly the same article from the search results of formulated queries.

2.2. Long Queries

As we formulate queries that are long enough to identify the target article, our work is related to the following research areas on long queries.

Some studies have addressed better understanding a natural language long query, like the description of a query in TREC datasets, because such a query is not effective or efficient in search. Kumaran and Allan [2007] use Mutual Information (MI) to select 10 subqueries and present them to the user to choose from. Bendersky and Croft [2008] propose automatically finding key concepts in long queries. Some researchers have studied query quality predictors and automatic selection of reduced queries [Balasubramanian et al. 2010a, 2010b; Kumaran and Carvalho 2009]. For example, Kumaran and Carvalho [2009] apply learning-to-rank framework in reducing long queries using query quality predictors, such as Query Clarity [Cronen-Townsend et al. 2002] and Query Scope [He and Ounis 2006].

Phrase extraction has also been used to extract some representative phrases from a document and to find related documents containing these phrases. Some studies [Manning and Schütze 1999; Medelyan and Witten 2006] utilize statistical information to identify suitable phrases, while others also leverage the relationships between phrases [Frantzi 1997] and apply learning algorithms in the extraction process [Turney 2000; Tomokiyo and Hurst 2003]. Mihalcea and Tarau [2004] propose TextRank, a graph-based ranking model, for text processing such as keyword extraction and document summarization.

The objective of the preceding studies is to find similar or relevant documents, and thus most words in the long query are not required to appear in the target document. However, we aim to find exactly the same document corresponding to the screenshot query. Therefore, any word should appear in the document, including a stop-word, which are usually filtered out in the approaches to long queries. As a result, their approaches are expected to be less effective than ours specifically designed for the search

by screenshots task. In this article, we reimplement keyword extraction methods, such as TextRank [Mihalcea and Tarau 2004], as our baselines. Experimental results confirm that our proposed methods considerably outperform keyword-based methods in search by screenshots.

2.3. Search by Images

As well as textual queries, multimodal queries (such as images, audio, and videos) have been widely explored as the input of search systems [Xie et al. 2008]. We survey existing work in this area both from industry and academia, especially on image queries like ours.

Google, one of the most popular Web search engines, provides an image search system[2] that can accept images as queries. When a user issues an image query to this system, it returns images similar to the given one and Web pages that contain the same image as the input. A retrieval system proposed by Fan et al. [2005] receives multimodal queries comprising images and optional text and returns images similar to the input queries. This system overcomes one of the problems of conventional content-based image retrieval [Smeulders et al. 2000], that is, the difficulty in applying large-scale images due to high computational cost, by first eliminating irrelevant images by using textual features and then ranking the remaining images by using visual features. Sikuli, proposed by Yeh et al. [2009], allows users to capture a screenshot of a windows element (such as buttons and icons) on the screen. Given an element's screenshot, Sikuli searches for documentation about the element by extracting visual features from the screenshot. This enables users to get help for elements without knowing their names. Images captured by people with their mobile devices have been utilized to recognize the people's locations and provide Web pages containing information on nearby areas [Yeh et al. 2004].

3. SEARCH BY SCREENSHOTS

This section begins by outlining the task settings of search by screenshots, followed by challenges of this task to be tackled, and our approach to these challenges.

3.1. Task Settings

We specify the input/output of search by screenshots specialized for articles in the following.

—**Input:** Search by screenshots receives as input a screenshot of an article displayed in a mobile device. It can be any part of the whole article, for example, the beginning part including its title or the middle part containing only a few paragraphs. A screenshot is regarded as an invalid input if it (a) consists of multiple articles or (b) contains no text of an article, since the ideal output does not exist for this screenshot.
—**Output:** Given an article's screenshot, the ideal output for this input is a URL of exactly the same article. Different Web pages sometimes post an identical article (with or without modification). If different pages contain the text content of a screenshot of an article, we regard all of them as valid output for that screenshot.

3.2. Challenges

Our basic idea for solving search by screenshots is as follows. First, we understand the contents in a given screenshot. Second, we formulate search queries from the input screenshot. Third, we identify the ideal URL from the search results for those queries.

[2]https://images.google.com/.

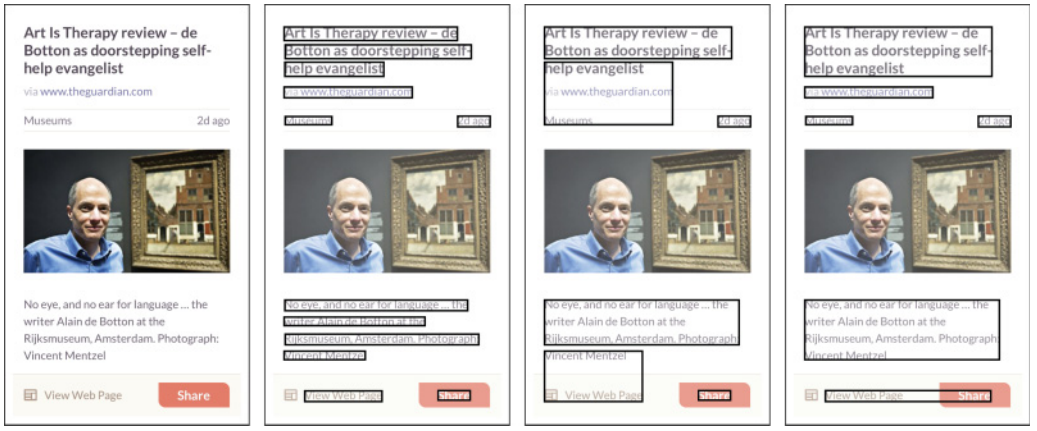
To make this idea more efficient and effective, there are three primary challenges to be tackled as described in the following.

(1) **How to represent a screenshot:** Since conventional Web search engines accept only textual queries as input for retrieving Web pages, we need a tractable representation of a given screenshot. A possible solution is to identify the text lines in a given screenshot by applying Optical Character Recognition (OCR). However, as the screen of mobile devices is small, lines detected by OCR often contain only a few words. As a result, each structural unit of articles, such as title and paragraphs, is often broken into several lines. Thus, individual lines are not long enough to formulate appropriate search queries that can retrieve the original article. Furthermore, articles usually include units that are not related to their main contents, for example, headers, navigation bars, and advertisements (ads). Queries formulated from such unrelated units are likely to return irrelevant search results. To address these issues, it is necessary to structure the content in a screenshot and identify important units before formulating search queries.
(2) **How to formulate search queries:** A straightforward approach to query formulation is extracting keywords from the screenshot text. However, given the objective of search by screenshots, that is, returning exactly the same article, much contextual information will be lost if we represent a screenshot with a set of keywords. For example, stop-words would be ignored through keyword extraction, although they might be useful to retrieve documents that use the same phrases as the screenshot's article. Efficiency is another challenge for query formulation. While issuing more search queries increases the likelihood of retrieving the original article, it also results in an increase in processing time. Therefore, a query formulation algorithm should achieve acceptable accuracy with as few search queries as possible.
(3) **How to identify the best URL:** While one may try to identify the original article by fetching the full text of each search result and comparing its similarity with the text in a given screenshot, such an approach consumes much more time for processing. Instead, investigating a search results page to find the original article will save the processing time. The issue here is that the original article is not always ranked at the top of search results, since a number of features, for example, relevance [Robertson et al. 1994], importance [Page et al. 1999], diversity [Agrawal et al. 2009], and freshness [Dai et al. 2011], are involved in document ranking. This leads to a new challenge of how to aggregate search result lists of multiple queries. In addition, when only a single query is affordable due to efficiency, it is necessary to identify the best URL from a single search result list, which is more challenging.

3.3. Screenshot Representation

3.3.1. Block Segmentation. Given a screenshot, we first apply OCR to identify the text in a given screenshot. A recognition result returned by our used OCR engine [Huo and Feng 2003] consists of the detected *language* and a list of *regions*. A region contains a list of *lines* and its bounding box; a line contains a list of *words* and its bounding box; a word also has its bounding box and a score representing the degree of recognition *confidence* ($\in [0, 1]$). Note that our approach is also applicable to other OCR engines if their return recognition results about lines.

To obtain a better representation of a screenshot, we segment it into structural units, which we call *blocks*. As units are often across more than one line in the screen of a mobile device, regarding each line as one block cannot capture the accurate structure for a given screenshot. This is illustrated in Figure 2(b), where lines detected by our OCR engine for an input screenshot in Figure 2(a) are marked by rectangles. One region may also be a block since our OCR engine merges adjacent lines with similar



(a) Input screenshot (b) OCR lines (c) OCR regions (d) Segmented blocks

Fig. 2. Before and after applying our segmentation algorithm.

width into the same region. However, we find that regions detected by the OCR engine are typically far from satisfactory in terms of semantic structure. For Figure 2(a) as input, regions detected by our OCR engine are marked by rectangles in Figure 2(c). The article title (comprised of the first three lines) is split into two regions. In addition, a part of the bottom toolbar is wrongly merged with the last line of a body paragraph. Therefore, we propose a two-phase rule-based method for better block segmentation.

(1) We first build candidate blocks by iteratively merging adjacent similar lines. The detailed process is shown in Algorithm 1. At the fifth line in this algorithm, two lines are considered similar if (a) they have similar heights, (b) the distance is less than a threshold, and (c) they have the same alignment. Take lines in Figure 2(b) as an example. The third line is deemed similar to the second line because they have similar heights, close to each other, and both are left aligned. The fourth line is smaller than, and far from, the third line. In this way, the second and third lines are merged in the same block, while the fourth one is assigned to another block.
(2) We find that the first phase tends to oversegment a single paragraph into more than one block due to the difference in height of two lines on the boundary. Thus, we refine segmentation done by the first phase by merging adjacent blocks that share a similar structure. To this end, we consider block-level features, instead of line-level ones used in the first phase. Algorithm 2 shows the detailed process of block refinement. At the fifth line in this algorithm, two blocks are considered similar if (a) they share the same alignment, (b) they have similar font size (approximated by the average height of lines), or (c) the space between them is less than a threshold.

We decide threshold values for segmentation rules empirically. As shown in Figure 2(d), the preceding method can identify blocks for the screenshot in Figure 2(a) reasonably accurately.

3.3.2. Attribute Prediction. Not all blocks are equally important in retrieving the full article. Title is a unique identifier of an article. Body paragraphs constitute a large fraction of the article. However, some other blocks, such as ads or toolbars, do not affect the main contents of the article and thus are less relevant for search by screenshot. To identify important blocks from which we are more likely to formulate promising

ALGORITHM 1: BUILDBLOCKS($\mathcal{L}$)

Input: a sequence of lines $\mathcal{L}$ appearing in a recognition result
Output: a sequence of candidate blocks $\mathcal{B}_{\text{cand}}$

$\mathcal{B}_{\text{cand}} \leftarrow ()$;
$L_{\text{cur}} \leftarrow$ dequeue the first element from $\mathcal{L}$;
$B \leftarrow (L_{\text{cur}})$;
foreach $L_{\text{new}} \in \mathcal{L}$ **do**
    if L_{cur} and L_{new} look similar and are located closely **then**
        append L_{new} to the end of B;
    else
        append B to the end of $\mathcal{B}_{\text{cand}}$;
        $B \leftarrow (L_{\text{new}})$;
    end
    $L_{\text{cur}} \leftarrow L_{\text{new}}$;
end
append B to the end of $\mathcal{B}_{\text{cand}}$;
return $\mathcal{B}_{\text{cand}}$

ALGORITHM 2: REFINEBLOCKS($\mathcal{B}_{\text{cand}}$)

Input: a sequence of candidate blocks $\mathcal{B}_{\text{cand}}$
Output: a sequence of refined blocks $\mathcal{B}$

$\mathcal{B} \leftarrow ()$;
$j_{\text{from}} \leftarrow 0$;
while $i_{\text{from}} <$ Length($\mathcal{B}_{\text{cand}}$) **do**
    $B \leftarrow ()$;
    $j_{\text{to}} \leftarrow$ last index of blocks similar to $\mathcal{B}_{\text{cand}}[j_{\text{from}}]$;
    for $j \leftarrow j_{\text{from}}$ **to** j_{to} **do**
        append each line in $\mathcal{B}_{\text{cand}}[j]$ to the end of B;
    end
    append B to the end of $\mathcal{B}$;
    $j_{\text{from}} \leftarrow j_{\text{to}} + 1$;
end
return $\mathcal{B}$

search queries, we predict the attribute of each block by applying the CRF [Lafferty et al. 2001], which is a state-of-the-art method for modeling structured data. In this article, we target three attributes: *title*, *body*, and *others*.

We regard each OCR line as a unit of observation data in our CRF model. The CRF allows us to naturally encode the relationship among successive lines into features. As shown in Table I, we extract nine features in total from different points of view. The first group contains three features related to title attribute prediction. For example, as title is often described with a larger font than other sentences, we include in this group the font size of each line (approximated by the height of its bounding box). We also consider the vertical position of a line, with the expectation that the title often appears at the top of articles. The second group contains three features related to body attribute prediction. As body blocks are likely to contain more sentences than other blocks, we include to this group the number of words and the presence or absence of punctuation. The last group contains three features, intended to predict whether two adjacent lines have the same attribute or not. We consider the consistency of line styles, such as alignment and height, for this prediction. All features are discretized with empirically

Table I. List of Features used for Attribute Extraction

Group	Name	Description	Bins
Title	FontSize	Font size of a line (approximated by the mean height of words in the line)	Small, Medium, Large
	Confidence	Recognition confidence averaged by words in a line	Low, Middle, High
	VerticalPosition	Vertical appearance position of a line	Beginning, Middle, Ending
Body	WordCount	Number of words appearing in a line	Less, 2, 3, 4, 5, More
	Punctuation	Presence or absence of punctuation in a line, for example, ‘,”, “.”, and “?”	True, False
	LetterCase	Presence or absence of letter styles in a line, for example, lower/upper case and number only	True, False
Consistency	Alignment	Coherence of (any of left-, right-, and center-) alignment of two successive lines	N/A, Mismatch, Match
	Distance	Distance of the top of the current line from the bottom of the previous line	N/A, Close, Near, Far
	Height	Difference in height between two successive lines	N/A, Similar, Different

defined thresholds, as was done in Ageev et al. [2011]. This makes the features more tractable by the CRF model. All bins for discretization are also shown in Table I.

Once attributes are assigned to all lines in a block, we determine the attribute of that block by majority voting. If more than one attribute label gets the most votes, we use the first occurred line attribute as the block attribute.

3.4. Query Formulation

As described in Section 3.2, selecting keywords to formulate search queries will lose information useful for retrieving the original article. In this section, we first propose a simple method that formulates *phrase* queries, that is, a sequence of words enclosed with double quotes, from each block. Next, we propose an advanced method that employs different strategies based on block attributes. Finally, we propose a method of selecting the most promising one from generated queries.

3.4.1. Simple Method. Our basic idea is to formulate phrase queries. As search engines perform the exact-match algorithm for such queries, we can leverage contextual information, e.g., term order and proximity, in an article's screenshot. When experimenting with phrase queries, we notice that a query is not discriminative enough if it is too short. For example, single-word phrase queries often return noisy documents. Thus, we restrict the minimum length of formulated queries to be σ_{qmin}. Although longer queries are more useful for capturing contextual information, we observe that search engines have their internal limitations on query length due to efficiency and sometimes do not perform exact-match for too long queries. Thus, we also restrict the maximum query length to be σ_{qmax}.

The pseudocode of formulating queries from one block is shown in Algorithm 3. This algorithm first cleans the text in a given block by removing punctuation. Then, we move the word selection window whose width is $n_{\max}$ along the text sequence and generate substrings without overlaps. For a substring s, if the word length of s is in $[n_{\min}, n_{\max}]$, we quote s to formulate a phrase query. We pass to this algorithm $n_{\min} = \sigma_{\text{qmin}}$ and $n_{\max} = \sigma_{\text{qmax}}$ as additional arguments for controlling the query length. We call queries formulated by this algorithm *simple* queries.

ALGORITHM 3: GENERATEQUERIES(B, $n_{\min}$, $n_{\max}$)

Input: a block B from which queries are generated;
min. and max. query lengths $n_{\min}$ and $n_{\max}$ in words
Output: a sequence of phrase queries Q

```
Q ← ();
t ← cleaned text (without punctuation) in B;
while Length(t) > 0 do
    s ← select min(n_max, Length(t)) words from t;
    t ← t[Length(s), Length(t) − 1];
    if Length(s) ≥ n_min then
        append quoted s to the end of Q;
    end
end
return Q
```

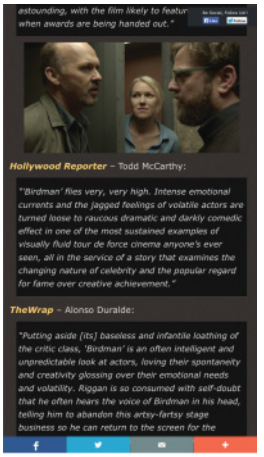


Fig. 3. Quotes from other articles.

3.4.2. Hybrid Method. In some cases, simple queries from body blocks are not unique enough to retrieve desirable articles. For example, the first long paragraph in Figure 3 quotes a review from Hollywood Reporter and the second long paragraph quotes a review from TheWrap. Queries formulated from such paragraphs will retrieve the original reports or some other articles that quote those reports. Fortunately, we find that different articles are less likely to share more than one paragraph.

To deal with this issue, we formulate *compound* queries by concatenating two simple queries generated from two distinct body blocks. More specifically, we first formulate a sequence of half-length simple queries Q_i from each body block B_i by

$$Q_i = \text{GENERATEQUERIES}\left(B_i, \frac{\sigma_{\text{qmin}}}{2}, \frac{\sigma_{\text{qmax}}}{2}\right).$$

Then, we iteratively dequeue $q_{i,m}$ from Q_i and $q_{i+1,n}$ from Q_{i+1} to formulate a compound query by combining $q_{i,m}$ and $q_{j,n}$ with a whitespace character as a delimiter. When there is no half-length query left in Q_i or Q_{i+1}, we select another sequence Q_{i+2} in order of appearance and continue to formulate compound queries. Note that when there remain half-length queries in only one sequence, we formulate compound queries from the single block.

Overall, search queries are formulated in a hybrid manner, depending on the attributes of blocks. We formulate compound queries for body blocks, whereas we still use simple queries for title and other blocks. The reason is that it is risky to combine a high quality block with a low quality block. A title block usually generates high quality queries, which are usually discriminative enough to identify the article, whereas most others blocks generate low quality queries. We do not exclude others blocks from query formulation resources, because they may contain title and body blocks misclassified by attribute prediction.

Given a screenshot and its blocks with attributes shown in Figure 4, the Simple and Hybrid methods formulate search queries listed in Table II, where the query length parameters are set to $\sigma_{\text{qmin}} = 4$ and $\sigma_{\text{qmax}} = 14$ (the same values as those used in our experiments in Section 4). The Simple method formulates simple queries from each block in the order of appearance. From the last block, two simple queries are formulated because the number of words in this block is larger than σ_{qmax}. No query is formulated from some blocks (e.g., the first and second ones) containing words less than σ_{qmin}. In the Hybrid method, a simple query "`Ferguson Reveals a Twitter Loop`"

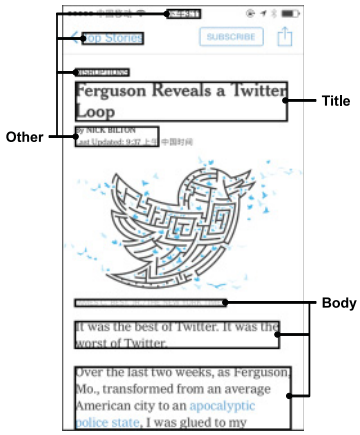


Fig. 4. Extracted attributes.

Table II. Queries Formulated by the Simple and Hybrid Methods from the Screenshot Shown in Figure 4

Method	Query
Simple	"Ferguson Reveals a Twitter Loop"
	"By NICK BILTON Last Updated 9 37 Eff"
	"JAMES C BEST JR THE NEW YORK TIMES"
	"It was the best of Twitter It was the worst of Twitter"
	"Over the last two weeks as Ferguson Mo transformed from an average American city to"
	"an apocalyptic police state I was glued to my"
Hybrid	"Ferguson Reveals a Twitter Loop"
	"JAMES C BEST JR THE NEW YORK TIMES" "It was the best of Twitter It"
	"was the worst of Twitter" "Over the last two weeks as Ferguson"
	"Mo transformed from an average American city" "to an apocalyptic police state"
	"By NICK BILTON Last Updated 9 37 Eff"

Table III. List of Features Used for Selecting Promising Query

Name	Description
Confidence	Recognition confidence for words in query
FontSize	Font size of words in query
BlockAttribute	Attribute of block from which query is generated
BlockPosition	Vertical position of block from which query is generated
WordPosition	Vertical position of words in query
QueryLength	Length of query (in terms of characters and words)
QueryScore	Score (of tf, tf-idf, and BM25) for words in query

is formulated from the title block first, followed by compound queries (e.g., "JAMES C BEST JR THE NEW YORK TIMES" "It was the best of Twitter It") from body block pairs and a simple query "By NICK BILTON Last Updated 9 37 Eff" from the others block.

3.4.3. Query Selection. In real applications, we may have limited quota of queries. Even if there is no limitation, it is still important to reduce the number of issued queries by considering efficiency. Thus, we propose a method to select queries in terms of how promising the query will retrieve the target URL.

After generating candidate queries, we select the most promising one by ranking them via a learning-to-rank approach [Liu 2009]. The features used for ranking queries are shown in Table III. As low confidence for words suggests misrecognition of our OCR engine, phrase queries containing such words are unlikely to return the original article. Thus, we include the average recognition confidence of query words as one feature (named Confidence). The FontSize feature reflects the rationale that words written in a large font size are considered more important than those with a small font size. The BlockAttribute feature is intended to put more importance on queries generated from title and body blocks than those from others. Queries generated from upper blocks may have different characteristics compared to those from lower ones. Thus, we consider the vertical positions of blocks and words from which a query is generated (BlockPosition and WordPosition). Since longer queries are likely to be more appropriate to our task than short ones, we count the number of characters/words in each query (QueryLength). Finally, we measure the importance of generated queries by using

word weighting functions such as tf-idf [Büttcher et al. 2010] and BM25 [Robertson et al. 1994] (QueryScore). To estimate document frequency for each word, we use the ClueWeb09 collection,[3] which contains around one billion Web pages.

When training a learning-to-rank model on our training dataset, we simply use binary labels ("relevant" and "irrelevant") for annotating the relevance of each generated query. The following annotation criterion is used for learning-to-rank queries: a query that our method generates for a given screenshot is regarded as relevant if the top-K search results retrieved for the query contain at least one ground-truth URL for the screenshot.

3.5. Result Aggregation

In this section, we first describe how to aggregate search result lists of multiple search queries to find the target article. Next, we propose a method of identifying the target article from a single search result list, with the aim of improving efficiency or for the application with limited quota of issued queries.

3.5.1. Multiple Search Result Lists. Once queries Q are formulated from a given screenshot, we retrieve the top-K search results by issuing each query to a Web search engine. We observe that good queries often return the original article at high ranks, while bad queries return diverse search results of incorrect pages. Thus, we use Borda Count [Borda 1781] to aggregate the search result lists of multiple queries. Specifically, the Borda Count score for a search result at rank $k \in [1, K]$ is given by

$$\mathrm{BordaCount}(q, k) = \frac{1}{\sqrt{k}}.$$

When attributes are available, queries composed from title or body blocks are more likely to be good queries than those from others. Therefore, we integrate query weights into the Borda Count formula as follows:

$$\mathrm{WeightedBordaCount}(q, k) = \frac{w(q)}{\sqrt{k}},$$

where $w(q)$ is the weight for the attribute block from which the query q is formulated. To distinguish from the Borda Count aggregation, we call it the Weighted Borda Count aggregation. The weight for each attribute can be viewed as the likelihood of good queries, that is, queries that return the target pages among the top-K results, generated from the attribute blocks. We learn the weight values using our training dataset, which resulted in 0.852, 0.778, and 0.252 for title, body, and other attributes, respectively.

3.5.2. Single Search Result List. Once the most promising query is selected by the approach described in Section 3.4.3, we retrieve the top-K search results by issuing that query to a Web search engine. Since the ideal page is not always ranked at the top of the search results, we cannot rely solely on the ranks when aggregating search results.

To solve this issue, we leverage the learning-to-rank technique to rerank the search results and select the most likely one from them. The features that we use for reranking are shown in Table IV. Since the ranking is informative, if not perfect, we add the Borda Count score for the original rank as one feature (named BordaCount). In addition, we calculate the similarity between each result and the screenshot from various points of view. Since some articles have descriptions of their source domains, we calculate the similarity between these domains and the domain part of the search result URL (DomainToDomain). When the screenshot does not contain such descriptions, we instead use a line in the screenshot that maximizes this similarity. We also prepare

[3]http://www.lemurproject.org/clueweb09.php.

Table IV. List of Features used for Reranking Search Results

Name	Description
BordaCount	Borda count for rank of search result
DomainToDomain	Similarity between result domain and extracted domain
TitleToTitle	Similarity between result title and extracted title
PathToTitle	Similarity between result path and extracted title
QueryToResult	Similarity between query and text in result
ContentToResult	Similarity between text in screenshot and text in result

another feature that represents the similarity between the search result title and the text in a title block (TitleToTitle). Since some Web sites include articles' titles for those URLs, the similarity of the path part of the search result URL and the title block's text is also calculated as another feature (PathToTitle). Furthermore, we also compare the whole text in the screenshot with the generated query and with the title and snippet of the search result to calculate the global-level overlap in words (QueryToResult and ContentToResult). Although it is possible to utilize the full text of the fetched Web page to calculate similarity, we do not add such features to avoid the additional processing time for page fetching.

We consider various measures for calculating similarity-based features. When calculating the DomainToDomain and PathToTitle features, we regard text as a list of characters and calculate character-based similarity by using the following two measures: Levenshtein distance and longest common subsequence. When calculating the TitleToTitle, QueryToResult, and ContentToResult features, we regard text as a bag of words and calculate word-based similarity by using the following two measures: Simpson's coefficient and cosine similarity.

As is the case in learning-to-rank queries, we use binary labels ("relevant" and "irrelevant") for annotating the relevance of each retrieved search result. The following annotation criterion is used for learning-to-rank search results: a search result that our method retrieves for a given screenshot is regarded as relevant if its URL is included in ground-truth URLs for the screenshot.

4. EXPERIMENTS

This section reports the results of experiments we conduct to evaluate the effectiveness of our approach to search by screenshots. In the experiments, we first evaluate each component of our approach and then evaluate the overall performance. As normality is not guaranteed for our experimental data, we use nonparametric significance tests in our analyses. In what follows, significant effects are reported on the significant level $\alpha = 0.05$.

4.1. Datasets

We create two datasets to objectively evaluate our approach to search by screenshots. One dataset contains 100 screenshots collected in May 2014, which is used for training machine learning models for attribute extraction, query formulation, and result aggregation. Another dataset consists of 200 screenshots collected in August 2014, aimed at testing the effectiveness and efficiency of our approach. The screenshots in these datasets are taken from different mobile apps (including news apps like NYTimes, aggregated news apps like Facebook Paper, and other popular apps like Etsy) and Web browsers for mobiles. As the Web is changing, the original article URLs for some screenshots had expired when we were conducting experiments in November 2014. As a result of filtering out those screenshots, 98 screenshots remain in the training dataset and 189 screenshots in the testing dataset.

Table V. Statistics on Testing Dataset

Condition	#Screenshots	Ratio
Captured from the beginning	69	36.5%
Captured in the middle	111	58.7%
Captured from the end	9	4.8%
Containing a complete title	69	36.5%
Containing a partial title	29	15.3%
Containing at least one image	91	48.1%
Containing at least one video	13	6.9%
Containing ads	48	25.4%
Not containing any body text	12	6.3%

Table VI. Distribution of Relevant Labels

Condition	#Pages	Ratio
Same content but different title	2	0.7%
Same content but different source	52	18.5%
Perfect	227	80.8%
Total	281	100.0%

The screenshots in our testing dataset were taken in different conditions to evaluate how well our approach could deal with a wide range of real situations. As Table V shows, 36.5% of the screenshots were captured from the beginning of articles, 58.7% taken in the middle, and 4.8% from the end. Only 36.5% of them contain a complete title, while 15.3% have a partial title. Since our proposed approach is based on text, some multimedia elements, such as images, video, and ads, are not useful for us; Rather, they could bring additional difficulties. On the one hand, some elements contain text unrelated to the article, from which effective queries cannot be formulated; on the other hand, the more space they occupy, the less useful text we can leverage for query formulation. To test these cases, we take 12 screenshots (6.3%), where images occupy most areas and no body text appears.

To prepare the ground truth for each screenshot in our datasets, we hire annotators to manually search on the Web and identify the best URLs. We use Google in this step to prevent their search/click behavior from influencing the ranking algorithms of Bing, which is used for our automatic approach. They label one of the following five categories to each of their found pages: “perfect,” “same content but different source,” “same content but different title,” “related topic,” and “totally different.” When looking into their labeled pages, we find that there are copies for some articles. As such copies convey the same information as the original ones, we regard pages belonging to one of the first three categories as correct answers in our evaluation.

Another challenge of evaluation is that the ground-truth data may change over time. Some new links are retrieved back in a few days or weeks. Those are not judged before. In addition, some judged links expire and thus we cannot retrieve them back. As a result, the same method may produce slightly different evaluation results if we run it at different time points. It is not fair to compare a method that is run at a time point with another method that is run at another time point. To alleviate this issue, we compare methods at the same time points. Before evaluation, we also ask annotators to judge the returned links that are new to the ground-truth set.

Table VI summarizes the distribution of relevant labels for our testing dataset. In total, 281 pages are judged as relevant, where 2 pages are labeled as “same content but different title,” 52 pages as “same content but different source,” and 227 pages as “perfect.” The minimum, maximum, and mean numbers of relevant pages per screenshot are 1, 5, and 1.5, respectively.

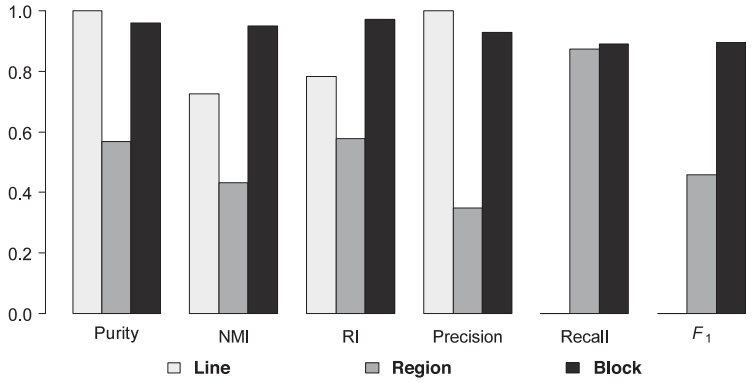


Fig. 5. Comparison of block segmentation methods.

4.2. Block Segmentation

We evaluate the following three methods in terms of segmentation quality:

(1) **Line**, which regards each OCR line as a segment;
(2) **Region**, which regards each OCR region as a segment; and
(3) **Block**, which is our segmentation method proposed in Section 3.3.1.

The segmentation problem can be viewed as a hard clustering problem since OCR lines are grouped into disjoint blocks. We manually group OCR lines for each screenshot in the training dataset and use them as the ground-truth clusters (as we do not need any training in this step). Then, we apply the standard clustering measures [Manning and Schütze 2008], that is, precision, recall, F_1, purity, Rand Index (RI), and Normalized Mutual Information (NMI), into blocks segmented by the three methods.

Results are summarized in Figure 5. As expected, the Line method achieves purity and precision as high as 1.000, because each line is regarded as a block/cluster. There is no wrongly clustered line in each block. The issue is that recall of this method is 0.000 because no pair of lines can be formed in such a block. Thus, F_1 score is also 0.000 for this method. Our proposed Block method outperforms the Line method by 31.0% in NMI and 23.8% in RI.

Although an OCR region considers a part of spatial information of lines, Figure 5 indicates that our Block method largely improves the Region method in all measures. In particular, our method improves precision by 165.6% over the Region method, which indicates that grouping lines with similar width into one segment is far from optimal in semantics. As we show using the examples in Section 3.3, the last line of a title is often wrongly grouped with lines on author or source. The Region method also works much worse than the Line method for all measures except recall and F_1.

The figure indicates that our Block method is effective in grouping semantically related lines. Our method achieves 0.950 in NMI and 0.970 in RI. Recall and Precision for this method are 0.891 and 0.928, respectively, which are also reasonably good. Friedman tests reveal significant effects of the segmentation method on the segmentation quality (e.g., $\chi^2(2) = 158.61,\ p < 0.01$ for NMI; $\chi^2(2) = 154.07,\ p < 0.01$ for RI; $\chi^2(2) = 192.18,\ p < 0.01$ for F_1). Post-hoc tests using Wilcoxon signed rank tests with Holm correction show significant improvement of the Block over the Line methods ($p < 0.01$ for NMI, RI, and F_1) and over the Region methods ($p < 0.01$ for NMI, RI, and F_1). High segmentation quality is important for later procedures because we assume text in one block shall continuously occur in target URLs.

Table VII. Comparison of Attribute Prediction Methods. Significant Improvement from the Heuristic Method at $p < 0.05$ is Shown in Boldface

	Title			Body		
Method	Precision	Recall	F_1	Precision	Recall	F_1
CRF	**0.928**	0.919	**0.868**	**0.967**	**0.880**	**0.893**
Heuristic	0.340	0.912	0.327	0.754	0.780	0.702

4.3. Attribute Prediction

We evaluate the following two methods in terms of attribute prediction performance:

(1) **CRF**, which is our attribute prediction method proposed in Section 3.3.2;
(2) **Heuristic**, which heuristically predicts the attribute of each block. This method regards a block that has the largest font size and contains more than one word as a title. The second condition is added because the site name is often bigger than a title. This method takes into account whether a block contains punctuation and its text is long enough in predicting a body block.

We manually label attributes for each OCR line over both the training and testing datasets. The training dataset is used to train our CRF model and to tune parameters of the Heuristic method. Then, we evaluate the two methods over the testing dataset. We use the standard classification measures [Manning and Schütze 2008], that is, precision, recall, and F_1, for evaluating the prediction performance of each attribute. We calculate these measures at both macro- (evaluating each screenshot and then averaging over a dataset) and microlevels (regarding each OCR line as an evaluation unit and averaging evaluation scores over all lines in a dataset). While we only report the macrolevel results in this article, a similar trend is observed for the microlevel ones.

We can see from Table VII that our CRF method achieves 0.868 for title prediction and 0.893 for body prediction in terms of macro-F_1, which are improvements of 165.4% and 27.2% from the Heuristic method, respectively. The major advantage of the CRF method comes from the improvement of precision. Recall that there exist many screenshots that do not contain a complete title (Table V). Wilcoxon Signed-rank tests show significant improvement of the CRF method over the Heuristic method for all measures (e.g., $Z = 10.16,\ p < 0.01$ for the title attribute and $Z = 8.27,\ p < 0.01$ for the body attribute in terms of F_1) except recall for the title attribute ($Z = 0.31,\ p = 0.74$). These results indicate that leveraging state-of-the-art CRF modeling enables the accurate attribute prediction even for such difficult cases.

4.4. Query Formulation and Result Aggregation

We evaluate the retrieval performance of our query formulation and result aggregation methods. When retrieving search results, we set the parameter K in Section 3.5 to $K = 8$, which is almost the same as the number of search results that conventional Web search engines return in a search engine results page.

We use two evaluation measures for retrieval effectiveness. The primary measure is F_1, which is used for evaluating the top URL retrieved for each screenshot. A score of F_1 is obtained in the following manner: (1) select the top URL from search results that a method retrieves for each screenshot; (2) calculate precision and recall for the selected URLs; and (3) output the harmonic mean of precision and recall. Precision is defined as the ratio of the number of correct output URLs to that of all output URLs. Recall is defined as the ratio of the number of correct output URLs to that of screenshots in the dataset. Note that one F_1 score is obtained for the whole dataset. Thus, no significance test is performed for the results based on this measure. To deal with

Table VIII. Contribution of Screenshot Representation to Retrieval Performance

	Line	Region	Block	Block+Attribute
F_1	0.840	0.810	0.862	0.905
RR@8	0.889	0.863	0.903	0.924
# of queries	11.6	7.0	7.3	7.3

this issue, we also use Reciprocal Rank (RR) as the secondary measure. RR evaluates the ranking retrieved for each screenshot by calculating the inverse rank of the first relevant article. We select RR instead of other ranking-based evaluation measures (e.g., Average Precision and Normalized Discounted Cumulative Gain), because one relevant URL suffices for UniClip to work properly.

As the search part is a bottleneck of the processing time of the search by screenshot task, we are also interested in measuring the efficiency of our approach. For this purpose, we use the average number of issued queries to the search engine as the efficiency measure. The less queries issued, the better an algorithm performed.

4.4.1. Effect of Screenshot Representation. We evaluate how our screenshot representation by block segmentation and attribute prediction contributes to the effectiveness of discovering the best URLs. In this evaluation, we compare the following four methods:

(1) **Line**, which formulates simple queries from each OCR line and aggregates search result lists by the Borda Count;
(2) **Region**, which formulates simple queries from each OCR region and aggregates search result lists by the Borda Count;
(3) **Block**, which formulates simple queries from each block and aggregates search result lists by the Borda Count;
(4) **Block+Attribute**, which formulates simple queries from each block and aggregates search result lists by the Weighted Borda Count.

Results are shown in Table VIII. Compared to the Block method, the Line and Region methods worsen the retrieval performance by 2.6% and 6.4% in terms of the F_1 measure and by 1.6% and 4.6% in terms of the RR@8 measure, respectively. The Block method issues almost the same number of search queries as the Region method, while the Line method issues four more queries on average. These results indicate that merging lines into semantic blocks does help for both effectiveness and efficiency. When we weight search queries with the predicted attributes, the Block+Attribute method further improves the F_1 measure by 5.0% and the RR@8 measure by 2.3% over the Block method, without increasing the number of issued queries. This indicates that block attribute prediction can enhance the retrieval performance. A Friedman test reveals significant effects of the screenshot representation method on the retrieval performance measured with RR@8 ($\chi^2(3) = 11.24,\ p < 0.05$). A Post-hoc test using Wilcoxon signed rank tests with Holm correction shows significant improvement of the Block+Attribute over the Region methods ($p < 0.05$), while we cannot find its significant improvement over the Line method ($p = 0.26$) and the Block method ($p = 0.06$).

4.4.2. Effect of Hybrid Queries and Weighted Aggregation. We evaluate the retrieval performance of our query formulation and result aggregation methods. We compare the Simple and Hybrid methods for query formulation and the Borda Count and Weighted Borda Count for result aggregation, which results in the following four method combinations:

(1) **Simple**, which formulates simple queries from each block and aggregates search result lists by the Borda Count (the same as Block in Section 4.4.1);

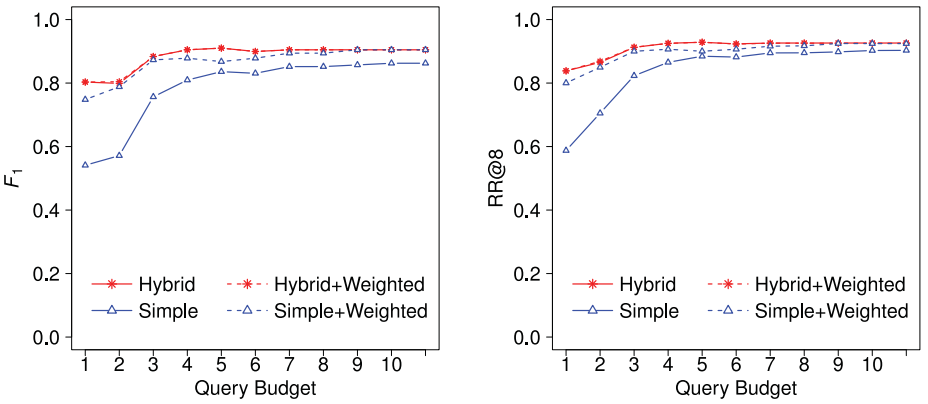


Fig. 6. Comparison of our query formulation and result aggregation methods with different query budgets.

(2) **Simple+Weighted**, which formulates simple queries from each block and aggregates search result lists by the Weighted Borda Count (the same as Block+Attribute in Section 4.4.1);
(3) **Hybrid**, which formulates hybrid queries from each block and aggregates search result lists by the Borda Count;
(4) **Hybrid+Weighted**, which formulates hybrid queries from each block and aggregates search result lists by the Weighted Borda Count.

As each method may submit several search queries and aggregate multiple search result lists, we draw, for each method, a curve of the retrieval performance calculated at 11 points. The first 10 points correspond to the condition when the method submits 1 through 10 search queries and aggregates different numbers of search result lists accordingly. The 11th point corresponds to the condition when the method submits all the generated queries and aggregates the search result lists of those queries.

Results are shown in Figure 6. When all the generated queries are issued, the Hybrid method, which utilizes different query formulation strategies for title, body, and others, dramatically improves the Simple method by 5.0% (from 0.862 to 0.905) in terms of the F_1 measure and by 2.5% (from 0.903 to 0.926) in terms of the RR@8 measure. We can observe from this figure that more improvements are gained when the budget of queries is small. For example, if only one query is issued, the Hybrid method improves the Simple method by 48.4% (from 0.541 to 0.803) in terms of the F_1 measure and by 42.5% (from 0.588 to 0.838) in terms of the RR@8 measure. The improvement rates in F_1 are 39.8% and 16.7% for the budgets of two and three search queries, respectively.

Focusing on result aggregation, we can find that the Simple+Weighted method outperforms the Simple method by 5.0% (from 0.862 to 0.905) in terms of the F_1 measure and by 2.3% (from 0.903 to 0.924) in terms of the RR@8 measure, when all queries are issued. However, compared to the Hybrid method, little further improvement is observed for the Hybrid+Weighted method. A possible explanation is that the Hybrid method has well utilized attribute information in applying different query formulation strategies and in ordering queries from title, body, and others. Thus, the Hybrid has less room to improve compared to the Simple method.

In summary, we find that (1) the Hybrid method obtains better performance than the Simple method and (2) the Weighted Borda Count aggregation improves the retrieval performance, especially when the number of issued queries is limited. When the query budget is one, for example, a Friedman test reveals significant effects of the query formulation and result aggregation methods on the retrieval performance measured

Table IX. Optimal Query Length Parameters and Corresponding Performance on Training Dataset

	$\mathbf{KW_{tfidf}}$	$\mathbf{KW_{BM25}}$	$\mathbf{KW_{TextRank}}$	**Chunk**	**Hybrid+Weighted**
σ_{qmin}	4	4	4	4	4
σ_{qmax}	10	9	11	11	14
F_1	0.612	0.663	0.687	0.837	0.857
RR@8	0.688	0.710	0.744	0.856	0.877
# of queries	7.50	7.08	4.65	9.15	7.84

with RR@8 ($\chi^2(3) = 84.77,\ p < 0.01$). A post-hoc test using Wilcoxon signed rank tests with Holm correction shows significant differences between the Hybrid and the Simple methods ($p < 0.01$), between the Hybrid+Weighted and Simple methods ($p < 0.01$), and between the Simple+Weighted and Simple methods ($p < 0.01$). In what follows, we use the Hybrid+Weighted method as our best approach.

4.4.3. Comparison with Keyword/Chunk Queries. We evaluate our best Hybrid+Weighted method (reported in Section 4.4.2) with baseline query formulation methods. We prepare two kinds of baselines. One formulates queries comprising a set of keywords extracted from the whole text of a screenshot. We consider three variants for keyword extraction: tf-idf [Büttcher et al. 2010], BM25 [Robertson et al. 1994], and TextRank [Mihalcea and Tarau 2004]. We refer to query formulation methods using each of them as $\mathbf{KW_{tfidf}}$, $\mathbf{KW_{BM25}}$, and $\mathbf{KW_{TextRank}}$. The other baseline formulates queries from each block by enclosing chunks, which are syntactic units such as noun phrases and verb phrases, with double quotes. We call this method **Chunk**, which extracts chunks based on the maximum entropy modeling [Koeling 2000]. For the keyword- and chunk-based query formulation methods, we use the Borda Count aggregation to aggregate multiple search result lists, as these baseline methods do not take into account block attributes.

We use the training dataset to tune the query length parameters σ_{qmin} and σ_{qmax} introduced in Section 3.4 for each method. The optimal values for these parameters are decided using the F_1 measure by applying grid searches for predefined parameter spaces, that is, $\sigma_{\text{qmin}} \in [1, 4]$ and $\sigma_{\text{qmax}} \in [5, 20]$. Table IX shows the optimal parameters along with the corresponding F_1 and RR@8 scores and the average number of issued queries for the training dataset. The optimal value for the minimum query length is four, which is common in all methods. This makes sense since very short queries are less likely to return the desired article in their top-K search results. In contrast, the optimal value for the maximum query length varies for different methods. When investigating the curve of F_1 scores with different parameter values, we find that the three keyword-based methods have peak values around 10 and the performance gets worse as the parameter value increases. On the other hand, the performance of the Chunk and Hybrid+Weighted methods, which enclose queries with double quotes, is almost flat for different parameter values. This result suggests the robustness of using quoted queries for search by screenshots. We use these optimal parameter values for subsequent experiments reported in the following.

Results are shown in Table IX. Our Hybrid+Weighted method achieves the highest performance ($F_1 = 0.857$ and RR@8 $= 0.877$), followed by Chunk, $\text{KW}_{\text{TextRank}}$, KW_{BM25}, and KW_{tfidf} in this order. However, it is not fair to evaluate the effectiveness of different methods by comparing the performance scores in that table, since these scores are measured on the training dataset, where we tune parameters and train machine learning models for our approach. Therefore, we apply these methods to the testing dataset and calculate F_1 and RR@8 scores as well. The results for the testing dataset are summarized in Table X.[4] The trend is similar to the training dataset. We can find

Table X. Performance of Different Query Formulation Methods on Testing Dataset

	$\mathbf{KW_{tfidf}}$	$\mathbf{KW_{BM25}}$	$\mathbf{KW_{TextRank}}$	**Chunk**	**Hybrid+Weighted**
F_1	0.613	0.619	0.677	0.841	0.868
RR@8	0.673	0.698	0.749	0.886	0.906
# of queries	5.47	4.70	3.72	7.44	7.15

Table XI. Contribution of OCR to Retrieval Performance. Significant Correlation Coefficients in the First Row and Significant Differences in RR@8 Scores between the Second and Third Rows at $p < 0.05$ are Shown in Boldface

	$\mathbf{KW_{tfidf}}$	$\mathbf{KW_{BM25}}$	$\mathbf{KW_{TextRank}}$	**Chunk**	**Hybrid+Weighted**
Spearman's ρ	**0.341**	**0.289**	**0.291**	**0.213**	−0.013
RR@8 for ill-recognized inputs	0.530	0.565	0.643	0.824	0.893
RR@8 for well-recognized inputs	**0.817**	**0.832**	**0.856**	**0.948**	0.920

from this table that our Hybrid+Weighted method again achieves the highest performance ($F_1 = 0.868$ and RR@8 $= 0.906$) on the testing dataset, followed by Chunk, $\mathrm{KW_{TextRank}}$, $\mathrm{KW_{BM25}}$, and $\mathrm{KW_{tfidf}}$ in this order. All keyword-based query formulation methods greatly underperform our Hybrid+Weighted method. While the Chunk method is almost comparable to ours, the former tends to issue more queries than the latter. A Friedman test reveals significant effects of the query formulation method on the retrieval performance measured with RR@8 ($\chi^2(4) = 108.3,\ p < 0.01$). A post-hoc test using Wilcoxon signed rank tests with Holm correction shows significant improvement of the Hybrid+Weighted method over the $\mathrm{KW_{tfidf}}$, $\mathrm{KW_{BM25}}$, and $\mathrm{KW_{TextRank}}$ methods ($p < 0.05$), while we cannot find its significant improvement over the Chunk method ($p = 0.31$).

4.4.4. Effect of OCR Quality. Search by screenshots are potentially affected by the OCR quality for screenshots. If our OCR engine works perfectly, retrieving the original articles may be easy. If the OCR quality is terrible, on the other hand, any approaches may not work at all. To answer this question, we evaluate how the OCR quality for screenshots influences the retrieval performance of the five methods presented in Section 4.4.3. To this end, we utilize the degree of recognition confidence (see Section 3.3.1 for more details) as an approximation to the OCR quality. We calculate the minimum, maximum, and mean confidence scores for each screenshot using word-level confidence scores in the screenshot. On average, the minimum, maximum, and mean scores of screenshot-level confidence scores in our testing dataset are 0.284, 0.999, and 0.899, respectively. With this measure, we carry out two analyses to investigate the effect of the OCR quality on the retrieval performance.

In the first analysis, we calculate Spearman's rank correlation coefficient between recognition confidence scores and RR@8 scores. A high correlation coefficient indicates that the retrieval performance is highly affected by the OCR quality. Results of this analysis are shown in the first row of Table XI. We can see that the retrieval performance of all baseline methods is highly correlated with the recognition confidence scores, where correlation coefficients are shown to be significant at $p < 0.01$. In contrast, the correlation coefficient for our Hybrid+Weighted method is nearly zero and not significant ($p = 0.863$), which indicates that the retrieval performance of ours is less correlated with the OCR quality.

In the second analysis, we first split the screenshots in our testing dataset into two subsets on the basis of their recognition confidence. One subset contains 95 screenshots whose recognition confidence is not more than the median of recognition confidence,

[4]Note that the scores of the Hybrid+Weighted method reported in Table X are different from those reported in Section 4.4.2. This is because the experiment in Section 4.4.2 and those in Sections 4.4.3 and 4.4.5 are conducted on different days, which may affect evaluation results as described in Section 4.1.

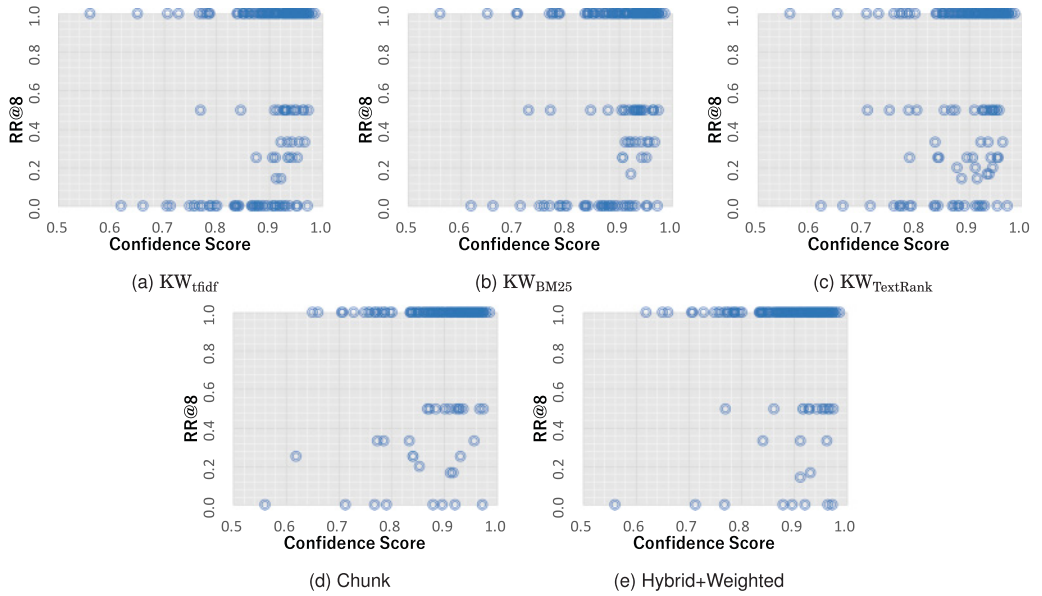


Fig. 7. Scatter plots on relationship between OCR quality and retrieval performance.

while the other contains the remaining ones (94 in total) whose confidence is above the median. The former and latter subset can be regarded as containing ill- and well-recognized screenshots by our OCR engine. We compare how the retrieval performance varies between these two subsets, with the rationale that a retrieval method highly depends on the OCR quality if it achieves quite good performance for the latter subset compared with the former one. Results of the analysis are shown in the second and third rows in Table XI. Large improvements in terms of the RR@8 measure are observed for all baseline methods. Mann-Whitney's U test reveals significant effect of the OCR quality ($Z = 4.37,\ p < 0.01$ for KW_{tfidf}; $Z = 4.16,\ p < 0.01$ for KW_{BM25}; $Z = 3.66,\ p < 0.01$ for $KW_{TextRank}$; and $Z = 2.89,\ p < 0.01$ for Chunk). On the other hand, the improvement by our Hybrid+Weighted method is relatively small (3.02%), which is shown to be insignificant by Mann-Whitney's U test ($Z = 0.33,\ p = 0.70$).

Both analyses lead to the same conclusion. That is, only our Hybrid+Weighted method tends to be less affected by the OCR quality. This is an important property for UniClip to work robustly for various screenshots and OCR engines. The robustness of UniClip is also demonstrated by Figure 7, which shows scatter charts on the relationship between the OCR quality (measured with mean recognition confidence score) and the retrieval performance (measured with RR@8) for each method. These plots indicate that the Hybrid+Weighted method can retrieve correct articles at the top of result ranking in many cases even if the confidence score of an input screenshot falls below 0.800.

4.4.5. Effect of Learning-to-Rank Methods with Only One Query Budget for Each Screenshot. While our Hybrid+Weighted method obtains the best retrieval performance in terms of F_1, it issues more than seven queries on average. To reduce the number of issued queries, we employ learning-to-rank methods proposed in Sections 3.4.3 and 3.5.2 and investigate how the retrieval performance varies if we have only one query budget for each screenshot.

Table XII. Performance of Our Approach With Different Learning-to-Rank Models

Method	F_1	Comparison with (1)	RR@8	Comparison with (1)
(1) Hybrid+Weighted$^{(one)}$	0.782	N/A	0.825	N/A
(2) $KW^{(one)}_{tfidf}$	0.542	−30.69%	0.591	−28.36%
(3) $KW^{(one)}_{BM25}$	0.529	−32.35%	0.578	−29.95%
(4) $KW^{(one)}_{TextRank}$	0.488	−37.60%	0.541	−34.40%
(5) Chunk$^{(one)}$	0.485	−37.98%	0.553	−32.91%
(q1) SVM for regression	0.751	−3.96%	0.792	−3.97%
(q2) MART for classification	0.751	−3.96%	0.806	−2.29%
(q3) MART for regression	0.759	−2.94%	0.807	−2.15%
(q4) SVM for ranking	0.772	−1.28%	0.819	−0.66%
(q5) SVM for classification	0.778	−0.51%	0.817	−0.95%
(q6) LR for classification	0.810	3.58%	0.844	2.32%
(q7) MART for ranking	0.820	4.86%	0.856	3.83%
(r1) MART for regression	0.681	−12.92%	0.762	−7.58%
(r2) MART for ranking	0.707	−9.59%	0.778	−5.67%
(r3) MART for classification	0.723	−7.54%	0.790	−4.24%
(r4) SVM for regression	0.745	−4.73%	0.801	−2.88%
(r5) LR for classification	0.766	−2.05%	0.812	−1.57%
(r6) SVM for ranking	0.777	−0.64%	0.819	−0.68%
(r7) SVM for classification	0.787	0.64%	0.822	−0.28%
(q7) + (r7)	0.831	6.27%	0.858	4.06%

As a comparison to learning-to-rank based models, we evaluate the retrieval performance of the five methods presented in Section 4.4.3, while we also restrict the query budget for these methods to one in this experiment. To avoid ambiguity, we refer to the KW_{tfidf}, KW_{BM25}, $KW_{TextRank}$, Chunk, and Hybrid+Weighted methods with the budget of one query as $\mathbf{KW^{(one)}_{tfidf}}$, $\mathbf{KW^{(one)}_{BM25}}$, $\mathbf{KW^{(one)}_{TextRank}}$, $\mathbf{Chunk^{(one)}}$, and $\mathbf{Hybrid+Weighted^{(one)}}$, respectively. Each method generates the first query in the following manner.

—**Hybrid+Weighted$^{(one)}$:** This method generates queries in the following order. First, it generates simple queries from the title block of a given screenshot. Then, it generates compound queries from pairs of the body blocks. Finally, it generates simple queries from the other blocks. Thus, the first query for this method is (a) a simple query if a given screenshot contains a title block, and (b) a compound query otherwise.

—**$KW^{(one)}_{tfidf}$, $KW^{(one)}_{BM25}$, and $KW^{(one)}_{TextRank}$:** These keyword-based methods first rank the words in a screenshot using tf-idf, BM25, and TextRank. Then, to formulate queries, these methods repeatedly select (without replacement) at most top σ_{qmax} words from the ranking. Thus, the first query for these methods contains keywords that are highly likely to characterize the text content of a screenshot.

—**Chunk$^{(one)}$:** This chunking-based method first converts the words in a screenshot into chunks (i.e., syntactic units such as noun phrases and verb phrases). Then, to formulate queries, this method repeatedly selects (without replacement) chunks containing at most σ_{qmax} words on the basis of the order of appearance. Thus, the first query for this method contains chunks that appear at the top of a screenshot.

The retrieval performance of these methods is shown in the rows numbered (1) through (5) in Table XII. Notice that as well as keyword-based query formulation methods, the Chunk$^{(one)}$ method also greatly underperforms the Hybrid+Weighted$^{(one)}$ method when the query budget is one. A Friedman test reveals significant effects of the query formulation method on the retrieval performance measured with RR@8 ($\chi^2(4) = 57.98,\ p < 0.01$). A post-hoc test using Wilcoxon signed rank tests with Holm

correction shows significant improvement of the Hybrid+Weighted$^{(one)}$ method over the $KW^{(one)}_{tfidf}$, $KW^{(one)}_{BM25}$, $KW^{(one)}_{TextRank}$, and Chunk$^{(one)}$ methods ($p < 0.05$).

In what follows, we report how our learning-to-rank methods can improve the performance over Hybrid+Weighted$^{(one)}$ and can approach that of Hybrid+Weighted. To begin with, we report the effect of each learning-to-rank method (one for query formulation and the other for result aggregation). Then, we report the performance of the combination of these two methods. In the training process of each learning-to-rank method, we conduct 10-fold cross-validation on the training dataset to adjust parameters. Then, we use the whole training dataset to build learning-to-rank models with the adjusted parameters. The performance of trained models is evaluated on the testing dataset. Note that the first query for models based on learning-to-rank queries is the one ranked at the top among all the generated queries.

The rows numbered (q1) through (q7) in Table XII show the retrieval performance of several models based on learning-to-rank queries. We experiment with Logistic Regression (LR) [Hosmer et al. 2013], Support Vector Machine (SVM) [Vapnik 1995], and Multiple Additive Regression Trees (MART) [Friedman 2000]. We rank queries on the basis of their probabilities of belonging to positive (i.e., relevant) class for classification models, while we do so by using their relevance scores returned by regressors for regression models. We can see from these rows that the best method for selecting the most promising query is the MART for ranking ($F_1 = 0.820$ and RR@8 $= 0.856$), which improves the Hybrid+Weighted$^{(one)}$ method by 4.86% in terms of the F_1 measure and by 3.83% in terms of the RR@8 measure.

We then investigate the effect of utilizing learning-to-rank search results. We experiment with the same algorithms used in the query formulation step, that is, LR, SVM, and MART for classification, regression, and ranking. The retrieval performance of the learned models of these algorithms is shown in the rows numbered (r1) through (r7) in Table XII. Only the SVM for classification achieves better performance ($F_1 = 0.787$) than that of Hybrid+Weighted$^{(one)}$. Its relative improvement is just 0.64% in terms of the F_1 measure, which is considerably smaller than the improvement in the query formulation step.

Finally, we report the effect of combining learning-to-rank queries and learning-to-rank search results. On the basis of the previously mentioned two results, we utilize the MART for ranking in the query formulation step and the SVM for classification in the result aggregation step. The row numbered (q7) + (r7) in Table XII shows the retrieval performance of the combined method. Our combined method improves the retrieval performance more than do the individual methods and achieves the best performance ($F_1 = 0.831$ and RR@8 $= 0.858$). It reduces the relative decline from Hybrid+Weighted to 4.26% in terms of the F_1 measure and to 5.30% in terms of the RR@8 measure. The relative improvement from Hybrid+Weighted$^{(one)}$ is 6.27% in terms of the F_1 measure and 4.06% in terms of the RR@8 measure. However, a Wilcoxon signed rank test does not reveal a significant difference between the best learning-to-rank method and the Hybrid+Weighted$^{(one)}$ method ($Z = 1.05,\ p = 0.27$).

In summary, the experimental results show that, by carefully selecting learning-to-rank models for query selection and result aggregation, we can achieve our objective of reducing the number of issued queries with as few drops in the retrieval performance as possible. Note, however, that the improvement by the best learning-to-rank models over our approach without learning is not significant. A possible reason is that the scale of our datasets is too small to train learning-to-rank models. Thus, we shall further investigate the effectiveness of utilizing learning-to-rank techniques for query selection and result aggregation with a reasonable sized data.

5. USER STUDY

We implement UniClip as an Android app[5] and conduct a user study to investigate its usability.

5.1. Participants

We ask 22 participants (11 males and 11 females) to participate in our user study. Since our app is built on Android, we conduct an interview on the participants' experience with Android before the study. We confirm that they have used at least one Android phone and regularly read news, blogs, and articles shared on social networks via their smartphones. The ages of participants range from 19 to 32 (the mean is 24.2). Two of them are high school students, six are graduate students, and the others are undergraduate students or have bachelor degrees. Their educational background is diverse, e.g., medicine, international relations, mechanical engineer, and linguistics. They have used smartphones for 2 to 7 years. The number of smartphones they have ever owned is from one to five (the mean is 2.6).

5.2. Study Flow

We first install our UniClip app to the smartphones of the participants while conducting the previously mentioned interview. Then, we show guidelines on the usage of UniClip to allow the participants to understand how to use our app. After that, the participants are asked to clip articles through the UniClip app and give feedback about the clipping results. When they finish this task, we conduct a SUS test [Brooke 1996] to analyze the usability of our app. At the end of the study, we administer questionnaires to investigate whether the participants prefer UniClip over conventional clipping methods and collect their honest opinions on the UniClip app. The whole user study takes more than 1 hour per participant.

5.3. User Satisfaction

We ask the participants to capture at least 10 screenshots during 10 minutes. After that, they rate the quality of retrieved articles using three criteria: *good*, *okay*, and *bad*. If a result is rated as not good, we further ask why they feel dissatisfied.

On average, the participants clip 12.4 articles in this task, of which 69.5% are rated as good, 17.3% as okay, and 12.8% as bad.

We categorize the dissatisfaction cases by analyzing the reasons reported by participants. The most dominant issue results from search by screenshots issues, which account for 54.4% of the dissatisfaction cases. Except for the algorithm failure on some difficult cases, a major failure cause is that the captured news items are so fresh that Bing has not indexed it yet. This shows a potential limitation of our method based on search for fresh articles, and implies that a retry function that recovers unindexed articles later would be needed to handle such cases.

The failure of main article extraction accounts for 30.4% of the dissatisfaction cases. In some cases, our main article extractor fails to extract images or some paragraphs from the original articles and extracts noisy information like ads. The other issue, which accounts for 15.2%, is related to the User Interface (UI) of our app. For example, some users had hoped that extracted articles could keep the original format, such as font style, while our app uses the same font to all text for consistency between different articles.

[5]https://play.google.com/store/apps/details?id=com.microsoft.snap2pin.

Table XIII. Ratings of Different Method to Clip Articles. The Screenshot Method Receives Significantly Higher Rating than Other Methods at $p < 0.05$

Clipping Method	Rating	Rank 1	Rank 2	Rank 3	Rank 4
ReadingApp	3.57	4	11	6	1
NoteTakingApp	3.23	3	1	6	12
Copy	2.77	1	2	10	9
Screenshot	**4.73**	14	8	0	0

5.4. Usability Analysis

As a result of the SUS test, our UniClip app obtains an SUS score of 89.3 on average (the standard deviation is 9.5). Considering that the SUS score for an average system is 68 [Brooke 2013], our app is evaluated highly by the participants in terms of usability.

One factor related to its usability is whether users know how to take screenshots on smartphones. If only a few users know that, most users face a barrier when starting to use UniClip. To understand this aspect, we ask participants whether they know how to take screenshots on their smartphones before the user study. As a result, 72.7% of the participants answer yes to that question. A possible reason for the high ratio is because our participants are relatively young. However, it would be worth noting that the others quickly learn how to take screenshots just after one or two trials.

5.5. Comparison to Other Clipping Methods

In the posterior questionnaires, we ask the participants to evaluate how they prefer the following four clipping methods, with both a five-point Likert scale and a ranking:

(1) **ReadingApp:** Save articles as favorites in reading apps;
(2) **NoteTakingApp:** Share articles to a note-taking app;
(3) **Copy:** Copy and paste the links of articles;
(4) **Screenshot:** Take screenshots of articles.

As shown in Table XIII, the Screenshot method obtains the highest rating score of 4.73 (the standard deviation is 0.46), which outperforms the rating scores of the other conventional methods. A Friedman test reveals a significant effect of the screenshot method on the rating score ($\chi^2(3) = 26.57,\ p < 0.01$). A post-hoc test using Wilcoxon signed rank tests with Holm correction shows significant improvement of the Screenshot method over the other methods ($p < 0.01$). This table also indicates that all the participants rate the rank of the Screenshot method as the first or second. In fact, 21 out of 22 participants answer in our questionnaires that they like the UniClip app. Note, however, that study participants try out only the UniClip app from among the four clipping methods. Thus, their mere familiarity with that app may possibly affect their answers.

5.6. Overall Comments

In the questionnaires, we also ask the participants to write down pros and cons of the UniClip app. They highly value its (1) *effectiveness*, for example, "The app copies the whole article even though I only take a screenshot of half of it," (2) *usability*, for example, "Very convenient to use, very simple app," and (3) *portability*, for example, "The app is very useful as we can read the articles later if we do not have time at particular moment. We can use it offline, too."

While most comments are positive, some participants give negative feedback. Their comments can be summarized as (1) *not found issue*, for example, "Few articles do not get extracted," and (2) *extraction issue*, for example, "sometimes the pictures are missing," which are consistent with the reasons why some participants feel dissatisfied in our user study.

Considering these issues, we shall further improve the accuracy of our approach for search by screenshots. We shall try to manage the failed screenshots in a better way so that users can easily know which articles are successfully recovered. Although the focus of this article is solving the search by screenshots problem, the quality of our main article extractor would have a big impact on users' offline reading. It would be important to improve this module for further development of the UniClip service.

6. CONCLUSIONS AND FUTURE WORK

In this article, we have addressed the search by screenshots task, which takes as input a screenshot of a partial article and outputs the full article by leveraging Web search. We focus specifically on article screenshots taken by mobile devices, with the goal of allowing users to clip articles from diverse mobile apps just by taking their screenshots. Our approach to the search by screenshots task (1) segments a screenshot into blocks using structural features and applies CRF to predict the attribute of each block, (2) formulates effective search queries in a hybrid manner, depending on the attributes of blocks, and (3) aggregates the search result lists of multiple queries using Weighted Borda Count. To improve retrieval efficiency, we also extend our approach so that we can find the desired article with only one query. Our extended approach utilizes learning-to-rank techniques to select the most promising query from all the generated queries and to find the most likely URL from the single search result list of the selected query. Experimental results reveal that our approach can discover whole articles from their partial images with $F_1 = 0.868$ by issuing all the generated queries and with $F_1 = 0.831$ by issuing only one query. Our approach considerably outperforms the baseline methods based on keyword extraction and chunking methods. We also find that integrating the MART and SVM models into query formulation and result aggregation improves our approach without learning by about 6%. The user study we conduct shows the usability of the UniClip app where our approach is implemented. Study participants are satisfied with the quality of results retrieved by the UniClip app for 69.5% of the articles they clip. The UniClip app is preferred by 21 out of 22 participants for its simplicity and effectiveness.

One limitation of our work is that we use the small-scale datasets in our experiments due to the large efforts expended for collecting diverse screenshots using many mobile apps and for preparing ground-truth URLs for those screenshots. This procedure is necessary for this work to evaluate the robustness of UniClip against different screenshot contents and mobile app interfaces. Nevertheless, the generalizability of our conclusions in this article should be tested with larger datasets in the future. One workaround for this issue is to focus only on screenshots from mobile web browsers, because the ground-truth URLs can be obtained easier than from other apps. To confirm the effectiveness of our proposed features for query selection, we also plan to add feature-based keyword extraction methods for long queries (e.g., Bendersky and Croft [2008] and Xue et al. [2010]) as the baseline of query formulation in future experiments. Future work also includes extending our framework to deal with various kinds of images. In this article, we assume that an input screenshot contains a part of a single online article written in English. However, users may capture other kinds of screenshots, like maps or error messages, in practical situations. Thus, we would need to identify whether a given screenshot should be applied into our search by screenshots framework. As a next step, we plan to solve the problem by using image processing technologies or matching between text of a screenshot with the discovered Web document. In addition to English, we also plan to expand our approach to other languages. We are also interested in search by a photo with embedded text. This allows a user to take a photo of a page from a magazine in order to find the full article from the Web.

Another interesting extension in the future is to investigate whether it is useful and how to find strongly related articles rather than identical ones.

REFERENCES

Mikhail Ageev, Qi Guo, Dmitry Lagun, and Eugene Agichtein. 2011. Find it if you can: A game for modeling different types of web search success using interaction data. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'11)*. ACM, New York, 345–354. DOI: http://dx.doi.org/10.1145/2009916.2009965

Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. 2009. Diversifying search results. In *Proceedings of the 2nd ACM International Conference on Web Search and Data Mining (WSDM'09)*. ACM, New York, 5–14. DOI: http://dx.doi.org/10.1145/1498759.1498766

Alexandr Andoni and Piotr Indyk. 2008. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM* 51, 1 (Jan. 2008), 117–122. DOI: http://dx.doi.org/10.1145/1327452.1327494

Niranjan Balasubramanian, Giridhar Kumaran, and Vitor R. Carvalho. 2010a. Exploring reductions for long web queries. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'10)*. ACM, New York, 571–578. DOI: http://dx.doi.org/10.1145/1835449.1835545

Niranjan Balasubramanian, Giridhar Kumaran, and Vitor R. Carvalho. 2010b. Predicting query performance on the web. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'10)*. ACM, New York, 785–786. DOI: http://dx.doi.org/10.1145/1835449.1835615

Michael Bendersky and W. Bruce Croft. 2008. Discovering key concepts in verbose queries. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'08)*. ACM, New York, 491–498. DOI: http://dx.doi.org/10.1145/1390334.1390419

Jean-Charles de Borda. 1781. Mémoire sur les élections au scrution. *Mémoires de l'Académie Royale des Sciences* (1781), 657–665.

John Brooke. 1996. SUS: A quick and dirty usability scale. In *Usability Evaluation In Industry*, Patrick W. Jordan, Bruce Thomas, Ian Lyall McClelland, and Bernard Weerdmeester (Eds.). Taylor & Francis, London, UK, 189–194.

John Brooke. 2013. SUS: A retrospective. *Journal of Usability Studies* 8, 2 (2013), 29–40.

Stefan Büttcher, Charles L. A. Clarke, and Gordon V. Cormack. 2010. *Information Retrieval: Implementing and Evaluating Search Engines*. MIT Press, Cambridge, MA.

Moses S. Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC'02)*. ACM, New York, 380–388. DOI: http://.dx.doi.org/10.1145/509907509965

Junghoo Cho, Narayanan Shivakumar, and Hector Garcia-Molina. 2000. Finding replicated web collections. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD'00)*. ACM, New York, 355–366. DOI: http://dx.doi.org/10.1145/342009.335429

Abdur Chowdhury, Ophir Frieder, David Grossman, and Mary Catherine McCabe. 2002. Collection statistics for fast duplicate document detection. *ACM Transactions on Information Systems* 20, 2 (April 2002), 171–191. DOI: http://dx.doi.org/10.1145/506309.506311

Kenneth Ward Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational Linguistics* 16, 1 (March 1990), 22–29. http://dl.acm.org/citation.cfm?id=89086.89095

Steve Cronen-Townsend, Yun Zhou, and W. Bruce Croft. 2002. Predicting query performance. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'02)*. ACM, New York, 299–306. DOI: http://dx.doi.org/10.1145/564376.564429

Na Dai, Milad Shokouhi, and Brian D. Davison. 2011. Learning to rank for freshness and relevance. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'11)*. ACM, New York, 95–104. DOI: http://dx.doi.org/10.1145/2009916.2009933

Fan Deng and Davood Rafiei. 2006. Approximately detecting duplicates for streaming data using stable bloom filters. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD'06)*. ACM, New York, 25–36. DOI: http://dx.doi.org/10.1145/1142473.1142477

Xin Fan, Xing Xie, Zhiwei Li, Mingjing Li, and Wei-Ying Ma. 2005. Photo-to-search: Using multimodal queries to search the web from mobile devices. In *Proceedings of the 7th ACM SIGMM International Workshop on Multimedia Information Retrieval (MIR'05)*. ACM, New York, 143–150. DOI: http://dx.doi.org/10.1145/1101826.1101851

Katerina T. Frantzi. 1997. Incorporating context information for the extraction of terms. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL'97)*. Association for Computational Linguistics, Stroudsburg, PA, 501–503. DOI:http://dx.doi.org/10.3115/976909.979682

Jerome H. Friedman. 2000. Greedy function approximation: A gradient boosting machine. *Annals of Statistics* 29 (2000), 1189–1232.

Ben He and Iadh Ounis. 2006. Query performance prediction. *Information Systems* 31, 7 (Nov. 2006), 585–594. DOI:http://dx.doi.org/10.1016/j.is.2005.11.003

Monika Henzinger. 2006. Finding near-duplicate web pages: A large-scale evaluation of algorithms. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'06)*. ACM, New York, 284–291. DOI:http://dx.doi.org/10.1145/1148170.1148222

David W. Hosmer, Stanley Lemeshow, and Rodney X. Sturdivant. 2013. *Applied Logistic Regression*. Wiley, New York.

Qiang Huo and Zhi-Dan Feng. 2003. Improving Chinese/English OCR performance by using MCE-based character-pair modeling and negative training. In *Proceedings of the 7th International Conference on Document Analysis and Recognition (ICDAR'03)*. IEEE, 364–368. DOI:http://dx.doi.org/10.1109/ICDAR.2003.1227690

Rob Koeling. 2000. Chunking with maximum entropy models. In *Proceedings of the 2nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning - Volume 7 (ConLL'00)*. Association for Computational Linguistics, Stroudsburg, PA, 139–141. DOI:http://dx.doi.org/10.3115/1117601.1117634

Giridhar Kumaran and James Allan. 2007. A case for shorter queries, and helping users create them. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics Human Language Technologies (NAACL-HCT'07)*. Association for Computational Linguistics, 220–227.

Giridhar Kumaran and Vitor R. Carvalho. 2009. Reducing long queries using query quality predictors. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'09)*. ACM, New York, 564–571. DOI:http://dx.doi.org/10.1145/1571941.1572038

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning (ICML'01)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 282–289. http://dl.acm.org/citation.cfm?id=645530.655813

Tie-Yan Liu. 2009. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval* 3, 3 (March 2009), 225–331. DOI:http://dx.doi.org/10.1561/1500000016

Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. 2007. Detecting near-duplicates for web crawling. In *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*. ACM, New York, 141–150. DOI:http://dx.doi.org/10.1145/1242572.1242592

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.

Christopher D. Manning and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

Olena Medelyan and Ian H. Witten. 2006. Thesaurus based automatic keyphrase indexing. In *Proceedings of the 6th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'06)*. ACM, New York, 296–297. DOI:http://dx.doi.org/10.1145/1141753.1141819

Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing order into texts. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP'04)*, Dekang Lin and Dekai Wu (Eds.). Association for Computational Linguistics, Barcelona, Spain, 404–411.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66. Stanford InfoLab. Retrieved from http://ilpubs.stanford.edu:8090/422/ Previous number = SIDL-WP-1999-0120.

Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at TREC-3. In *Proceedings of the 3rd Text Retrieval Conference (TREC-3)*. Gaithersburg, 109–126.

Arnold W. M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh Jain. 2000. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 12 (Dec. 2000), 1349–1380. DOI:http://dx.doi.org/10.1109/34.895972

Takashi Tomokiyo and Matthew Hurst. 2003. A language model approach to keyphrase extraction. In *Proceedings of the ACL 2003 Workshop on Multiword Expressions: Analysis, Acquisition and Treatment - Volume 18 (MWE'03)*. Association for Computational Linguistics, Stroudsburg, PA, 33–40. DOI:http://dx.doi.org/10.3115/1119282.1119287

Peter D. Turney. 2000. Learning algorithms for keyphrase extraction. *Information Retrieval* 2, 4 (May 2000), 303–336. DOI: http://dx.doi.org/10.1023/A:1009976227802

Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer.

Linkai Weng, Zhiwei Li, Rui Cai, Yaoxue Zhang, Yuezhi Zhou, Laurence T. Yang, and Lei Zhang. 2011. Query by document via a decomposition-based two-level retrieval approach. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'11)*. ACM, New York, 505–514. DOI: http://dx.doi.org/10.1145/2009916.2009985

Xing Xie, Lie Lu, Menglei Jia, Hua Li, Frank Seide, and Wei-Ying Ma. 2008. Mobile search with multimodal queries. *Proceedings of the IEEE* 96, 4 (April 2008), 589–601. DOI: http://dx.doi.org/10.1109/JPROC.2008.916351

Xiaobing Xue, Samuel Huston, and W. Bruce Croft. 2010. Improving verbose queries using subset distribution. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM'10)*. ACM, New York, 1059–1068. DOI: http://dx.doi.org/10.1145/1871437.1871572

Yin Yang, Nilesh Bansal, Wisam Dakka, Panagiotis Ipeirotis, Nick Koudas, and Dimitris Papadias. 2009. Query by document. In *Proceedings of the 2nd ACM International Conference on Web Search and Data Mining (WSDM'09)*. ACM, New York, 34–43. DOI: http://dx.doi.org/10.1145/1498759.1498806

Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. 2009. Sikuli: Using GUI screenshots for search and automation. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology (UIST'09)*. ACM, New York, 183–192. DOI: http://dx.doi.org/10.1145/1622176.1622213

Tom Yeh, Konrad Tollmar, and Trevor Darrell. 2004. Searching the web with mobile images for location recognition. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'04)*. IEEE Computer Society, Washington, DC, 76–81. http://dl.acm.org/citation.cfm?id=1896300.1896312

Shuyi Zheng, Ruihua Song, and Ji-Rong Wen. 2007. Template-independent news extraction based on visual consistency. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2 (AAAI'07)*. AAAI Press, 1507–1512. http://dl.acm.org/citation.cfm?id=1619797.1619887

Received June 2016; revised October 2016; accepted December 2016