CoreEx: Content Extraction from Online News Articles

Jyotika Prasad
Stanford University
Stanford, CA-94305
jyotika@stanford.edu

Andreas Paepcke
Stanford University
Stanford, CA-94305
paepcke@cs.stanford.edu

ABSTRACT

We developed and tested a heuristic technique for extracting the main article from news site Web pages. We construct the DOM tree of the page and score every node based on the amount of text and the number of links it contains. The method is site-independent and does not use any language-based features. We tested our algorithm on a set of 1120 news article pages from 27 domains. This dataset was also used elsewhere to test the performance of another, state-of-the-art baseline system. Our algorithm achieved over 97% precision and 98% recall, and an average processing speed of under 15ms per page. This precision/recall performance is slightly below the baseline system, but our approach requires significantly less computational work.

1. INTRODUCTION

Automatic core content extraction from Web pages is a challenging yet significant problem in the fields of Information Retrieval and Data Mining. The problem arises particularly on the World-Wide Web, because Web pages are often decorated with side bars, branding banners and advertisements. Many important automated and manual text analysis tasks are handicapped by this mixture of core content and only peripherally related information on a single page.

For example, news summarization algorithms work better when their input is focused. Similarly, the transcoding of Web pages for delivery to small PDA and cell phone screens requires fast and effective extraction of core content from the original pages.

Figures 1 and 2 show examples of the problem. The parts marked *Extracted Text* are the body of the respective news content.

We encountered the problem of needing to identify core content even during processes that involve human inspection. In our work with political scientists our partners attempted to code (tag) news Web pages for content differences that were too subtle for machine learning algorithms

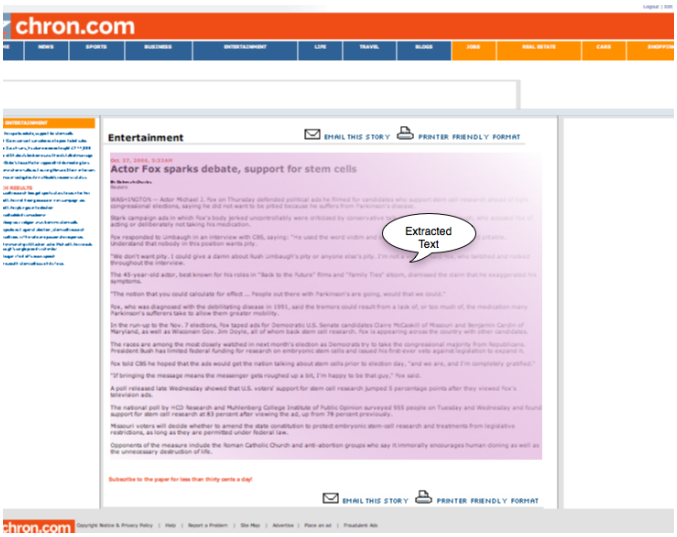

Figure 1: The shaded area is the text extracted by the algorithm

to succeed. The analysis was to touch a large set of pages in a mixed-site Web news archive.

The strain of this work was significantly increased, and its duration extended, by the irregularity of Web page layout. For each page, the human coders needed to locate the actual article of interest. Automated core content identification would have helped speed the coding process.

Yet it is precisely the large variations in page structure across domains that make automatic extraction of core content difficult. Previous approaches have therefore constructed site specific wrappers, or deployed machine learning. The wrappers are small programs that are tailored to the pages of a Web site, and which extract the core content through their explicit knowledge of page structure.

Unfortunately, even within a single domain, providers often change the structure of their pages over time. Approaches that rely on site specific wrappers therefore need (i) the ability to detect the occurrence of structural changes that will impact the respective wrapper's performance, and (ii) will need to be manually updated periodically to reflect these changes.

In contrast to wrapper approaches, the machine learning techniques can be used across sites, but again need to be re-trained as sites evolve.

A third family of approaches applies Natural Language

Figure 2: The shaded area is the text extracted by the algorithm

Processing (NLP) techniques. These algorithms automatically extract named entities and parts of speech, thereby identifying page fragments that are most likely to be continuous, body prose.

The disadvantage of NLP approaches for large datasets is that the algorithms are often expensive. Their complexity thereby stands in the way of large scale analysis over sizable archives of pages.

We developed CoreEx, a simple heuristic technique that extracts the main article from online news Web pages. CoreEx uses a Document Object Model (DOM) tree representation of each page, where every node in the tree represents an HTML node in the page. We analyze the amount of text and number of links in every node, and use a heuristic measure to determine the node (or a set of nodes) most likely to contain the main content.

In the following section we describe our algorithm, beginning with a most basic version. From that version we extend the approach and show how these extensions improve extraction results. Section 3 describes our validation experiment. Experimental results will be presented in Section 5, and discussed in Section 6. We then present a related work section and conclude with a summary and opportunities for future work.

2. ALGORITHM

CoreEx is motivated by the observation that the main content in a news article page is contained in a DOM node, or a set of DOM nodes, with significantly more text than links. The central idea is to compute the text-to-link ratio of every node in the Web page's DOM tree. A score is then computed after additional, simple heuristics are combined with that ratio. The algorithm then selects the node with the highest score.

The very simple example in Figure 3 illustrates the relationship between a Web page's HTML code, that code's manifestation in a browser window, and the corresponding DOM tree.

The Figure illustrates that each HTML tag is represented in the DOM tree as an interior node (the diamonds). We also observe two types of terminal nodes, square and oval. The square nodes correspond to text that is not linked. The—in this example sole—oval terminal node represents linked content. In this case the linked content is text. The node might instead be a linked image. We call the squares *text nodes*, and the ovals *link nodes*. Note that not all terminal nodes are text or link nodes. One might encounter, for example, an unlinked image that is inline with text.

We parse the Web page using the Java Library HTMLParser to obtain its DOM tree. We then recursively score every node in the DOM tree. The details are described below.

2.1 Basic Algorithm

For every node in the DOM tree, we maintain two counts. $textCnt$ holds the number of words contained in the node. For interior nodes this count includes the sum of the words in the subtree to which the node is the root.

Similarly, $linkCnt$ holds the number of links in or below the node. $textCnt$ of a node is thus recursively defined as the sum of the $textCnt$ of its children. $linkCnt$ follows a similar definition. A node's score is a function of its $textCnt$ and $linkCnt$. To keep the score normalized between 0 and 1, a link is counted as one word of text.

Once the DOM tree of a page is obtained, we walk up the tree, taking the following steps.

- For a terminal node T :
  - If T is a text node
    $textCnt(T) =$ word count of text in T
    $linkCnt(T) = 0$.
  - If T is a link node
    $textCnt(T) = 1$
    $linkCnt(T) = 1$.
  - In all other cases
    $textCnt(T) = 0$
    $linkCnt(T) = 0$.
- For a non-terminal node N:
  - Initialize $textCnt(N) = 0$ and
    $linkCnt(N) = 0$
  - Iterate through all its children. For every child

$$textCnt(N) = textCnt(N) + textCnt(child)$$

$$linkCnt(N) = linkCnt(N) + linkCnt(child)$$

Note above from the procedure for link node terminals that $textCnt$ is the sum of non-linked words, and the number of links. The $linkCnt$ is the sum of links.

If two nodes reach identical scores, the node higher in the DOM tree is selected. In case of a tie among siblings, a random choice is made among the two.

Figure 4 shows a replica of the DOM tree in Figure 3.

The word and link counts are appended to each node in the Figure. Reading from bottom to top in the left-most tree we see that the text *with a link* only receives a one for word counts and a one for link count as this text is a link anchor. In contrast, one level up, the text *A paragraph* receives a two in the word count, and a zero in the link count, because the text is not contained in an HTML `<a>` tag. Traveling

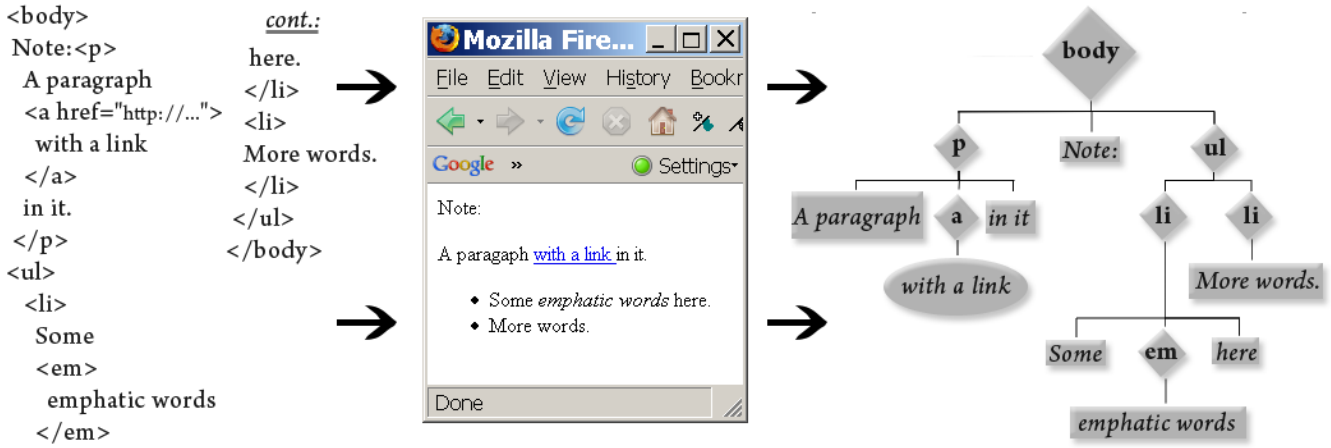

Figure 3: Example of HTML code, its screen manifestation, and its DOM tree.

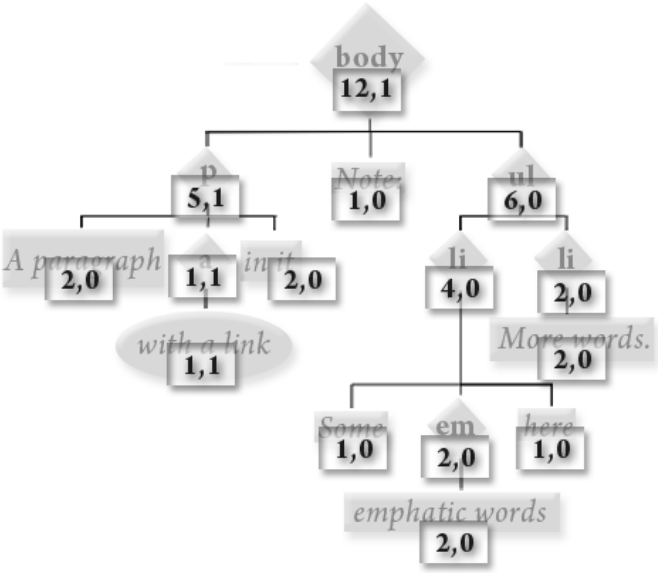

Figure 4: DOM tree annotated with the word and link counts.

upwards, the two counters accumulate, until the top level `<body>` tag is reached.

Once the text and link counts are known for each node, the nodes are scored. The DOM node with the highest score is picked as the *Main Content Node*. The DOM subtree rooted at this node contains the content. The question then is how to compute a score that leads to good choices for core content.

2.1.1 Scoring

Basic Scoring Function. The basic scoring function measures the ratio of the amount of text contained in the node to the number of links in the node. For a node N, the basic score is

$$\frac{textCnt - linkCnt}{textCnt}$$

Observe that in this scoring scheme a node with only links has score 0, while a node with only text and no links has score 1.

This basic algorithm has two significant drawbacks. First, it favors small nodes with no links over larger nodes with a few links. This behavior is undesirable since the latter is more likely to contain the main content. Consider the example in Figure 5. Basic scoring would pick one of the smaller nodes, which have a score of 1.0 instead of the entire `<div>` node, since the score of the `<div>` node is reduced by the single link.

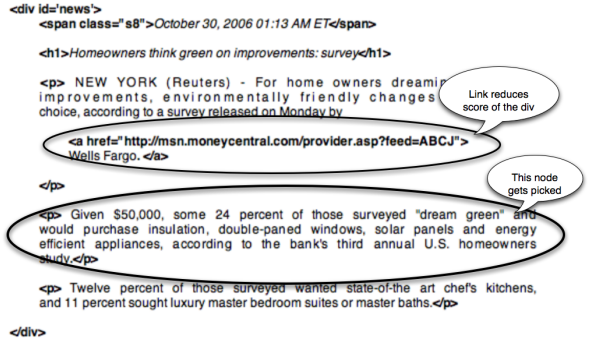

Figure 5: An example where basic scoring fails.

Second, the algorithm fails when the main content is contained in a *set* of nodes, instead of being held under one node. We resolve these issues by adding two modifications to the basic algorithm.

2.2 Modifications to Basic Algorithm

2.2.1 Weighted Scoring Function

The first modification adds another component to the scoring function, which captures the fraction of the total

text in the page that is contained in the node being scored. As usual, the 'total text' is measured in words.

This component biases the score in favor of nodes which contain more text. If $page_{text}$ is the total text the webpage contains and $textCnt$ and $linkCnt$ are defined as before, the new scoring function becomes

$$weight_{ratio} \times \frac{textCnt - linkCnt}{textCnt} + weight_{text} \times \frac{textCnt}{page_{text}}$$

$weight_{ratio}$ and $weight_{text}$ are weights assigned to both the components of the scoring function. Both the weights lie between 0 and 1, and have to sum to 1 (to keep the score between 0 and 1). $weight_{text}$ is much lower than $weight_{ratio}$, since we only require a gentle push towards the node that captures more text.

Figure 6 shows our running example DOM tree with scores labeling the nodes. Each label contains two numbers. The first is the basic score, the second is the weighted score.

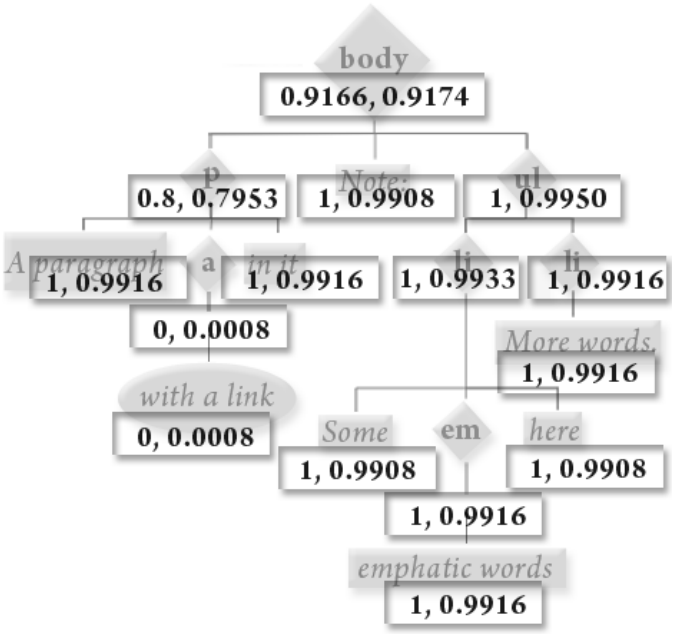

Figure 6: DOM node labeled with both basic and weighted scores. The weighted score scheme selects the `ul` tag node as the most promising subtree root.

In this case the *Note:* node and the *ul* nodes seem equivalent under the basic scheme, but correctly diverge once weighting is added. The `ul` tag is in fact recognized correctly as the best choice of root for the content subtree.

For an empirical determination of the weight constants we ran a set of experiments with different values of $weight_{ratio}$ and $weight_{text}$ on 500 news articles that were chosen at random from our test data (described in Section 3). The best performance was found to be $weight_{ratio} = 0.99$ and $weight_{text} = 0.01$. In all the subsequent results in Section 5, we use these values for scoring.

2.2.2 Subset selection

Even with the weighted scoring, the algorithm can fail in cases where the content is distributed across several nodes, and is not contained entirely under one node. Consider the structure in Figure 7. Selecting the `<div>` node would include the advertisements. So the algorithm would either select the `<div>` node along with the ads, or pick one of the text nodes. Neither of these cases gives us the desired result.

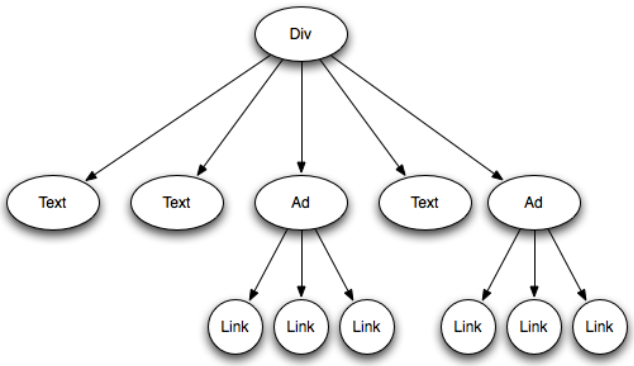

Figure 7: Example structure for subset selection

We overcome this issue by only selecting a subset of each node's children, instead of all its children as described so far. For every node, we now store a set of nodes S and keep new counts, $setTextCnt$ and $setLinkCnt$, along with the earlier $textCnt$ and $linkCnt$. We iterate through its children and add the children whose text-to-link ratio is above a certain threshold to S. $setTextCnt$ and $setLinkCnt$ are the sum of the $textCnt$ and $linkCnt$ of the nodes added to S.

More formally, the earlier algorithm is modified for non-terminal nodes as follows:

For a non-terminal node N:

- Initialize
  - $textCnt(N) = 0$ and $linkCnt(N) = 0$
  - set S(N) as an empty set.
  - $setTextCnt(N) = 0$ and $setLinkCnt(N) = 0$
- For every child of N,

  $$textCnt(N) = textCnt(N) + textCnt(child)$$

  $$linkCnt(N) = linkCnt(N) + linkCnt(child)$$

  Calculate

  $$childRatio = \frac{textCnt - linkCnt}{textCnt}$$

  If $childRatio > Threshold$
  - add the child to S(N).

    $$setTextCnt(N) = setTextCnt(N) + textCnt(child)$$

    $$setLinkCnt(N) = setLinkCnt(N) + linkCnt(child)$$

  Store S(N), $textCnt(N)$, $linkCnt(N)$, $setTextCnt(N)$ and $setLinkCnt(N)$

The nodes are now scored based on their $setTextCnt$ and $setLinkCnt$. For the node with the highest score, return the corresponding set S as the set of nodes that contains the content.

As with the $weight_{ratio}$ and $weight_{text}$, we empirically determined the threshold to be 0.9.

3. EXPERIMENT

We tested CoreEx's ability to extract the core contents from news Web pages. The DOM nodes from a gold set were compared with the nodes that CoreEx extracted automatically. We examined how many nodes the algorithm missed, how many nodes it erroneously selected as core content, and how fast the implementation was able to process pages.

3.1 Test data

Our test data consisted of 1620 news article pages from 27 different sources. After using 500 pages for the one-time determination of our weight and threshold constants, we were left with 1120 pages for experimentation. Of the original 1620 pages, the core content was unique for 338 pages. Others were core content duplicates. Such duplication is extremely common in Web news archives, because many news outlets draw from the same providers, such as the Associated Press. However, when duplicates originate from different sources, the decoration of the core content varies significantly. The Web pages are thus structured very differently, and the content that surrounds the core content tends to be entirely different [13].

This dataset was created by the MITRE corporation for testing their work on Adaptive Web-page Content Identification [2]. The MITRE team labeled components of each page, employing a combination of site-specific, manually created wrappers, and manual labeling. The pages were labeled for *Title, SubTitle, Author, Body, Location, Source,* and *Copyright.*

The MITRE dataset contains two copies of each page. The *raw* copy is the original page, except that all embedded scripts have been wrapped in *noscript* tags. The *labeled* page is a copy of the page where the nodes containing the article have been labeled as described above. We use the nodes marked as *Content* in the *labeled* pages as our gold standard answer, and compare the nodes extracted by our algorithm against those answers.

3.2 Preprocessing Variants

Many algorithms benefit from page preprocessing. However, these preparatory procedures can be expensive. One therefore attempts to keep them to a minimum.

In order to test which type of preprocessing would be helpful for CoreEx we repeated all our experiments for several variants of page preprocessing.

1. **Ignoring Form tags.** Nodes with the following HTML tags: *Select, Form, Input, Textarea,* and *Option* are ignored while processing the DOM tree. This step is motivated by the observation that some pages contain text within these tags that is mostly not content. Due to a high text-to-link ratio, these nodes tend to confuse the algorithm.

2. **Removing scripts.** All nodes containing scripts are removed using a regular expression processor, before the DOM tree is constructed.

3. **Converting to XHTML.** The page is converted to XHTML using TagSoup. The intuition here was that a cleaner DOM tree might yield better results. This is also one of the preprocessing steps taken in [2], and we wanted to test its effect on CoreEx.

In the following section we report the additive effect of these variants on overall performance.

3.3 Performance Measures

We use the term *core contents*, or just *contents* to mean all the fragments of a Web page that ought to be extracted by an ideal algorithm. These include the title, the article's date, the byline, the copyright information, the main body and so on. We use *body* to mean the central portion of the article.

We measured three performance dimensions:

1. **Node-based Precision-Recall.** These are straightforward precision and recall measures, based on the number of nodes correctly labeled.

2. **Text-based Precision-Recall.** The above measure does not reflect an accurate picture of the performance, since what we are really concerned with is the text of the news articles. Missing one node which contains a significant portion of the text matters more than missing two nodes with little text in each. We therefore modify the above measure to capture the amount of text extracted by weighting each node by the amount of text it contains, and using these weights in the recall and precision calculation.

   More formally, let the text in node n_i have a word count of w_i. We weight each node by w_i, so that missing a content node or including a non-content node in the result incurs a penalty of w_i. If $CorrectlyExtracted$ is the set of all nodes that the algorithm labeled correctly, $Extracted$ is the set of nodes extracted by our algorithm, and $Answer$ the set of nodes marked as the correct answer in the gold set, then the precision is

   $$\frac{\sum_{n_i \in CorrectlyExtracted} w_i}{\sum_{n_i \in Extracted} w_i}$$

   while the recall is

   $$\frac{\sum_{n_i \in CorrectlyExtracted} w_i}{\sum_{n_i \in Answer} w_i}$$

3. Processing Time.Precision and recall are, of course, not the only important performance measures. The time required to process each page must be considered as well if core contents of large Web archives are to be extracted.

4. EXPERIMENT SETUP

The tests were run on a workstation with four AMD Opteron 1.8GHz dual core processors and 32 GB main memory . The CoreEx implementation was written in Java 1.5, and the system is single threaded.

5. RESULTS

5.1 Performance Overview

Table 1 shows the recall and precision results for different configurations of our algorithm when run on all 1120 test pages. The Table reflects our testing procedure of adding the described refinements one by one. We have not tested all possible combinations of the variants.

Figure 8 provides the time taken by each of our variants to process the 1120 pages. Most of the configurations are quite fast, consuming under 15 ms per page. Converting to XHTML, on the other hand, causes a significant increase in processing time.

The recall/precision numbers are computed following the text-based approach as described in Section 3.3. The numbers refer to extraction results for *core content*, that is all the text fragments in the page that a human being would consider part of the article, not merely the article body.

Our best system (weighted scoring, subset selection, ignoring forms, and removing scripts) performs remarkably well, with 97% precision and 98% recall. The node-based precision and recall for this configuration are 93.0% and 92.5% respectively. These numbers are included parenthetically in the optimal row of Table 1. We observe that the text-based recall and precision are significantly higher than the node-based recall and precision, implying that the nodes CoreEx labeled incorrectly do not contain much text.

Table 1: *Text-Based* Recall/Precision Performance

	Configuration	Precision	Recall	F_1
R_1	basic scoring	0.8387	0.0271	.053
R_2	weighted scoring	0.9562	0.9088	.932
R_3	weighted scoring, subset selection	0.9776	0.9183	.947
R_4	weighted scoring, subset selection, ignoring forms	0.9783	0.9541	.966
R_5	weighted scoring, subset selection, ignoring forms, removing scripts	0.9715 *(93.04)*	0.9862 *(92.52)*	.979
R_6	weighted scoring, subset selection, ignoring forms, removing scripts, converting to XHTML	0.9685	0.9868	.978

We summarize results R_i below. Section 6 will discuss how some of these measurements arise.

R_1:Using the basic algorithm, precision is around 85%, but recall is only around 2%. This poor result occurs because, as pointed out earlier, using only the text-to-link ratio favours small nodes with no links, and very often only one paragraph from the article is extracted. This behavior leads to the high precision but extremely low recall.

R_2: As hypothesized, using the weighted scoring approach fixes this problem. The recall is significantly improved. The precision improves as well, albeit less dramatically than recall.

R_3: Subset selection, that is the inclusion of more than a single node in the result, further increases both precision and recall by small amounts.

R_4 and R_5: The two variants of ignoring forms and removing scripts, both cause additional increases in recall. Ignoring scripts causes a very slight loss of precision. Nonetheless, R_5 produces the highest F_1 measure and is therefore the optimal solution, assuming equal importance of precision and recall.

R_6: Finally, the conversion to XHTML does not significant impact performance.

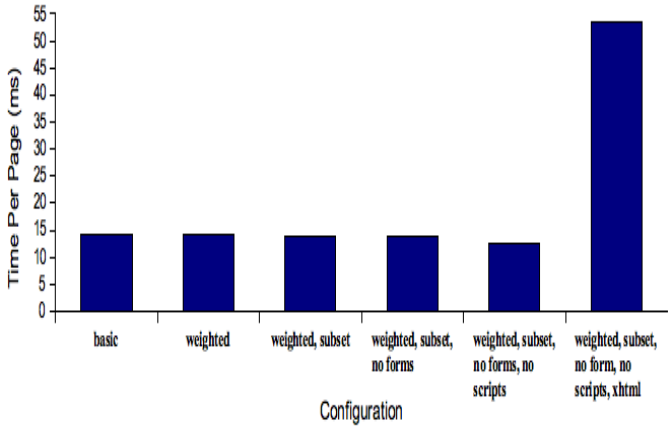

Figure 8: Processing time per page for each algorithm variant.

5.2 Breakdown by Article Parts

Figure 9 shows a breakdown of recall by different article fragments (date, byline, etc.). The numbers reflect the fraction of times the content of each type was recalled correctly. Note that the article *Body* was recalled correctly over 99% of the time. *CopyRight* was recalled the least reliably, only 20% of time. This low recall performance occurs because the copyright notice is often placed at a distant DOM location from the main article text. Similarly, the *Title* and *Date* are recalled much less frequently than the content core. We talk more about the title in the next section.

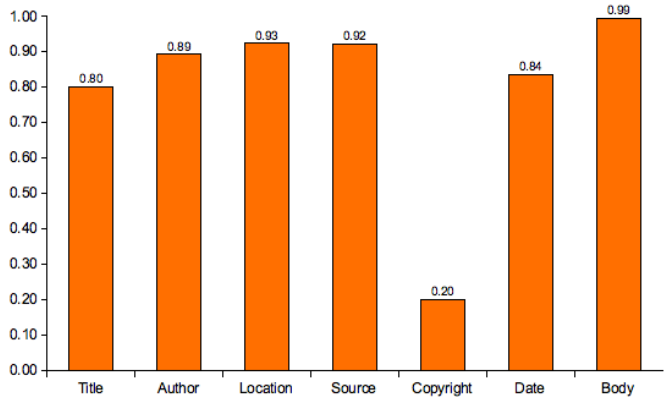

Figure 9: Recall for page fragment types.

5.3 Breakdown by source

As our dataset consists of 27 sources, we calculate average text-based precision and recall per source to probe the variation in performance across different sources. Figure 10 shows the result.

The algorithm performs well across almost all of the (news) sources, attaining perfect precision and recall for *Guardian Unlimited (guardian)*, *The San Diego Union-Tribune (sandieg_tribune)*,

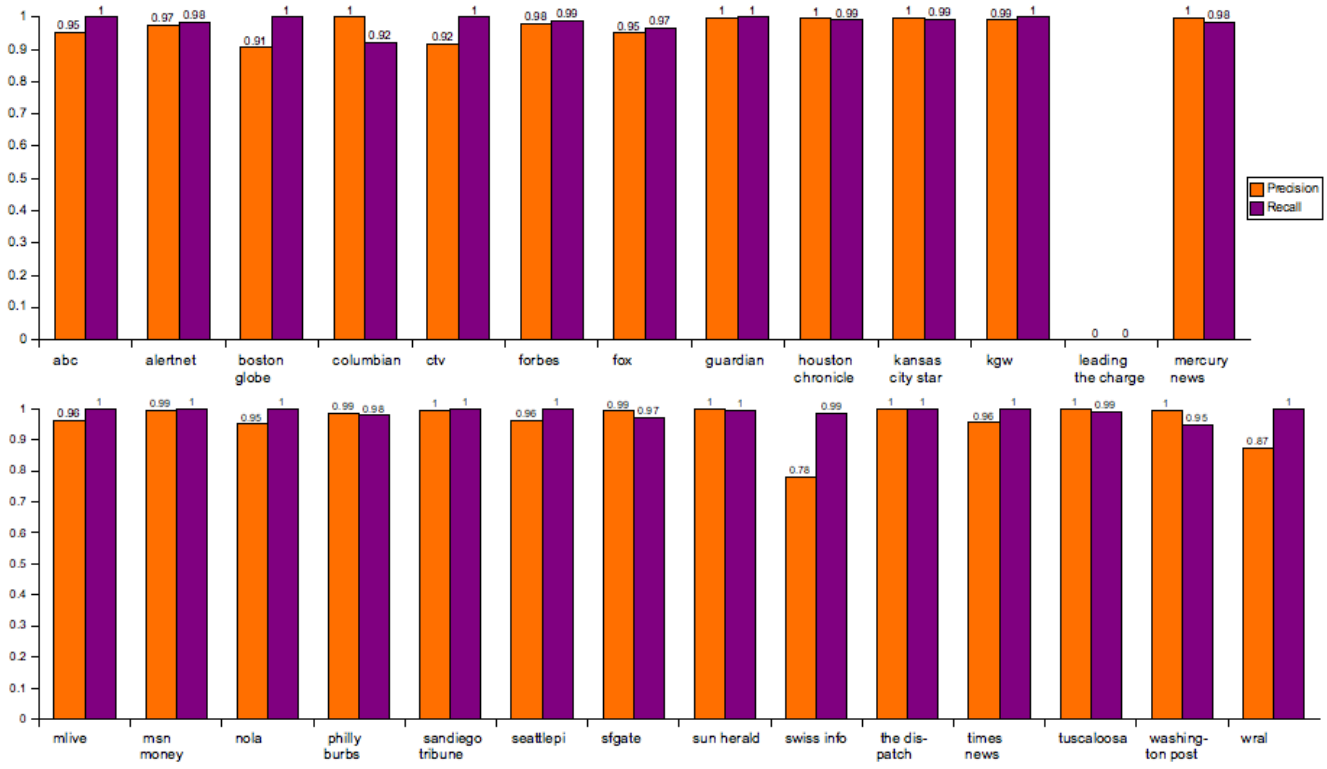

Figure 10: Breakdown of the performance by source.

The Dispatch (the_dispatch) and *SunHerald.com (sun_herald)*. For most other sources, the precision and recall are in the high 90's.

There are, however, some particular trouble points, which we now point out in the following section.

5.4 Comparison to Baseline System

As explained, we deployed for our testing a dataset that was developed for measuring performance of the algorithm in [2]. The algorithm described there is based on machine learning. The results reported for the optimal configuration is 98% precision, and 99% recall for the body of the pages.

Our best system yields 97% precision and 98% recall under the text-based scheme, and 93% precision and 92% recall under the node-based scheme. Neither of these measures are directly comparable with the numbers in [2], which follows a *block-based* scoring scheme. Each page is converted to XHTML, tokenized and divided into *blocks* based on a set of tags. The blocks are then classified as content/not-content.

We assume that if text-based precision and recall were calculated for [2], the numbers would be similar to the block-based precision and recall. In this case, their result is a percentage point better than CoreEx. On the other hand, [2] requires a training stage, and the test pages were converted to XHTML. As evident from Figure 8 this pre-processing step is quite expensive. The approach in [2] also applies Named Entity Recognition in generating their feature set, which would further increase processing time. We therefore note that CoreEx trades a percentage point in precision and recall for higher runtime efficiency. We do not have timing data from [2] for a direct speed comparison.

6. DISCUSSION

Note in Table 1 that precision increases somewhat when subset selection is added. On first thought this seems curious: if additional nodes are selected then core content precision should at best remain constant.

This apparent contradiction is resolved as follows. Consider the DOM tree in Figure 7. Depending on the particular page, one of two outcomes may occur *without* subset selection:

1. One of the three text nodes receives the highest score and is chosen as the core content. In this case adding subset selection expands the amount of material, likely improving recall, but possibly hurting precision, as intuition predicts.

2. The algorithm decides on the `<div>` node as the best choice. This will occur when (*i*)each of the text nodes by themselves contain little text compared to the overall amount of text under `<div>`. In addition, (*ii*) the number of links under the advertising nodes are insufficient to weaken the `<div>`'s score appreciably.

   Under these circumstances, subset selection actually contracts the amount of included material, and precision might benefit.

We next discuss the sources of error that we encountered, speculate on applicability to non-news sources, and summarize the lessons from the experiment.

6.1 Sources of error

Three notable outliers in Figure 10 are *Leading the Charge (leading_the_charge)*, *Swiss Info (swiss_i)* and *Wral.com (wral)*. These outliers provide insight into weaknesses of our approach.

`Some unusual HTML structures.` The zero precision and recall for *leading_the_charge* come about because the entire body of every page on this site is enclosed in an HTML `form` tag, which CoreEx ignores. This structure is highly unusual, and its intent is unclear to us. Clearly, though, such unorthodox, yet legal HTML does occur and will CoreEx will miss them.

`Lack of structural separation.` When the HTML code offers no indication that two content chunks are unrelated, then CoreEx cannot distinguish between those portions of a page. For example, the pages from *Swiss Info* include a list of dates at the bottom of the page, which CoreEx incorrectly extracted as content.

`Content in image captions.` The error in pages from *Wral.com* is due to text that is enclosed in image captions within the page. CoreEx includes this text as core content, since it appears in the middle of the article. Yet the captions were not marked as core content in the MITRE set, hence the decrease in precision.

Of course, inclusion of image captions is not necessarily incorrect, since captions of images in the middle of an article are frequently relevant to that article.

`Article Title.` The article titles are of particular concern. Clearly, they are an important part of any news article. Yet sometimes the title of the article occurs several nodes away from the body, and is therefore excluded from the extracted content. As shown in Figure 9, the best configuration correctly recalled 80% of the article titles.

One approach to handling this shortcoming is to include any text between HTML header tags, which occurs close to the article body. That is, given the extracted set of nodes S, we look at the previous siblings of S's parent and include the first header node found among them.

We tested this approach and it correctly recalled 92% of the article titles. Another workaround might be to include a node if the text is the same as the Web page title (the text that appears in the browser window's title bar), since the title of most article pages tends to be the title of the article.

6.2 Other Genres

Our approach has primarily been developed for news articles. To get an initial feel of its performance on other genres, we tested CoreEx on a small number of pages from four other genres: blogs (*boingboing.net, arstechnica.com, gigaom.com, theonlinephotographer.blogspot.com*), product pages (*amazon.com, ebay*), reviews(*cnet, yelp, dpreview*), and three facebook group pages. The algorithm correctly pulled out the article from the blogs but missed the title in all four cases. It also extracted the reviews from the review sites correctly. The performance on the product pages and the facebook group pages was inconsistent.

We expect the approach to work well for any structure where content nodes differ from the non-content nodes significantly in the amount of text and links they contain. The weights and threshold might need to be modified for other genres.

6.3 Lessons

While CoreEx is, of course, vulnerable to unusual HTML structure, an important insight is gained from Table 1's final row. It is unnecessary in this context to convert HTML pages into XHTML. XHTML is a form of HTML that is subject to stricter syntactic and structural rules. The hypothesis that this additional purity might benefit extraction algorithms is reasonable.

If this hypothesis were confirmed, however, the practical utility of CoreEx would be impacted. Note in Figure 8 that the automated conversion to XHTML is quite expensive. In aggregate over a large collection this expense would constitute a high cost.

One might expect that core content extraction would require deep semantic analysis. Such analysis would, of course, be helpful. For example, if two articles were placed on a page, separated by advertisement, a semantic analysis would likely stand a higher chance at including both articles than CoreEx.

Nevertheless, it turns out that customs on the Web favor single, full news articles per page. Extraction algorithms also benefit from the format conventions that were passed down from print media. For example, the consolidation of related materials into structural blocks, and the special formatting treatment of titles benefit extraction.

7. RELATED WORK

7.1 DOM Tree Exploration

Gupta et al.[4] and Mukherjee et al.[10] explore the idea of using a DOM tree based approach. Gupta et al.[4] describe Crunch, a content extraction tool that provides a set of customizable filters to reduce clutter from the page. They utilize the link-to-text ratio in their *link list remover* which eliminates nodes with a high link-to-text ratio. As our experiments show, using only the text-to-link ratio yields extremely low recall, and this approach alone is insufficient to extract the article from a webpage. Crunch differs from CoreEx in two main respects. Firstly, it is intended as a general tool for pages from different domains, and is not news-focused. Secondly, it uses some amount of human input, and is not fully automated. An extension of their work[3] attempts to automatically classify a Web site and use previously adjusted settings for extraction.

Mukherjee et al.[10] have developed a system to automatically annotate the content of news Web sites using structural and semantic analysis of the DOM nodes. They partition the HTML using structural analysis. The partitions are assigned semantic labels using a prefixed ontology and lexical associations with the aid of WordNet. They achieve 100% recall and precision for 35 news article pages from 8 sites. It must be pointed out here that their precision and recall are over *concept instances*, and not actual content blocks as in our system. A news article page has only one instance of the concept of *detailed news*, which their system extracts perfectly.

7.2 Scoring Blocks

Lin and Ho[9], Yin and Lee[15], and Tseng and Kao[14] present approaches that divide a page into blocks and score them to measure their importance. Lin and Ho[9] identify informative content blocks by calculating the entropy values of terms in a block based on their occurrence in a previously seen set of pages from the same site. Their results are restricted to pages that use an HTML `<table>` layout. Also, since the entropy value is calculated on a *per-page cluster* (set of pages from a website with similar structure), their system cannot process single pages from unseen websites, unlike CoreEx.

Yin and Lee[15] construct a graph model of a Web page and then apply link analysis on this graph to compute a PageRank-like importance value for each basic element. Unlike our system, they give a continuous measure of importance for every element. Their system yields a recall of around 85% for 788 news articles.

Tseng and Kao[14] propose a technique for identifying primary informative blocks by weighting the blocks using features that capture the "regularity, density and diversity" of each block. We cannot compare CoreEx with their work as they do not report results on news sites.

7.3 Machine Learning

Previous work has also applied machine learning techniques to the problem. The Columbia Newsblaster Project[1] uses an article extraction module that extracts 34 text based features used with the Ripper machine learning program.

Song et al[12] train models to learn the importance of blocks in webpages using neural networks and SVMs. The authors found that a feature based on number of links proved to be the most discriminative in their set of 42 features.

Lee et al.[8] proposed PARCELS, a system that uses a co-training approach with stylistic and lexical features to classify the blocks within a page.

Gibson et al.[2] use a Conditional Random Field sequence labeling model to label parts of the page as content/not-content.

The best performing of these is [2], which is also the system we use as our baseline.

7.4 Wrappers

Laender et al.[7] provide a brief survey of various Web data extraction tools that use wrappers. Muslea et al.[11] and Kushmerick[6] discuss automatic learning of wrappers. Metanews[5] is an information gathering agent for online news articles that employs wrappers. It removes redundant HTML tags, and uses pattern matching with site-specific manually defined patterns on the reduced page to extract news articles.

While wrappers can provide excellent text extraction, they work only on specific pages, or sets of pages that share a common layout. Once the layout changes, the wrappers need to be updated as well. This brittleness unfortunately requires continuous supervision of wrapper-based approaches.

8. CONCLUSION

Many important Web processing algorithms perform at improved levels if provided with input that is free of extraneous contents. For example, natural language processing, including topic analysis, emotional valence analysis or even the collection of basic word statistics for index optimization can benefit from focused input.

The problem of core content extraction is not trivial because Web pages usually contain navigational widgets, advertisements and branding materials that can obscure core content for automated techniques.

We presented CoreEx, a simple, heuristic approach that extracts the main content from online news article pages. Beginning with a basic version of the algorithm, we successively introduced refinements that improved performance to its final level of 97% precision at 98% recall on a given set of news articles from 27 Web sources.

We compared these results to prior work on the same dataset. While this earlier work yielded higher precision and recall, CoreEx makes due without expensive prior conversion to XHTML and is also entirely automated.

Future improvements to this work will include better handling of article titles and image captions. The algorithm shows the promise of applying to non-news genres as well. However, we have not explored such compatibility beyond small pilot experiments. A full validation will required a labeled, gold standard dataset.

Another interesting avenue to explore is to use CoreEx's scored DOM tree to differentiate between article pages and index pages of Web sites.

Multi-linguality is an aspect of our algorithm that has also remained unexplored to date. Since CoreEx does not use any language specific features, one can hope that it can be applied as is to pages in other languages. This supposition remains to be tested as well.

CoreEx yields high precision and recall, works fast and is consistent across different domains.

9. ACKNOWLEDGMENTS

We owe special thanks to John Gibson, Susan Lubar, and Ben Wellner of the MITRE corporation for compiling, and then providing us with their dataset for our tests. The use of common test sets made our research much more meaningful and allowed direct comparison with prior art.

10. REFERENCES

[1] D. K. Evans, J. L. Klavans, and K. R. McKeown. Columbia newsblaster: Multilingual news summarization on the web. In *Proceedings of Human Language Technology Conference/North American chapter of the Association for Computational Linguistics Annual Meeting*, pages 1–4, 2004.

[2] J. Gibson, B. Wellner, and S. Lubar. Adaptive web-page content identification. In *WIDM '07: Proceedings of the 9th Annual ACM International Workshop on Web Information and Data Management*, pages 105–112, New York, NY, USA, 2007. ACM.

[3] S. Gupta. *Context-based Content Extraction of HTML Documents*. PhD thesis, New York, NY, USA, 2006. Adviser-Gail E. Kaiser.

[4] S. Gupta, G. E. Kaiser, P. Grimm, M. F. Chiang, and J. Starren. Automating content extraction of html documents. *World Wide Web*, 8(2):179–224, 2005.

[5] D.-K. Kang and J. Choi. Metanews: An information agent for gathering news articles on the web. In *KDD '02: Proceedings of the 8th ACM SIGKDD*

International Conference on Knowledge Discovery and Data Mining, pages 588–593, New York, NY, USA, 2002. ACM.

[6] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artif. Intell.*, 118(1-2):15–68, 2000.

[7] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Rec.*, 31(2):84–93, 2002.

[8] C. H. Lee, M.-Y. Kan, and S. Lai. Stylistic and lexical co-training for web block classification. In *WIDM '04: Proceedings of the 6th Annual ACM International Workshop on Web Information and Data Management*, pages 136–143, New York, NY, USA, 2004. ACM.

[9] S.-H. Lin and J.-M. Ho. Discovering informative content blocks from web documents. In *KDD '02: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 588–593, New York, NY, USA, 2002. ACM.

[10] S. Mukherjee, G. Yang, and I. Ramakrishnan. Automatic annotation of content-rich html documents: Structural and semantic analysis, 2003.

[11] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In *AGENTS '99: Proceedings of the 3rd Annual Conference on Autonomous Agents*, pages 190–197, New York, NY, USA, 1999. ACM.

[12] R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma. Learning important models for web-page blocks based on layout and content analysis. *SIGKDD Explor. Newsl.*, 6(2):14–23, 2004.

[13] M. Theobald, J. Siddharth, and A. Paepcke. Spotsigs: Robust and efficient near duplicate detection in large web collections. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2008. Accessible at http://dbpubs.stanford.edu/pub/2008-14.

[14] Y.-F. Tseng and H.-Y. Kao. The mining and extraction of primary informative blocks and data objects from systematic web pages. In *WI '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 370–373, Washington, DC, USA, 2006. IEEE Computer Society.

[15] X. Yin and W. S. Lee. Using link analysis to improve layout on mobile devices. In *WWW '04: Proceedings of the 13th International Conference on World Wide Web*, pages 338–344, New York, NY, USA, 2004. ACM.