# Automatic Extraction of Informative Blocks from Webpages

Sandip Debnath, Prasenjit Mitra, C. Lee Giles,

Department of Computer Sciences and Engineering
The Pennsylvania State University
University Park, PA 16802 USA
debnath@cse.psu.edu

School of Information Sciences and Technology
The Pennsylvania State University
University Park, PA 16802 USA
pmitra@ist.psu.edu, giles@ist.psu.edu

## ABSTRACT

Search engines crawl and index webpages depending upon their informative content. However, webpages — especially dynamically generated ones — contain items that cannot be classified as the "primary content", e.g., navigation sidebars, advertisements, copyright notices, etc. Most end-users search for the primary content, and largely do not seek the non-informative content. A tool that assists an end-user or application to search and process information from webpages automatically, must separate the "primary content blocks" from the other blocks. In this paper, two new algorithms, *ContentExtractor*, and *FeatureExtractor* are proposed. The algorithms identify primary content blocks by i) looking for blocks that do not occur a large number of times across webpages and ii) looking for blocks with desired features respectively. They identify the primary content blocks with high precision and recall, reduce the storage requirement for search engines, result in smaller indexes and thereby faster search times, and better user satisfaction. While operating on several thousand webpages obtained from 11 news websites, our algorithms significantly outperform the Entropy-based algorithm proposed by Lin and Ho [7] in both accuracy and run-time.

## Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

## Keywords

Electronic Publishing, Data Mining, Information Systems Applications.

## 1. INTRODUCTION

Search engines crawl the World-Wide Web to collect webpages. These pages are stored and indexed. An end-user who performs a search using a search engine is interested in the *primary informative content* of the webpage. However, a substantial part of webpages — especially those that are created dynamically — is content that cannot be classified as the primary informative content of the webpage. These blocks are not relevant to the main content of the page and are seldom sought by the users of the website. We refer to such blocks as *non-content blocks*. Non-content blocks are very common in dynamically generated webpages. Typically such blocks contain advertisements, image-maps, plug-ins, logos, counters, search boxes, category information, navigational links, related links, footers and headers, and copyright information.

Before the content from a webpage can be used, it must be subdivided into smaller semantically-homogeneous components based on their content. We refer to such components as *blocks* in the rest of the paper. A block (or webpage block) $\mathcal{B}$ is a portion of a webpage enclosed within an open-tag and its matching close-tag, where the open and close tags belong to an ordered tag-set $\mathcal{T}$ that includes tags like <TR>, <P>, <HR>, and <UL>. Figure 1, shows a webpage obtained from CNN's website[1] and the blocks in that webpage. In this paper, we address the problem of identifying the primary informative content blocks of a webpage (e.g., in the figure, the text block containing the news is the primary content block).

An added advantage of identifying blocks in webpages is that if the user does not require the non-content blocks or requires only a few non-content blocks, we can delete the rest of the blocks. This contraction is useful in situations where large parts of the web are crawled, indexed and stored. Since the non-content blocks are often a significant part of dynamically generated webpages, eliminating them results in significant savings with respect to storage and indexing.

We propose simple yet powerful algorithms, called *ContentExtractor* and *FeatureExtractor*, to identify and separate content blocks from non-content blocks. We have characterized different types of blocks based on the different features they possess. *FeatureExtractor* is based on this characterization and uses heuristics based on the occurrence of certain features to identify content blocks. *ContentExtractor* identifies non-content blocks based on the appearance of the same block in multiple webpages.

First, the algorithms partition the webpage into blocks based on heuristics. Lin and Ho [7] have proposed an entropy-based algorithm that partitions a webpage into blocks on the basis of HTML tables. In contrast, not only do we consider HTML tables, but also other tags and use heuristics to parti-

[1] http://www.cnn.com


*SAC'05* March 13-17, 2005, Santa Fe, New Mexico, USA

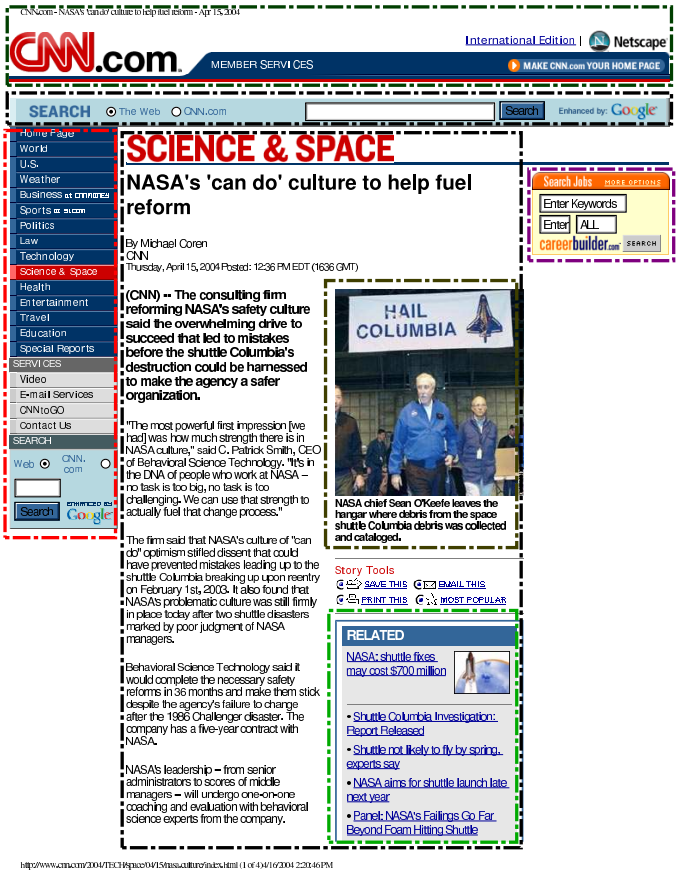


**Figure 1: A webpage from CNN.com and its blocks (shown using boxes)**

tion a webpage. Second, our algorithms classifies each block as either a content block or a non-content block. While the algorithm decides whether a block, $\mathcal{B}$, is content or not, it also compares $\mathcal{B}$ with stored blocks to determine whether $\mathcal{B}$ is similar to a stored block. If $\mathcal{B}$ already exists in the repository, a second copy is not stored; instead a pointer to the stored block identical to $\mathcal{B}$ is retained.

Both *FeatureExtractor* and *ContentExtractor* produce excellent precision and recall values and above all, do not use any manual input and require no complex machine learning process. While operating on several thousand webpages obtained from 11 news websites, our algorithms significantly outperform the Entropy-based blocking algorithm proposed by Lin and Ho [7].

The rest of the paper is organized as follows: In Section 2 we have discussed the related work. We describe our algorithms in Section 3. We outline our performance evaluation plan and the data set on which we ran our experiments in Section 4. Then, we compare our algorithms with the *LH* algorithm in Section 5 and conclude thereafter.

## 2. RELATED WORK

Our work is closely related to the work by Lin and Ho [7]. We have implemented their algorithm and show that the simpler algorithms that we propose outperforms them with respect to accuracy and scalability. Yi and Liu [10, 8] have proposed an algorithm for identifying non-content blocks (the refer to it as "noisy" blocks) of webpages. Their algorithm is based on constructing "Style Tree"s and constructing entropies of blocks. Our algorithms use simple heuristics to determine non-content blocks, it does not incur the overhead of constructing "Style Tree"s and hence outperforms their algorithm in runtime, but achieves comparable accuracy for the experimental set we used. Bar-Yossef and Rajagopalan [1] have proposed a method to identify frequent templates of webpages and pagelets (identical to our blocks). Yi and Liu argue that their entropy-based method supersedes the template identification method. We show that our method produces better result than the entropy-based method. Kushmerick [6, 5] has proposed a feature-based method that identifies internet advertisements in a web-page. This method is designed only to remove advertisements, requires training sets and manual procedures to specify the feature selection. Our approach does not require any manual training. Other related works include [3, 4, 2, 9], which have tried to extract information that originally came from databases.

## 3. ALGORITHMS

We now discuss the two algorithms *ContentExtractor* and *FeatureExtractor*

### 3.1 ContentExtractor

The ContentExtractor algorithm, shown in Algorithm 1 eliminates blocks depending upon the inverse block-document frequency, $IBDF$, of a block. The inverse block-document frequency, $IBDF$, is inversely proportional to the number of documents in which a block occurs or has a similar block. Blocks that are similar to blocks occurring in multiple pages in the same domain, e.g. blocks that occur in multiple pages at cnn.com, are identified as redundant blocks. Blocks that occur only in one page are identified as content-blocks. If $\mathcal{S}$ is a set of Web pages of the same class, i.e., obtained from the same source. Then

$$\mathcal{S} = \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \ldots \mathcal{P}_M\}.$$

Let us assume $IBDF^i$ represents the IBDF of a block $\mathcal{B}_i$ in a set of pages $\mathcal{S}$. Typically, the set $\mathcal{S}$ consists of similar pages from the same source. $IBDF^i$ is inversely proportional to the number of webpages the block $\mathcal{B}_i$ occurs in. Then

$$IBDF^i \equiv f(\frac{1}{|\mathcal{S}^i| + 1})$$

where

$$\mathcal{S}^i = \cup\{\mathcal{P}_l : sim(\mathcal{B}_i, \mathcal{B}_k) < \epsilon, \forall \mathcal{B}_k \in \mathcal{P}_l, \forall \mathcal{P}_l \in \mathcal{S}\}.$$

$f$ denotes a function, usually linear or log function. The function $sim(\mathcal{B}_i, \mathcal{B}_k)$ is a similarity measure of the two blocks. An expert provides the threshold $\epsilon$. Given two blocks, the similarity measure, $sim$, returns the cosine between their block feature vectors. Examples of features are: the number of terms, the number of images, the number of java-scripts, etc. However, for text blocks, simply taking the number of terms in the block may result in falsely identifying two blocks as similar. Therefore, we augment the features by adding a binary feature for each term in the corpus of documents. If a feature occurs in a block, the entry in the corresponding feature vector is a one, otherwise it is zero. We used a threshold value of $\epsilon = 0.9$. That is, if the similarity measure

is greater than the threshold value, then the two blocks are accepted as identical.

## 3.2 FeatureExtractor

The *FeatureExtractor* algorithm, shown in Algorithm 2, is invoked to identify blocks with a set of desired features. Within the set of chosen blocks we sort the blocks again according to their probability values, and chose the winner block.

The pseudo-code for the *FeatureExtractor* algorithm is shown below.

# 4. EVALUATION PLAN

We implemented and compared our algorithm with *LH*, the entropy-based algorithm, proposed by Lin and Ho [7]. They use the terms *precision* and *recall* to refer to the metrics to evaluate their algorithm. Although, the use of these terms are somewhat different from their usual sense in the "Information Retrieval" field, in order to avoid confusion, we use the same terms to refer to the evaluation metrics of our work.

## 4.1 Metric Used

Precision is defined as the ratio of the number of relevant items (actual primary content blocks) $r$ found and the total number of items (primary content blocks suggested by an algorithm) $t$ found. $Precision = \frac{r}{t}$. Recall has been defined as the ratio of the number of relevant items found and the desired number of relevant items. The desired number of relevant items includes the number of relevant items found and the missed relevant items $m$. $Recall = \frac{r}{r+m}$.

## 4.2 Data Set

Like Lin and Ho, we chose several websites from the news domain. We crawled the web for news articles and other types of websites to collect documents. The details (name, source, category, number) of the dataset are shown in Table 1.

We took 15 different news websites whose design and page-layouts are completely different. Unlike Lin and Ho's dataset [7] that is obtained from one fixed category of news sections (only one of them is "Miscellaneous" news from CDN), we took random news pages from every section of a particular website. This choice makes the dataset a good mix of a wide variety of HTML layouts. This step was necessary to compare the robustness of their algorithm to ours.

# 5. PERFORMANCE COMPARISON

We implemented all three algorithms in Perl 5.8.0 on a Pentium-based Linux platform. With the help of a few graduate students and professors, we calculated the precision and recall values for each website and layout category for text feature. These values are shown in table 2.

Our algorithms outperform *LH* in all news sites in all categories. The recall is always good since both algorithms could find most relevant blocks but the results obtained by running the *LH* algorithm were less precise than those obtained by *ContentExtractor* since the former algorithm also includes lots of other non-content blocks.

## 5.1 Precision and Recall

**Algorithm 1:** *ContentExtractor algorithm and GetBlockSet function. GetBlockSet function is also used by FeatureExtractor algorithm*

**Input** : Set $\mathcal{S}$ of HTML pages, Sorted tag-set $\mathcal{T}$
**Output**: Primary Content Blocks and their associated pages in $\mathcal{S}$

**begin**
- $\mathcal{M}_{BD} \longleftarrow \emptyset$
- { Here the $\mathcal{M}_{BD}$ matrix is the block-document matrix where rows represent document and columns represent block identifier.}
- **for** *each* $H^k \in \mathcal{S}$ **do**
  - { Here $B^k$ represents the $k$th row of the $\mathcal{M}_{BD}$ matrix. }
  - $B^k \longleftarrow$ **GetBlockSet**$(H^k, \mathcal{T})$
  - $\mathcal{M}^k_{BD} \longleftarrow B^k$
- **for** *each* $b_{ij} \in \mathcal{M}_{BD}$ **do**
  - $IBDF_{ij} \longleftarrow 1$
  - **for** *each* $b_{kl} \in \mathcal{M}_{BD}$ **do**
    - { Here $i \neq k$.}
    - $sim_{ijkl} \longleftarrow sim(b_{ij}, b_{kl})$
    - **if** $sim_{ijkl} > \epsilon$ **then**
      - $IBDF_{ij} \longleftarrow Update(IBDF_{ij})$
      - {Update Recalculates IBDF}
- { If IBDF value above threshold we will produce the output }
- **for** *each* $b_{ij} \in \mathcal{M}_{BD}$ **do**
  - **if** $IBDF_i > \theta$ **then**
    - Output the content of the block

**end**

**Function: GetBlockSet**
**Input** : HTML page $H$, Sorted tag-set $\mathcal{T}$
**Output**: Set of Blocks in $H$

**begin**
- $B \longleftarrow H$; // set of blocks, initially set to H.
- $f \longleftarrow Next(\mathcal{T})$
- **while** $f \neq \emptyset$ **do**
  - $b \longleftarrow First(B)$
  - **while** $b \neq \emptyset$ **do**
    - **if** $b$ *contains* $f$ **then**
      - $B^N \longleftarrow GetBlocks(B, f)$
      - $B \longleftarrow (B - b) \cup B^N$
    - $b \longleftarrow Next(B)$
  - $f \longleftarrow Next(\mathcal{T})$

**end**

| *Site* | *Address* | *Category* | *Number* |
|---|---|---|---|
| ABC | http://www.abcnews.com | Main Page, USA, World, Business, Entertainment, Science/Tech, Politics,Living | 415 |
| BB | http://www.bloomsberg.com | Main Page, World, Market, US Top Stories, World Top Stories, Asian, Australia/New Zealand, Europe, The Americas | 510 |
| BBC | http://www.bbc.co.uk | Main Page, The Continents, Business, Health, Nature, Technology, Entertainment | 890 |
| CBS | http://www.cbsnews.com | Main Page, National, World, Politics, Technology, Health, Entertainment | 370 |
| CNN | http://www.cnn.com | Main Page, World, US, All Politics, Law, Tech(nology), Space (Technology), Health, Showbiz, Education, Specials | 717 |
| FOX | http://www.foxnews.com | Main Page, Top Stories, Politics, Business, Life, Views | 476 |
| FOX23 | http://www.fox23news.com | Main Page, General, Local, Regional, National, World, In Depth, Sports, Business, Entertainment, Health | 658 |
| IE | http://www.indianexpress.com | Main Page, International, Sports, National Network, Business, Headlines | 269 |
| IT | http://www.indiatimes.com | Main Page, Main Stories, Top Media Headlines | 454 |
| MSNBC | http://www.msnbc.com | Main Page, Business, Sports, Technology an Science, Health, Travel | 647 |
| YAHOO | http://news.yahoo.com | Main Page, Top Stories, US (National), Business, World, Entertainment, Sports, Technology, Politics, Science | 505 |

**Table 1: Details of the dataset. Number of pages taken from individual categories are not shown due to the enormous size of the latex table. But interested reader can contact authors to get the details.**

| **Site** | **Prec of LH** | **Recall of LH** | **F-measure of LH** | **Prec of CE** | **Recall of CE** | **F-measure of CE** | **Prec of FE** | **Recall of FE** | **F-measure of FE** |
|---|---|---|---|---|---|---|---|---|---|
| ABC | 0.811 | 0.99 | **0.89** | 0.915 | 0.99 | **0.95** | 1.00 | 1.00 | **1.00** |
| BB | 0.882 | 0.99 | **0.93** | 0.997 | 1.00 | **0.998** | 1.00 | 1.00 | **1.00** |
| BBC | 0.834 | 0.99 | **0.905** | 0.968 | 1.00 | **0.983** | 1.00 | 1.00 | **1.00** |
| CBS | 0.823 | 1.00 | **0.902** | 0.972 | 1.00 | **0.985** | 0.98 | 0.977 | **0.978** |
| CNN | 0.856 | 1.00 | **0.922** | 0.977 | 1.00 | **0.988** | 0.98 | 0.98 | **0.98** |
| FOX | 0.82 | 1.00 | **0.901** | 0.967 | 1.00 | **0.983** | 1.00 | 0.99 | **0.994** |
| FOX23 | 0.822 | 1.00 | **0.902** | 0.985 | 1.00 | **0.992** | 1.00 | 1.00 | **1.00** |
| IE | 0.77 | 0.95 | **0.85** | 0.911 | 0.993 | **0.95** | 0.93 | 0.99 | **0.959** |
| IT | 0.793 | 0.99 | **0.878** | 0.924 | 0.981 | **0.951** | 0.96 | 0.98 | **0.969** |
| MSNBC | 0.802 | 1.00 | **0.89** | 0.980 | 1.00 | **0.989** | 0.92 | 1.00 | **0.95** |
| YAHOO | 0.730 | 1.00 | **0.84** | 0.967 | 1.00 | **0.98** | 1.00 | 0.95 | **0.974** |

**Table 2: Block level Precision and Recall values from *LH* algorithm, *ContentExtractor* and *FeatureExtractor*. The second, third, and fourth columns are from *LH* algorithm, the fifth, sixth, and the seventh columns are from *ContentExtractor* and the eighth, ninth, and tenth columns are from *FeatureExtractor***

**Algorithm 2:** FeatureExtractor

**Input** : HTML pages $H$, Sorted Tag Set $\mathcal{F}$, Desired Feature $\mathcal{F}_I$

**Output**: Content Blocks of $H$

**Feature**: Feature set $\mathcal{F}_S$ used for block separation sorted according to importance taken from $\mathcal{F}$

**begin**

$B \longleftarrow GetBlockSet(H, \mathcal{F})$

**for** *each* $b \in B$ **do**

$P_1 \longleftarrow Pr(\mathcal{F}_I|\mathcal{F})$

**if** $P_1 > 0.5$ **then**

$\mathcal{W} \longleftarrow W \cup b$

**for** *each* $b \in \mathcal{W}$ **do**

$P_b \longleftarrow Pr(\mathcal{F}_I|\mathcal{F}, \mathcal{W})$

{ Output: Sort $\mathcal{W}$ according to the Probability value $P_b$ and (1) Produce the content of the Winner block }

**end**

Both *FeatureExtractor* and *ContentExtractor* performed better than *LH* in almost all cases.

We compare the features of all three algorithms in Table 3.

## 5.2 Execution Time

Figure 2 shows execution time taken by the three algorithms averaged over all test webpages.

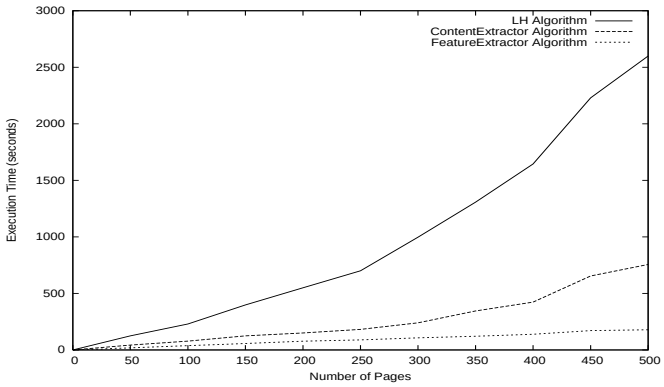


**Figure 2: Run-times for the LH and the ContentExtractor algorithm.**

Here in table 3 we present a comparison table for the features of both algorithms.

# 6. CONCLUSIONS AND FUTURE WORK

We devised simple, yet powerful, and modular algorithms, to identify primary content blocks from webpages. Our algorithms outperformed the LH algorithm significantly, in precision as well as run-time, without the use of any complex learning technique. The *FeatureExtractor* algorithm, provided a feature, can identify the primary content block with respect to that feature. The *ContentExtractor* algorithm detects redundant blocks based on the occurrence of the same block across multiple webpages. The algorithms, thereby, reduce the storage requirements, make indices smaller, and result in faster and more effective searches.

| *Property* | *LH* | *ContentExtractor* |
|---|---|---|
| Precision | Low | *High* |
| Recall | High | *Very High* |
| Number of pages needed | All the pages to calculate Entropy of features | *Very few (5 − 10) pages from same class are enough to give high performance* |
| Time of completion | Always more than ContentExtractor | *Less than LH (shown in figure 2)* |

**Table 3: A property-wise comparison table for both algorithms**

# 7. REFERENCES


[1] Ziv Bar-Yossef and Sridhar Rajagopalan. Template detection via data mining and its applications. In *Proceedings of WWW 2002*, pages 580–591, 2002.

[2] Boris Chidlovskii, Jon Ragetli, and Maarten de Rijke. Wrapper generation via grammar induction. In *Machine Learning: ECML 2000, 11th European Conference on Machine Learning, Barcelona, Catalonia, Spain, May 31 - June 2, 2000, Proceedings*, volume 1810, pages 96–108. Springer, Berlin, 2000.

[3] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 109–118, 2001.

[4] C. Hsu. Initial results on wrapping semistructured web pages with finite-state transducers and contextual rules. In *AAAI-98 Workshop on AI and Information Integration*, pages 66–73. AAAI Press, 1998.

[5] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.

[6] Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper induction for information extraction. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[7] Shian-Hua Lin and Jan-Ming Ho. Discovering informative content blocks from web documents. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 588–593, 2002.

[8] Bing Liu, Kaidi Zhao, and Lan Yi. Eliminating noisy information in web pages for data mining. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 296–305, 2003.

[9] Ion Muslea, Steven Minton, and Craig A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[10] Lan Yi, Bing Liu, and Xiaoli Li. Visualizing web site comparisons. In *Proceedings of the eleventh international conference on World Wide Web*, pages 693–703, 2002.