Content Code Blurring: A New Approach to Content Extraction

Thomas Gottron
Institut für Informatik
Johannes Gutenberg-Universität Mainz
55099 Mainz, Germany
Email: gottron@uni-mainz.de

***Abstract*— Most HTML documents on the World Wide Web contain far more than the article or text which forms their main content. Navigation menus, functional and design elements or commercial banners are typical examples of additional contents. Content Extraction is the process of identifying the main content and/or removing the additional contents. We introduce *content code blurring*, a novel Content Extraction algorithm. As the main text content is typically a long, homogeneously formated region in a web document, the aim is to identify exactly these regions in the document in an iterative process. Comparing its performance with existing Content Extraction solutions we show that for most documents content code blurring delivers the best results.**

I. Introduction

Nowadays most HTML documents on the World Wide Web are generated from templates by content management systems. Beside the main textual content they comprise several additional contents, such as navigation menus, functional and design elements or commercials. Already in 2005 Gibson, Punera and Tomkins [3] estimated those additional contents to make up around 40 to 50% of most web pages on the Internet, predicting this ratio to increase constantly.

Content Extraction (CE) is the process of determining the parts of an HTML document which contain its main textual content. Several applications benefit from CE under different aspects: Web Mining and Information Retrieval applications use CE to pre-process the raw HTML data to reduce noise and to obtain more accurate results, other applications use CE to rewrite web pages to improve presentation on small screen devices or access via screen readers for visually impaired users. The aim of reducing the size of documents can also be to speed up the download time for devices with narrow bandwidth access.

This paper introduces *content code blurring* (CCB), a novel CE algorithm. CCB is robust to invalid or badly formatted HTML documents, it is fast and concerning its extraction performance delivers very good results on most documents.

We proceed as follows. In section II we give an overview of related works in the field of CE, mentioning in particular some of the algorithms we will use for comparison when evaluating CCB. The CCB algorithm itself is defined and explained in section III. We continue by describing our evaluation setup and compare the performance of CCB with other CE methods in IV. The paper is concluded with a discussion of the results and a proposal for further extensions to the algorithm.

II. Related works

In most application scenarios mentioned in the introduction the identification and extraction of the main content has to be done on-the-fly. Rahman et al. list in [11] requirements such a Content Extraction system for HTML documents should comply with. Being generic enough to work with any website and using a fast extraction algorithm are the most important aspects for the extraction part of such a system.

One of the more prominent solutions for CE is the Crunch framework. It was introduced by Gupta et al. in [6] and is refined continuously [8], [7], [5]. Avoiding a one-solution-fits-all approach, Crunch combines several heuristics to discover and remove e.g. link lists, text lists, blocks with a too high link ratio or commercial banners. The main objective of Crunch is to optimise HTML documents for presentation on small screen devices or to improve accessibility for users employing screen reader software. A detection of link lists is also used in the link quota filter (LQF) of Mantratzis et al. in [9]. Debnath et al. developed the Feature Extractor algorithm and its extension the K-Feature Algorithm in [1]. The underlying idea is to segment a web document in blocks and analyse these blocks for the presence and prevalence of particular features like text, images, JavaScript etc. The extraction process is based on retrieving those blocks which correspond best to certain desired features, e.g. text for a classical article main content. Finn et al. introduced the Body Text Extraction (BTE) algorithm in [2] as a pre-processor for their application classifying news articles on the web. BTE identifies a part of the document which contains most of the text while excluding most of the tags. Pinto et al. [10] extended the BTE approach to construct Document Slope Curves (DSC). They use a windowing technique to locate several parts of the document which fit the main content characteristics as formulated for BTE in order to overcome BTE's drawback of extracting only a single and continuous part of the document.

In [4] we developed a way to measure, evaluate and compare CE algorithms based on the standard IR measures precision, recall and F1. In the course of this work we also compared different CE approaches. An adaptation of the DSC algorithm turned out to be the best performing general CE method.

Fig. 1. An example for a web document with an outlined main content. The main text content is usually a long and homogeneously formatted region, while additional contents, such as navigation menu, commercials or layout elements contain short texts and are highly structured.

III. Content Code Blurring

The idea underlying content code blurring is to take advantage of typical visual features of the main and the additional contents. Additional contents are usually highly formatted and contain little and short texts. The main text content on the other hand commonly is long and homogeneously formatted. The example document in fig. 1 demonstrates this observation.

As in the source code of an HTML document any change of format is indicated by a tag, we will accordingly try to identify those parts of a document which contain a lot of text and few or no tags. This corresponds to finding areas with a lot of content and little code.

A. Concept and Idea

So, the idea and aim of content code blurring is to locate those regions in a document which contain mainly content and little code. To formalise this task we need to define what is a region in a document, what we mean by content and by code and how to measure the amount of content or code in a region.

The question of what is content and what is code can be answered quite easily. Roughly said, all the tags in the source code correspond to code while everything else is content. After all, the tags provide the structure, layout and formatting of a web document. The text outside the tags instead makes up the content.

The next question is, how to turn a document into a structure for which we can define the concept of regions. We will take two different approaches here. The first approach is striking a new path for document representations in the CE context by determining for each single character whether it is content or code. So, a document is turned into a sequence of code and content characters. The second approach is based on a token sequence as used by BTE and DSC. Each tag and each word correspond to a token. The whole document is accordingly represented as a sequence of tag and word tokens. Both ways lead to a representation of a document as a sequence of elements which are either content or code. This sequence can be characterised as a vector of atomic content or code elements. We will refer to this vector from now on as the *content code vector*. Note, that this document representation is very robust to syntax errors in the HTML code: as long as the tags can be identified this vector can be built.

Now, based on this vector of atomic elements we will form regions, which are based on the criteria of consisting mainly of content or of code. Accordingly, it is immediately clear whether to retain or to discard a region in an extraction process. To determine these regions, we will calculate for each atomic element a *content code ratio* of how many content and code elements surround it. If the content code ratio is high for several atomic elements in a row we have found a part of the document with a relatively uniform format, as it implies few tags in a part of the document which mainly consists of text.

How much an element is surrounded by content or code depends on the appearance of content or code elements in its neighbourhood. The neighbourhood is defined individually for each atomic element and corresponds to a symmetric range of entries in the content code vector. To calculate the content code ratio in this neighbourhood we use a process inspired by the blurring filters of image processing applications.

B. Blurring the Content Code Vector

Before starting the process of determining the content code ratio we will change the representation of the content code vector into a more suitable format. We will represent it as a vector of floating point values. Each entry in the vector is initialised with a value of 1 if the according element is of type content and with a value of 0 for code.

To obtain the content code ratios we calculate a weighted and local average of the values in the neighbourhood of each entry, i.e. for each atomic element. Based on these local average values we create a new vector which represents for each element the individual ratio of content and code in its neighbourhood. If all the elements in a neighbourhood started with a value of 1, also the neighbourhood average will be 1. The same is valid for neighbourhoods with an initial value of 0. In mixed neighbourhood the resulting average will be between 0 and 1 and depends on the values of the surrounding elements. If they are mainly content, the ratio will be high, if they are mainly code, the ratio will be low – exactly the effect that we intended to achieve.

The weights in the average calculation are used for modelling a stronger influence of near elements and a weaker influence for those further away. We chose the weights according to a Gauss distribution to obtain this effect. To further realise an

influence of elements which are beyond the neighbourhood boundaries we iteratively repeat this process of calculating neighbourhood averages. In each iteration we use the resulting vector of content code ratios as input for the next step. The iteration is stopped as soon as the values start to settle.

Visually the whole process corresponds to constructing a one dimensional image from the atomic elements, in which each pixel represents a single element and is initially coloured white if it represents content and black if it represents code. The iterative calculation of the content code ratio corresponds to applying repeatedly a Gaussian blurring filter[1] – hence the name content code blurring.

Figure 2 demonstrates this visual interpretation. The original image has been generated from HTML source code as described above. When blurring the image the abrupt transitions between black and white are smoothed by shades of grey. The parts of the image which were initially mainly black end up being coloured in darker shades, those which have initially been mainly white will remain in brighter shades. Translated into the content code ratio, the bright areas have a high ratio of content to code and are accordingly rich in content, the dark ones have a low ratio and are rich in code.

Finding the regions in a document which contain mainly content then corresponds to selecting those elements which have a high content code ratio, i.e. a value closer to 1 or a brighter colour in the image interpretation. We will use a fixed threshold for this ratio and select all elements of the document which have a content code ratio above this threshold as being part of the main content.

C. Adaptation and Implementation

Though the visual interpretation of blurring a black and white image is very descriptive the overhead of creating an image representation of the HTML code is not necessary. Instead we use the afore mentioned way of calculating weighted averages of the content code vector of floating point values.

The iteration of this calculation is stopped when meeting a certain stop criterion. We will use a low rate of changes in the finally as main content extracted text as stop criterion. This means, we determine after each step of the iteration the content which would be declared main content given the current content code ratios of the atomic elements. And if this extracted content is not changing any more, the iteration stops.

The extraction itself is based on the final values of the content elements. If their value is above a threshold t they are considered main content, otherwise additional content and are removed from the source code. A few downstream refinements take care that in the character based version words are always extracted entirely. We will refer to this initial form of the algorithm as CCB.

In [4] we observed that most CE algorithms have problems with highly fragmented contents which additionally contain a lot of in-text hyperlinks. This caused all CE methods to perform remarkably poor on wiki style web documents.

Especially the in-text links contribute very strong to the fragmentation, which is lethal also to CCB's attempt to find areas with few tags. Hence we will analyse a variation of CCB which is intended to cope with this problem. Thinking of our initial idea of text blocks with uniform layout we create an *adapted CCB* (ACCB) which ignores anchor-tags entirely during the creation of the content code vector. After all, hyperlinks are not influencing the format intentionally. Their visual influence is more a side-effect of the necessity to reference another document.

At the first glance, this approach might seem too specialised for the wiki style pages and even counterproductive for other HTML documents. LQF for example uses the presence of hyperlinks as a sure sign for additional contents. Ignoring hyperlinks might accordingly weaken the general extraction performance. Hence, we will pay special attention to the performance of ACCB in comparison with the original CCB.

So, for the evaluation we will end up with three variations of content code blurring. The character based version in its original form (CCB), with the adaptation of ignoring hyperlinks (ACCB) and the token based version (TCCB).

[1]Gaussian filters can be found in nearly all image processing programs. They achieve the blurred effect in an image by spreading a pixels colour value to its neighbour pixels according to a Gauss distribution.

TABLE I
OVERVIEW OF THE EVALUATION PACKAGES.

Package	Web site	URL	Size
bbc	BBC online	http://news.bbc.co.uk	1000
chip	Chip online	http://www.chip.de	361
economist	Economist.com	http://www.economist.com	250
espresso	L'espresso	http://espresso.repubblica.it	139
golem	Golem	http://golem.de	1000
heise	heise online	http://www.heise.de	1000
manual	several	–	65
repubblica	La Repubblica.it	http://www.repubblica.it	1000
slashdot	Slashdot	http://slashdot.org	364
spiegel	Spiegel online	http://www.spiegel.de	1000
telepolis	Telepolis	http://www.telepolis.de	1000
wiki	Wikipedia	http://de.wikipedia.org	1000
yahoo	Yahoo! news	http://news.yahoo.com	1000
zdf	ZDF heute.de	http://www.heute.de	422

IV. EVALUATION

For evaluation, we will use the same evaluation methods as in [4]. We collect web documents and provide a gold standard for their main text content. To compare the extract provided by a CE algorithm with the gold standard we need to compute an overlap between the texts. For this purpose we determine the longest common (but not necessarily continuous) sub sequence of words in both texts. Considering this sub sequence as the intersection between retrieved (i.e. extracted) and relevant text (i.e. part of the gold standard) allows to apply standard IR measures like recall, precision and F1.

The evaluation data is organised in packages which are listed in table I. The manual package consists of documents for which the main contents have been outlined manually. For the other packages we applied dedicated programs, which are capable of harvesting the main content from the documents of

Fig. 2. The blurring of a content code vector interpreted as a grayscale image.

particular web sites. In this way we obtained large amounts of documents for large scale tests. Altogether we have 14 packages and a total of 9,601 documents for evaluating CE algorithms. These packages cover different scenarios of document styles, layout techniques and main content lengths.

In most documents the main content is quite obvious. The only problematic case are the documents of the slashdot package. Here the main article is always extended by a rather lengthy discussion thread of slashdot users. We will consider the discussion not to be part of the main content.

A. Fixing the Parameters

The content code blurring algorithms have two main parameters: the range of the neighbourhood and the threshold value for the extraction. The range defines the direct influence of the atomic elements on their neighbourhood; the threshold provides the minimum content code ratio an element has to satisfy for being declared part of the main content. To find good settings for these parameters we evaluated the performance of the CCB, ACCB and TCCB manually on a small set of documents.

It turned out quite soon, that a threshold of 0.75 is a good setting for all neighbourhood ranges and all variations of the algorithm. Keeping the threshold fixed allowed an easy exploration of settings for the range. Here a range of 40 turned out to be a good choice for the character based algorithms CCB and ACCB, while for the token based TCCB a range setting of 25 is the best choice.

B. Results

The average F1 results for extracting the main content from the documents are shown in table II. The table also includes the performance of the DSC algorithm as the best algorithm in our last comparison. The alternative of not using any CE is listed under the “plain” method and forms the baseline. Each CE algorithm should perform better than not using CE at all.

The results of the content code blurring algorithms are generally quite good. First of all we can notice that – with the exception of the wiki pages – all versions of CCB are achieving better results than the plain method baseline. This result qualifies CCB as a valid CE method. The second important insight is, that ACCB does not show a significant drawback in comparison to CCB. So, the adaptation of ignoring hyperlink tags during the construction of the content code vector does not cause a drop in the performance of the content code blurring approach. We can deduce that ACCB – though also improving the F1 performance on the wiki package significantly – is not overfitted for Wikipedia documents. ACCB is actually performing better for some of the other packages as well. So, the adaptation, which was specifically introduced for the particular case of main contents with a high ratio of in-text links is also useful for other scenarios.

The next good news is the performance of ACCB in comparison to DSC. While for the original, character based CCB and for the token based TCCB the performance does not show clear advantages or disadvantages, ACCB in general performs better than DSC. Looking at the total of our 14 evaluation scenarios, ACCB is scoring considerably higher F1 scores than DSC for five packages, comparable results on six packages and worse results only for three packages. Further, on four of the comparable packages ACCB is having slightly higher F1 scores, which might underline the tendency of a better performance. Among the packages, where ACCB is inferior to DSC is the generally problematic slashdot package. Though ACCB is achieving better recall values for slashdot documents, it is much less precise. The problem of the short main content and long additional text contents of the user discussions seems to affect the character based content code blurring stronger than DSC. Interesting is, that on the same package, DSC itself is coming second to TCCB when considering the F1 performance. This is also the case for the spiegel package, where ACCB is outperformed by DSC, but TCCB is still better than DSC. So, the question is, if in these cases it is solely the character based approach with has exceptional drawbacks in comparison to a token based approach. This might be a hint, that a more sophisticated construction of the content code vector could improve the results of the content code blurring idea. A solution somewhere between the character and token based construction of the vector might deliver still better results.

When looking in more detail at the packages where ACCB is performing better than DSC, it becomes obvious that the secret of ACCB’s success is a better recall. It is usually slightly less precise than DSC, but achieves better recall values. Accordingly, in the light of F1 it reaches a better tradeoff between recall and precision. This also explains the observations for the slashdot package. ACCB extracts the user comments better and more complete than DSC. But as they are not considered to be part of the main content, ACCB is punished for this extraction in the precision measure.

However, ACCB’s recall is not perfect either. If parts of the main content are highly formatted, they might not be recognised as main content. Especially if they are positioned close to other highly structured additional contents, the situation is very difficult for all content code blurring implementations.

TABLE II

EVALUATION RESULTS OF NEW SINGLE DOCUMENT ALGORITHMS: AVERAGE F1.

	bbc	chip	economist	espresso	golem	heise	manual	repubblica	slashdot	spiegel	telepolis	wiki	yahoo	zdf
accb-40	0.924	0.703	0.890	0.875	0.959	0.916	0.419	0.968	0.177	0.861	0.908	0.682	0.732	0.929
ccb-40	0.923	0.716	0.914	0.876	0.939	0.841	0.420	0.964	0.160	0.858	0.913	0.403	0.742	0.929
tccb-25	0.914	0.842	0.903	0.871	0.947	0.821	0.404	0.918	0.269	0.910	0.902	0.660	0.758	0.745
dsc	0.937	0.708	0.881	0.862	0.958	0.877	0.403	0.925	0.252	0.902	0.859	0.594	0.780	0.847
plain	0.595	0.173	0.613	0.624	0.502	0.575	0.371	0.704	0.106	0.549	0.858	0.823	0.582	0.514

TABLE III

AVERAGE NORMALISED PROCESSING TIME IN S/KB

	bbc	chip	economist	espresso	golem	heise	manual	repubblica	slashdot	spiegel	telepolis	wiki	yahoo	zdf
accb-r40	0.001	0.008	0.015	0.016	0.009	0.012	0.020	0.014	0.013	0.015	0.052	0.028	0.013	0.001
ccb-r40	0.007	0.008	0.015	0.016	0.009	0.011	0.018	0.011	0.013	0.016	0.052	0.024	0.013	0.001
tccb-25	0.001	0.001	0.002	0.002	0.001	0.001	0.001	0.001	0.001	0.001	0.004	0.003	0.001	0.001
crunch	0.027	0.012	0.014	0.033	0.033	0.032	0.047	0.048	0.019	0.018	0.077	0.073	0.024	0.028
dsc	0.001	0.001	0.002	0.002	0.001	0.001	0.001	0.001	0.001	0.001	0.003	0.002	0.001	0.001
lqf-50	0.004	0.003	0.016	0.010	0.004	0.011	0.017	0.001	0.009	0.009	0.051	0.013	0.011	0.001

This phenomenon can typically be observed for the headlines of news articles. They usually follow a section of additional contents, have a short text themselves and are separated from the rest of the main content by some other formatting instructions. However, DSC suffers from the same problem, even to a higher degree. ACCB's improvement in recall is also due to a better performance exactly in these tricky cases. The flexibly determined regions seem to be more appropriate in comparison to DSC's windowing approach.

When looking at the processing time in table III, the token based approaches of DSC and TCCB are comparably fast. ACCB and CCB need longer to process a document, simply due to the much longer content code vector. However, ACCB and CCB are both reasonably fast. Their time performance is similar to the LQF filters and still faster than the Crunch framework, which we have listed in this table as well.

V. CONCLUSIONS AND FUTURE WORK

We presented content code blurring, a Content Extraction algorithm which can be based either on characters or tokens. Comparing CCB with other CE algorithms we have shown that on most documents it yields better results than DSC. Especially on the difficult Wikipedia documents the adapted version ACCB is clearly superior to DSC. The results in absolute terms could still be improved, though.

Future works on CCB will comprise fine tuning on the parameters, combining it with other methods to achieve still better results and to incorporate some notion of the DOM block elements to enhance recognition of structurally related parts of a document. Further, we will do some experiments with more sophisticated models for constructing the content code vector.

REFERENCES

[1] S. Debnath, P. Mitra, and C. L. Giles. Identifying content blocks from web documents. In *Foundations of Intelligent Systems*, Lecture Notes in Computer Science, pages 285–293, 2005.

[2] A. Finn, N. Kushmerick, and B. Smyth. Fact or fiction: Content classification for digital libraries. In *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*, 2001.

[3] D. Gibson, K. Punera, and A. Tomkins. The volume and evolution of web page templates. In *WWW '05: Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*, pages 830–839, New York, NY, USA, 2005. ACM Press.

[4] T. Gottron. Evaluating content extraction on HTML documents. In *ITA '07: Proceedings of the 2nd International Conference on Internet Technologies and Applications*, pages 123–132, Sept. 2007.

[5] S. Gupta, H. Becker, G. Kaiser, and S. Stolfo. Verifying genre-based clustering approach to content extraction. In *WWW '06: Proceedings of the 15th International Conference on World Wide Web*, pages 875–876, New York, NY, USA, 2006. ACM Press.

[6] S. Gupta, G. Kaiser, D. Neistadt, and P. Grimm. DOM-based content extraction of HTML documents. In *WWW '03: Proceedings of the 12th International Conference on World Wide Web*, pages 207–214, New York, NY, USA, 2003. ACM Press.

[7] S. Gupta, G. Kaiser, and S. Stolfo. Extracting context to improve accuracy for HTML content extraction. In *WWW '05: Special Interest Tracks and Posters of the 14th International conference on World Wide Web*, pages 1114–1115, New York, NY, USA, 2005. ACM Press.

[8] 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.

[9] C. Mantratzis, M. Orgun, and S. Cassidy. Separating xhtml content from navigation clutter using dom-structure block analysis. In *HYPERTEXT '05: Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, pages 145–147, New York, NY, USA, 2005. ACM Press.

[10] D. Pinto, M. Branstein, R. Coleman, W. B. Croft, M. King, W. Li, and X. Wei. Quasm: a system for question answering using semi-structured data. In *JCDL '02: Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 46–55, New York, NY, USA, 2002. ACM Press.

[11] A. F. R. Rahman, H. Alam, and R. Hartono. Content extraction from html documents. In *WDA 2001: Proceedings of the First International Workshop on Web Document Analysis*, pages 7–10, 2001.