World Wide Web (2016) 19:1077–1101
DOI 10.1007/s11280-015-0374-9

Following the dynamic block on the Web

Sha Hu[1,2] · Ji-Rong Wen[1,2] · Zhicheng Dou[1,2] ·
Shuo Shang[3]

Received: 17 May 2014 / Revised: 30 August 2015 /
Accepted: 1 November 2015 / Published online: 18 November 2015

Abstract With the rapid changes in dynamic Web pages, there is an increasing need for receiving instant updates for dynamic blocks on the Web. In this paper, we address the problem of automatically following dynamic blocks in Web pages. Given a user-specified block on a Web page, we continuously track the content of the block and report the updates in real time. This service can bring obvious benefits to users, such as the ability to track top-ten breaking news on CNN, the prices of iPhones on Amazon, or NBA game scores. We study 3,346 human labeled blocks from 1,127 pages, and analyze the effectiveness of four types of patterns, namely visual area, DOM tree path, inner content and close context, for tracking content blocks. Because of frequent Web page changes, we find that the initial patterns generated on the original page could be invalidated over time, leading to the failure of extracting correct blocks. According to our observations, we combine different patterns to improve the accuracy and stability of block extractions. Moreover, we propose an adaptive model that adapts each pattern individually and adjusts pattern weights for an improved combination. The experimental results show that the proposed models outperform existing approaches, with the adaptive model performing the best.

Keywords Dynamic block · Wrapper · Block tracking

✉ Sha Hu
sallyshahu@gmail.com

Ji-Rong Wen
jirong.wen@gmail.com

Zhicheng Dou
dou@ruc.edu.cn

Shuo Shang
jedi.shang@gmail.com

[1] Beijing Key Laboratory of Big Data Management and Analysis Methods, Beijing, China

[2] School of Information, Renmin University of China, Beijing, China

[3] China University of Petroleum, Baijing, China

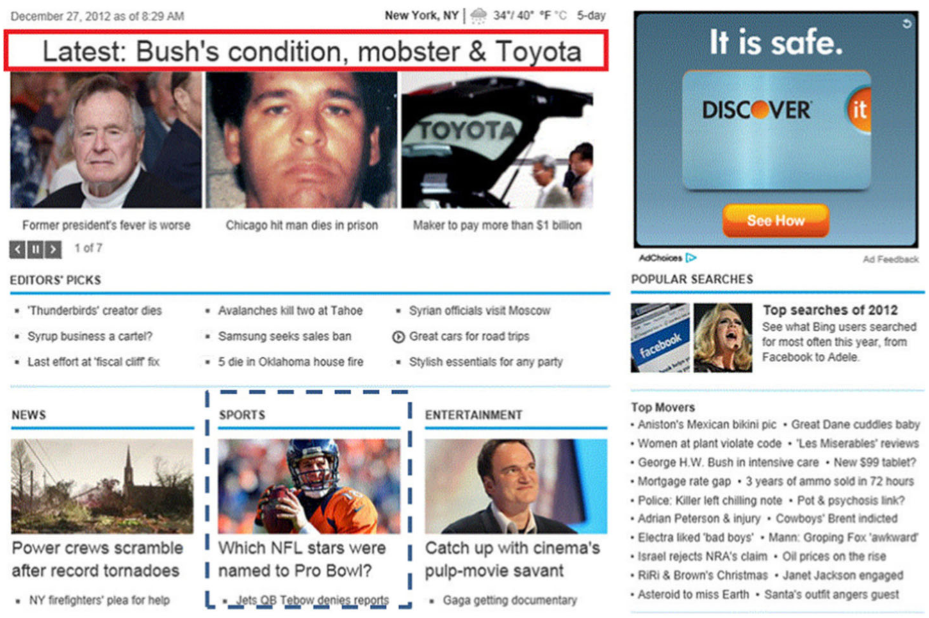
1 Introduction

The Web has become much more dynamic in recent years. To obtain instant information, some websites provide the "follow" mechanism to help users keep track of specific information blocks (e.g., following someone's status in a social network or using a weather app on a mobile phone to track city weather). Currently, only a few websites provide this follow function to users, which is normally provided by the websites themselves. The type and range of information blocks that can be followed are decided and controlled by the applications, not by users. For the majority of dynamic blocks on the Web, no following methods are available: users must revisit the web site to receive the updates.

In this paper, we propose a general approach to help users follow any dynamic block they are interested in, from any Web page. A user simply needs to specify the block within a Web page, then our system will regularly revisit the page, extract the content in the specified block, and return the extracted content to the user. Users have the freedom to decide the block and the granularity of what they want to follow. For example, they can track top-ten breaking news from CNN.com, product prices, citation numbers for her or his academic publications, or anything else they are interested in. A tracking example is shown in Figure 1. As an example, a news editor waiting to track the latest headlines on MSN.com, clicks a block containing a headline about "Bush's condition" (marked with a red border) on the homepage. The system begins to trace this headline block, and not the track topical contents about "Bush." Later, the system extracts this block with an updated headline, which is about the death of an American General, from a new version of the page, and returns it to the user. Another example shown in Figure 1 is a sports column (marked with a blue border), for a sports fan to get the latest sports news.

One of the key problems behind the system, which is also the focus of this paper, is how to identify the corresponding block in a evolved pages after the user has marked a block on the original page. A relevant and well-studied technique to this problem is wrapper induction [18–20, 22, 27–29]. Wrapper induction uses supervised learning to learn data extraction rules from manually labeled training examples. However, most existing wrapper induction algorithms cannot work well in our problem setting. They typically require multiple human labels to train a reliable model, whereas here there is only one training example (i.e., the block marked on the original page) available for tracking a specific block. An even bigger challenge is that the content or structure of Web pages keeps changing over time, causing the original wrapper to quickly become invalidated and fail to work on subsequent page versions. For example, the region containing the headline block in Figure 1 has significantly changed. The original headline block in Figure 1a appeared in the center of the top news area, and was above three news blocks; whereas the updated headline in Figure 1b has moved to the right of a dominant picture. Studies [3, 10, 16, 25, 26] have shown that a large portion of active pages change more than hourly, either in content or structure. Furthermore, because of the diversity of Web pages, the original page could be updated using various styles. The rapid and uncertain changes in Web page structure and content make it difficult to consistently track a block over a long time.

Several studies examine the block (or content) tracking problem [1, 4, 5, 14–16, 24]. However, their targets of study are the automation of Web browsing or the automatic creation of personalized portals, homepages, or start pages. Commonly used patterns in these works include the visual layout [5, 15, 24], page structure [4, 14, 16], and text similarities [1, 16]. To better understand whether these existing patterns are effective for the dynamic block following problem, we collect two data sets including 3,346 random labeled blocks from 1,127 URLs. Inspired by the existing literature, we extend or design four types of



December 27, 2012 as of 8:29 AM

New York, NY | 34°/ 40° °F °C 5-day

Latest: Bush's condition, mobster & Toyota

Former president's fever is worse

Chicago hit man dies in prison

Maker to pay more than $1 billion

1 of 7

EDITORS' PICKS

- 'Thunderbirds' creator dies
- Syrup business a cartel?
- Last effort at 'fiscal cliff' fix
- Avalanches kill two at Tahoe
- Samsung seeks sales ban
- 5 die in Oklahoma house fire
- Syrian officials visit Moscow
- Great cars for road trips
- Stylish essentials for any party

NEWS

Power crews scramble after record tornadoes

- NY firefighters' plea for help

SPORTS

Which NFL stars were named to Pro Bowl?

- Jets QB Tebow denies reports

ENTERTAINMENT

Catch up with cinema's pulp-movie savant

- Gaga getting documentary

POPULAR SEARCHES

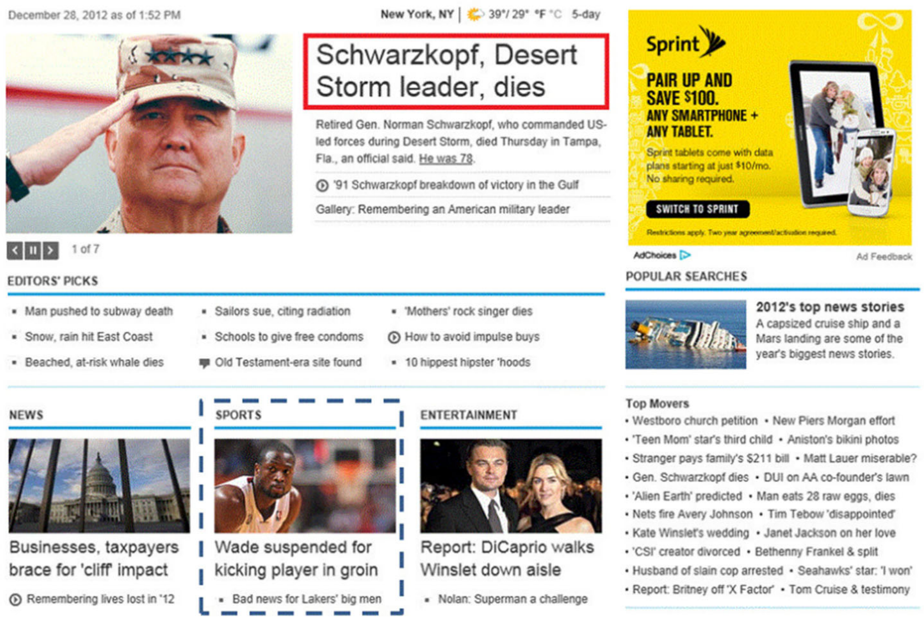
Top searches of 2012

See what Bing users searched for most often this year, from Facebook to Adele.

Top Movers

- Aniston's Mexican bikini pic • Great Dane cuddles baby
- Women at plant violate code • 'Les Miserables' reviews
- George H.W. Bush in intensive care • New $99 tablet?
- Mortgage rate gap • 3 years of ammo sold in 72 hours
- Police: Killer left chilling note • Pot & psychosis link?
- Adrian Peterson & injury • Cowboys' Brent indicted
- Electra liked 'bad boys' • Mann: Groping Fox 'awkward'
- Israel rejects NRA's claim • Oil prices on the rise
- RiRi & Brown's Christmas • Janet Jackson engaged
- Asteroid to miss Earth • Santa's outfit angers guest

(a) the original version



December 28, 2012 as of 1:52 PM

New York, NY | 39°/ 29° °F °C 5-day

Schwarzkopf, Desert Storm leader, dies

Retired Gen. Norman Schwarzkopf, who commanded US-led forces during Desert Storm, died Thursday in Tampa, Fla., an official said. He was 78.

'91 Schwarzkopf breakdown of victory in the Gulf

Gallery: Remembering an American military leader

1 of 7

EDITORS' PICKS

- Man pushed to subway death
- Snow, rain hit East Coast
- Beached, at-risk whale dies
- Sailors sue, citing radiation
- Schools to give free condoms
- Old Testament-era site found
- 'Mothers' rock singer dies
- How to avoid impulse buys
- 10 hippest hipster 'hoods

NEWS

Businesses, taxpayers brace for 'cliff' impact

Remembering lives lost in '12

SPORTS

Wade suspended for kicking player in groin

- Bad news for Lakers' big men

ENTERTAINMENT

Report: DiCaprio walks Winslet down aisle

- Nolan: Superman a challenge

Sprint
PAIR UP AND SAVE $100.
ANY SMARTPHONE + ANY TABLET.
Sprint tablets come with data plans starting at just $10/mo. No sharing required.
SWITCH TO SPRINT
Restrictions apply. Two year agreement/activation required.
AdChoices
Ad Feedback

POPULAR SEARCHES

2012's top news stories

A capsized cruise ship and a Mars landing are some of the year's biggest news stories.

Top Movers

- Westboro church petition • New Piers Morgan effort
- 'Teen Mom' star's third child • Aniston's bikini photos
- Stranger pays family's $211 bill • Matt Lauer miserable?
- Gen. Schwarzkopf dies • DUI on AA co-founder's lawn
- 'Alien Earth' predicted • Man eats 28 raw eggs, dies
- Nets fire Avery Johnson • Tim Tebow 'disappointed'
- Kate Winslet's wedding • Janet Jackson on her love
- 'CSI' creator divorced • Bethenny Frankel & split
- Husband of slain cop arrested • Seahawks' star: 'I won'
- Report: Britney off 'X Factor' • Tom Cruise & testimony

(b) a new version

Figure 1 Tracking the headline and sports news on MSN.com

patterns namely visual area, *Dom* tree path, inner content, and close context, and study their effectiveness. We find that each individual pattern extracted from an original labeled block could be invalidated over time, because of Web page updates. None of the patterns could consistently extract correct blocks, and their probabilities of extracting correct blocks decreased over time. On the basis of these observations, we propose a *combined pattern model* (CPM) to integrate four patterns for block tracking. Experimental results show that the combined model does outperform existing approaches.

The initial patterns may become unreliable over time, and eventually hurt the accuracy of the combined model. We must adapt the patterns so that they can still work in future page versions. In CPM, we treat the four patterns as equally critical, although we find that different pages have different layouts and change differently. A pattern that works well on one page for a long period of time, may quickly fail on other pages or blocks. This indicates that different patterns should have different weights for tracking different blocks, and even for tracking the same block within different time ranges. We must adapt pattern weights based on the historical change patterns of the block, assigning higher weights to patterns that are more stable and reliable. Therefore, we develop an *adaptive pattern model* (APM) that adapts patterns based on recent tracking results. When a block is extracted from the newest version of a page, we add it as a positive sample to individual patterns. We also adjust the weights of the patterns, so that the patterns that help identify the correct blocks for this page over time can yield higher weights. We then use the adapted patterns, combined with adjusted weights, to process the upcoming versions of the page. Experimental results show that this adaptive model further improves the extraction accuracy over CPM.

The major contributions of this paper are as follows:

- We give an exclusive experimental study on investigating the effectiveness of four typical types of patterns that are used for following or tracking blocks on the Web. Experimental results show that each pattern cannot consistently extract correct blocks on their own, but their combinations have higher potential to work better.
- We propose two models, namely the CPM to go beyond the sole use of each individual pattern, and the APM to automatically adapt extraction on the basis of one user-defined instance.
- The experimental results show that our proposed combination models outperform existing approaches. The APM is superior because it adapts patterns with the changes in Web pages.

The rest of the paper is organized as follows. Section 2 discusses related work, and Section 3 introduces the problem and datasets. Single pattern approaches and combined approaches are detailed in Section 4 and Section 5. After a report of experimental results in Section 6, this paper is concluded in Section 7.

2 Related work

Many studies analyze the changes in Web pages. Cho and Garicia-Molina [9] follow 720,000 pages over four months. Comparing different page versions by MD5, they find that over 40 % of pages change in a week and 23 % of domains change daily. Fetterly et al. [13] extend Cho's study and find that larger pages change more frequently and extensively than smaller ones, and that past changes could be a good predictor of future changes. More recently, Adar et al. [3] crawl 55,000 pages hourly for five weeks and find that a large portion of dynamic pages change hourly or less. They show that there is a strong connection between Web dynamics and revisitation [2]. All of these studies emphasize that Web dynamics is pervasive in the modern Web, so it is valuable to provide a way for users to track the changes in Web pages.

Many approaches compare or recognize Web page changes. ChangeDetector [6] monitors an entire Web site to discover new information by using page classifications and entity detection. HtmlDiff [12] detects changes by using HTML attributes. WebCQ [21] notifies users of page changes in a personalized way. Both HtmlDiff and WebCQ highly depend on the stability of HTML attributes, which no longer hold well for the current Web environment. DiffIE [26] is a browser plug-in that highlights the changes in a page since the last visit on the basis of page caches. In contrast to these approaches that focus on page-level changes, this study focuses on tracking the changes of a block.

As discussed in Section 1, the dynamic block-tracking problem is relevant to wrapper induction, a well-studied problem in the information extraction area. Many existing wrapper techniques use multiple patterns to extract specific records from similar pages (e.g., specific product information from product pages). For example, ViNT [29] uses both visual rectangles and structure tags in data extraction. Liu and Zhai [20, 27] extract data records on the basis of tree mapping and visual information. They try to extract items from detailed pages by using a few labeled instances [28]. Muslea et al. [22] also use a few labeled examples in extraction. Dontcheva et al. [11] generate patterns to collect content from similar pages. In the current study, our task is to track one specific block (not a type of blocks) on a specific page (not similar pages) over time, which is a different problem from wrapper induction. Most wrapper techniques do not work well in our task because they require multiple samples, which means that the user must mark the block in multiple versions of the page. In fact, the system must start tracking once a block is labeled by the user, which means that only one labeled example is provided.

Some studies and applications extract or track blocks within the pages. For instance, Internet Scrapbook [24] and Smart Bookmarks [17] discuss the problem of automating Web browsing task, which aims to automatically re-play the browsing behaviors of a user after recording the user's interaction history. The relevance to our work is that they try to find the recorded elements in the pages being replayed. By assuming that Web pages do not change significantly, they use brief patterns (e.g., the XPath[1] of HTML elements within the DOM tree) to match elements. Another branch of relevant work concerns the automatic creation of personalized portals, homepages, or start pages [5, 14, 16]. Montage [5] uses visual information, including size and position, to extract blocks from pages and display them in users' personalized start pages. WebViews [14] takes XPath to accomplish similar tasks. HomepageLive [16] leverages a modified tree edit distance algorithm to measure the similarities between blocks and pages, and then uses the similarities to rank blocks. EShopMonitor [4] monitors Web content by its XPath, including tiny price tracking. Overall, these applications do not focus on the reliability of the tracking algorithm; most use a single pattern, such as visual layout [5, 15, 24] or page structure [4, 14, 16]. In this paper, we extend these patterns and propose four typical patterns, evaluate their effectiveness, and explore solutions that can more reliably track blocks. Zoetrope [1] studies three types of patterns for tracking blocks: visual patterns, structural patterns, and textual patterns. It asks users to decide which type of "lens" (pattern) to use. Instead, we propose to combine these patterns and automatically weight them according to the tracking history. Furthermore, we propose to adapt the patterns to accommodate page changes, which is rarely mentioned in block tracking.

[1] www.w3schools.com/xpath

Some prior work exists on segmenting a Web page into blocks. One popular page segmentation tool is VIPS [7], which uses an automatic top-down, tag-tree independent approach to detect Web content structure. Block-based Web search [8] investigates how to take advantage of segmented blocks to improve retrieval performance. As a clarification, a block is a simple HTML element selected by a user in this paper, not the segmented block studied by these approaches.

3 Preliminaries

3.1 Problem description

The problem of following (or tracking) blocks can be described as follows. When a user finds an interesting block on a Web page and wants to track its changes over time, she or he marks the block by using a simple click (or other operations that are specified by applications). The system receives the request and generates a pattern based on the original page and the user's selection, to extract the block in new versions of the same page. Then the system regularly downloads the updated page of the same URL, extracts and ranks candidate blocks, and returns the most probable block to the user. The system, under different implementations, may display the entire history when the user proactively views the block, or else notify the user when new content is extracted for the block. The extraction can be executed either on the user's client side, or on a centralized server. The design of a specific block-tracking system exceeds the scope of this paper; we focus on the design of the algorithms for reliable block tracking.

Note that a block is restricted to be a DOM element of a page in this paper.

3.2 Data

To evaluate the effectiveness of existing block-tracking algorithms, we generate two datasets. The dataset statistics are listed in Table 1.

We crawled the homepages of the top 100 frequently visited websites ranked by Alexa.com, and extracted their intra-domain outlink targets (intra-domain means the link targets are also on the same website). From these target URLs, we manually selected 69 pages that are quickly updated and potentially revisited. We crawled the pages every 2 hours and collected 43 versions in 4 days. We named this dataset Data1. Data1 includes the frequently updated and revisited pages that users mostly like to follow. This dataset shows block evolutions in hours.

We then generated a new set of 1,000 randomly selected URLs from these outlink targets, and the top 500 most frequently visited URLs from a one-day query log of a commercial search engine. After removing the pages containing only a few words (e.g., google.com) or requiring log in (e.g., facebook.com), we obtained 1,058 URLs. To cover a longer time span,

Table 1 The dataset statistics

Name	Frequency	URLs	Page Versions	Labeled Blocks	Blocks Versions	Selected Mode
Data1	2 hours	69	43	178	7654	manually
Data2	8 hours	1058	20	3168	63360	randomly

we crawled these pages every 8 hours and collected 20 versions over a week. We named this dataset Data2. Data2 includes randomly selected pages from frequently visited websites and URLs that are widely used daily and have a higher probability of being followed. Note that besides the homepages of these websites, we also selected detailed pages that are linked by the homepages. This dataset shows block evolutions over a week. We did not collect pages for a longer time period (e.g., a month versions) because the follow-up of manually labeling blocks for so many versions was too expensive.

For each URL in Data1 or Data2, we asked a labeler to freely annotate three nonoverlapped blocks that she or he would like to follow. To cover different granularities, we encouraged the labelers to select three blocks of different ranges, including large and small blocks. For example, in Figure 1, three well-labeled blocks could be "sports column" (large block), "main headlines" (small block), and "New York weather column" (small block). The labeler annotates three blocks in the first version, and tracks and annotates corresponding blocks in later versions. A block is required to be tagged as *missing* in which it does not exist. A second labeler is then asked to double-check the annotation quality. A block is discarded if the second labeler does not agree with the annotation of any version for this block. On average, it takes three minutes to label the first version and one minute to label a subsequent version. Because collecting and labeling the data is quite costly, we plan to share the datasets with the public in the future.

In total, 178 blocks in Data1 and 3,168 blocks in Data2 are successfully labeled. By analyzing the chosen blocks in the two datasets, we find that: (a) All the chosen blocks in Data1 were unique, including 2 partially overlapped blocks, whereas Data2 had 2 duplicate blocks and 23 partially overlapped blocks. (b) In Data1, the average block size was 167,494 pixels, and 52 % of the blocks were larger than 100,000 pixels (e.g., "sports column" in Figure 1); hence, Data1 contained more "large blocks." In Data2, the average block size was 94,422 pixels and 53 % of the blocks were smaller than 50,000 pixels (e.g., "the main headlines" in Figure 1); hence, Data2 included more "small blocks." (c) The inner structures of blocks in Data1 had 131 nodes in average, whereas the average node number of blocks in Data2 was 43. This indicates that the blocks in Data1 were more complicated than those in Data2. (d) Regarding the visual locations of these blocks, 59 % of blocks in Data1 and 50 % of blocks in Data2 were in the left parts of pages, and 56 % of blocks in Data1 and 64 % of blocks in Data2 were in the upper parts of pages.

4 Simple pattern approaches

A pattern is used to locate and extract a specific block from a Web page. Different patterns are used in [1, 4, 5, 14–16, 24] as introduced in Section 2. We extend the existing works and observe that there are four patterns that individuals typically leverage to locate a block. We call them area pattern, path pattern, content pattern, and context pattern. Given a block, we first generate these patterns. When a new version of the page is published, we use these patterns to extract candidate blocks, rank them, and output the best block. In this section, we describe these patterns in detail, and then evaluate their effectiveness using the data described in Section 3.2.

4.1 Patterns

Area Pattern An area pattern is based on a rectangular area on a displayed page. It is obtained by the visual position of a block on a rendered page. If the page always displays a specific block in a fixed position, we can easily locate the block in the new versions by its positions. This visual layout is widely used in [1, 5, 15, 24], and works well for pages with a stable visual display. Formally, we define **Area Pattern** as $\boldsymbol{A}= (l, r, r-l, t, b, b-t)$, which specifies the left, right, width, top, bottom, and height coordinates of a block. Normally four components (l, r, t, b) are enough to represent a rectangle. We use six components because it is easier to describe the change in a block with six components instead of four. Figure 2 shows three common moving situations for blocks. In the figure, the solid and dashed boxes are the bounding rectangles of the original and changed version of the same element, respectively. In case 1, the block moves downwards, with left, right, width, and height remaining the same. This usually happens when a new node is inserted before the block. In case 2, the block has fixed top and right coordinates. In case 3, the block remains at the same position, but its height is increased. By analyzing the datasets, we find that 85 % of blocks have at least two fixed components during evolutions, while 55 % of blocks have an area overlap.

Path Pattern For a block with a fixed path in a DOM tree, it is reasonable to use the path for its locate. The path has been widely studied and used in existing works [1, 4, 14, 26]. In this paper, we define **Path Pattern** as $\boldsymbol{P}= T_1[i_1]/T_2[i_2]/\ .../T_n[i_n]$, which is a simplified XPath format. T_k represents the tag name of the block's corresponding DOM element at level k and i_k is the node's index (starting from 1) within all the sibling nodes with the same tag. Note that i_k of a node only changes when nodes with the same tag are inserted or removed before the node. Figure 3 shows two common situations of path evolution over time. In version 2, a sibling div node is removed and i_k of the second part of the path changes. In version 3, a new node is inserted as an ancestor of the block, which causes the insertion of a new part into the path.

Content Pattern In addition to the area and path patterns, the contained and surrounding text are also effective signals for identifying a block. For example, to track the sports news on MSN.com as shown in Figure 1, the content word SPORTS is a very strong pattern to locate the block in the blue dotted frame, even if the SPORTS and ENTERTAINMENT blocks swap their positions (both area and path patterns change in this case). The textual lenses of Zoetrope [1], which measures text similarity, is a type of content pattern. The edit distance of HomepageLive [16] also partially considers content similarity. In this paper, we propose a brief presentation of content pattern: a list of leaf text or image DOM elements contained in the block. Formally, a **Content Pattern** is defined as $\boldsymbol{N}= \{E\}$, where $E = \langle text, tpath\rangle$, $text$ is the text or image source URL of a leaf element, and $tpath$ is the inner tag path starting from the block root to the element (see N in Example 1).

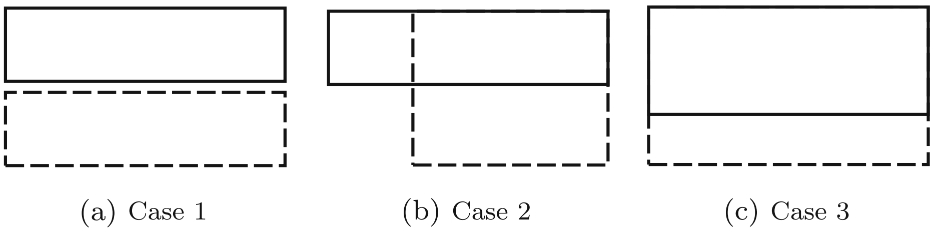


Figure 2 Examples of moving blocks. Solid and dashed boxes are bounding area of the original and changed versions of a block

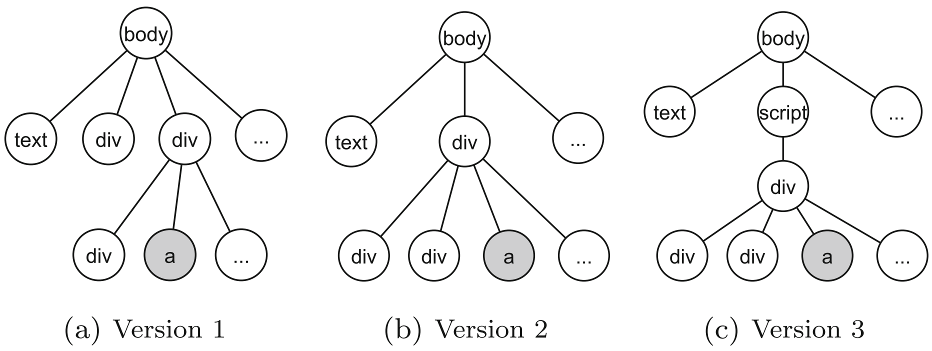


Figure 3 An example of the path pattern and its evolution. Shade node "a" is the targeted content block, Version 1, 2, 3 show its DOM trees at different time. $P_1 = body[1]/div[2]/a[1]$, $P_2 = body[1]/div[1]/a[1]$, $P_3 = body[1]/script[1]/div[1]/a[1]$

Context Pattern The content pattern will fail to track a block when most of the block content changes. In these cases, the context information can play a crucial role in tracking. For the tracking headline example in Figure 1, the new headline content is completely changed, but we find that its context remains relatively stable: the headline is always under the weather block and above the EDITORS' PICKS block. Using context to locate information is a very common practice for people. Moreover, when users want to follow small blocks, such as the price of a product, it is unreliable to use the price number for tracking because it frequently changes; however, some information close to the price, such as the product name, logo, or producer, is quite stable. Therefore, we introduce the **Context Pattern** denoted as X. X includes a set of surrounding text or image elements $\{E\}$, where $E = text$, and the relative visual distance from E to the block is less than $maxdistance$; $text$ is the text or image source URL of the element, and the threshold $maxdistance$ is used to select a close context (set to 50 pixels in our experiments).

In summary, Example 1 shows an example of the four patterns for the headline block mentioned in Figure 1a.

Example 1 Patterns for tracking the headline block in Figure 1

$A = (11, 650, 639, 263, 302, 39)$
$P = body[1]/div[1]/div[2]/span[1]/a[1]$
$N = \{\langle$"Latest:Bush's condition...", $/a\rangle\}$
$X = \{$"December 27...","New York,NY","34^o...", ...$\}$

4.2 Candidate generation and ranking

As described in the introduction to Section 4, the generated patterns are used to extract candidate blocks from new versions of Web pages. These candidates are further ranked and the best possible candidate is selected as the final output.

For the area pattern, we extract the first block with the exact area as a candidate and directly output it as the final output. We do the same for the path pattern. Because our target is to track the change in the block, in most cases, the content contained in or around the block will partially change. This means that we cannot use an exact match to extract candidates for content and context patterns. We use a partial match instead and select the best matched candidate later. For the content pattern, an element is extracted as a candidate if it has at least one text or image descendant overlapping with the elements in pattern N (both text and tpath are matched). For the context pattern, a candidate should have at least one text or image neighboring element found in pattern X (identical text).

After candidate blocks are extracted, the optimal block is determined. Because the area and path patterns already have their perfect matches and do not need this step, we introduce their ranking functions for later use.

Supposing Can is a candidate block, we use $Can.A$, $Can.P$, $Can.N$, and $Can.X$ to stand for its visual area, DOM tree path, a set of text or image descendant elements, and a set of neighboring text or image elements, respectively. For each pattern A, P, N, or X, the similarity between a candidate Can and the pattern is calculated as follows.

Area Pattern Similarity. Sim_a represents the similarity between $Can.A$ and A. Traditional similarity measures for two visual areas, such as the ratio of overlapping area and the Jaccard coefficient similarity for the location (l, r, t, b), may not work well in block tracking. For example, considering the typical case in which the block moves downward in Figure 2a, the shifted block has no overlap with the original one and it cannot be found by using the overlap ratio method. When using the Jaccard similarity of (l, r, t, b), the left and right edges remain stable and thus the similarity is $Sim_a = 2/4 = 0.5$; however, the shifted block should be valued more because it also shares the same width and height of the original block.

Recalling that our area pattern is denoted by six components $(l, r, r - l, t, b, b - t)$, we leverage them to measure the visual area similarity. In particular, we treat the X axis $(l, r, r - l)$ and the Y axis $(t, b, b - t)$ separately. We calculate a match score for each axis based on the overlap between the components. If an axis has more than two components overlapping with the pattern, then the score is two; the score is one if only one component is the same as that in pattern A; otherwise, the score is zero. The similarity between the candidate and the pattern is the normalized sum of the two axes, i.e., $Sim_a = (score(X) + score(Y))/4$. The similarities for the three cases in Figure 2 are $(2 + 1)/4 = 0.75$, $(1 + 1)/4 = 0.5$, and $(2 + 1)/4 = 0.75$.

In Case 3 of Figure 2, the candidate has the same matched components as Case 1, but visually it has a larger overlap with the pattern. In such a case, we additionally calculate the overlap area ratio and choose this value if it is larger than the previous calculation. This overlap area ratio is only used when one area is completely inside the other.

Path Pattern Similarity. Sim_p represents the similarity between $Can.P$ and P. Because the path pattern is organized by the DOM tree tags and indexs $T_1[i_1]/T_2[i_2]/ .../T_n[i_n]$, we follow the basic idea for the Jaccard similarity and measure the path similarity by using the ratio of matched tags and indexes [1]. Here, an equal index pair is treated as a match if and only if their corresponding tag pairs are identical. However, traditional path mapping commonly compares nodes from root to leaf (top-down), which may not work well in block tracking. Consider the situation that a new node is inserted as an ancestor to the original block as shown in Figure 3b to 3c: $P_2 = body[1]/div[1]/a[a]$ becomes $P_3 = body[1]/script[1]/div[1]/a[a]$. The top-down mapping piece of P_2 and P_3 contains only one node, $body[1]$, whereas their common tail piece $div[1]/a[a]$ shows an improved similarity. Therefore, to leverage better matched piece, we count the matched score twice from both top-down (td) and bottom-up (bu) directions. We use the larger score to calculate its ratio, and take the longer path as the base: $Sim_p = \frac{max(score_{td}, score_{bu})}{max(|Can.P|, |P|)}$. For instance, given $Can.P = body[1]/div[1]/div[1]/div[2]/span[1]/a[1]$ and $P =$

$body[1]/div[1]/div[2]/span[1]/a[1]$, we have $score_{td} = 2+2+1+0+0 = 5$, $score_{bu} = 2+2+2+2+0 = 8$, and get $Sim_p = max(5,8)/max(10,12) = 0.67$.

Content Pattern Similarity. Sim_n represents the similarity between $Can.N$ and N. Because the content pattern is formed as sets of text and tpaths $\{E\langle text, tpath\rangle\}$, an element is identified as a match if and only if its text and tpath are both matched. To describe how well $Can.N$ matches N, this similarity is calculated as the ratio of matched elements in N, $Sim_n = \frac{|Can_n\langle text,tpath\rangle \bigcap N\langle text,tpath\rangle|}{|N\langle text,tpath\rangle|}$. If $|N|$=0, then $Sim_n = 0$. If $Can.N$ contains all the elements of N, then $Sim_n = 1$. For example, $N = \{\langle$"SPORTS", $/a\rangle$, $\langle$"Which NFL...", $/div\rangle$, $\langle$"More...", $/div/a\rangle\}$, and $Can.N = \{\langle$"SPORTS", $/a\rangle$, $\langle$"Wade...",$/div\rangle$, $\langle$"More...", $/div/div/a\rangle$, then $Sim_n = 1/3 = 0.33$. We do not use Jaccard coefficient, $\frac{|Can.N \bigcap N|}{|Can.N \bigcup N|}$, because when the intersection is the same, it may value a candidate with a less different content to N as superior to our target block, which contains more updated (and hence, different) content.

Context Pattern Similarity. Sim_x represents the similarity between $Can.X$ and X. For the context pattern $\{text\}$, the similarity could be calculated according to value and direction. Dynamic changes may make the direction unreliable; therefore, we only identify a matched element by the equal text, and count the matched ratio in X like the content pattern, $Sim_x = \frac{|Can_x\langle text\rangle \bigcap X\langle text\rangle|}{|X\langle text\rangle|}$. If $|X| = 0$, then $Sim_x = 0$. For example, $X = \{$"December 27...", "New York, NY", "34°..."$\}$, and $Can.X = \{$"December 28...", "New York, NY", "39°..."$\rangle\}$, then $Sim_x = 1/3 = 0.33$. We do not consider Sim_x by using the Jaccard coefficient that contains the entire $Can.X$ in the base, because the mismatched context from $Can.X$ is unhelpful to locate the target and will negatively affect the similarity as was explained for Sim_n.

4.3 Effectiveness of the patterns

We evaluate the effectiveness of the patterns by calculating *recall on the candidate set* and *accuracy* on the final extracted blocks.

- **Recall on the candidate set** indicates the potential for extracting correct blocks depending on the pattern. We calculate *version-level* and *block-level recalls* separately. Version-level recall is the percentage of page versions in which we successfully determine the correct block from the candidates extracted by the pattern. Block-level recall is the percentage of pages in which we can correctly determine the candidates in all versions (i.e., the percentage of pages with version-level recalls equaling 1). We calculate block-level metrics because we want to know how likely we could perfectly track a block (always returning the correct blocks for a URL). Obviously, block-level recall is much stricter than version-level recall.
- **Accuracy** is the percentage of cases that the correct (human labeled) block can be finally outputted. This can be viewed as a special recall of our top-one result (recall@1). Note that the block can be finally extracted if and only if it is in the candidate set, so accuracy should not be larger than recall on the candidate set (i.e., recall on the candidate set is the upperbound of accuracy). We also calculate block-level and version-level accuracies separately.

We test the four patterns on the two datasets. The performance of each pattern is listed in Table 2. Table 2 shows that the area pattern performs the worst, in terms of either accuracy

Table 2 Single pattern results

Pattern	Recall of candidates				Accuracy			
	Data1		Data2		Data1		Data2	
	Block	Ver	Block	Ver	Block	Ver	Block	Ver
Area	0.303	0.615	0.307	0.585	0.281	0.577	0.295	0.567
Path	0.674	0.854	0.599	0.816	0.674	0.854	0.599	0.816
Content	0.787	0.859	0.716	0.837	0.444	0.543	0.366	0.478
Context	0.876	0.952	0.792	0.897	0.421	0.581	0.491	0.662

or recall on the candidate set. The area of a block is very likely to change, and we fail to find exact matched candidates on many versions. The path pattern, which also uses an exact match to extract candidates, yields a higher recall on the candidate set and a higher accuracy than the area pattern. This indicates that the DOM path of an element is relatively more stable than its block area.

The differences in recall and accuracy are notable. Area and path patterns extract only one block (see Section 4.2); hence, recall and accuracy should be the same. Table 2 shows that for the path this is indeed the case, but it is not so for the area. The reason for this is that the path pattern can always uniquely identify one block node for a page, but the area pattern cannot. For example, a targeted node ($body[1]/div[1]/a[1]$) and its child node ($body[1]/div[1]/a[1]/text[1]$) share the same visual area; thus, an area-matched node may be incorrect. For Area pattern, accuracy is a little lower than recall.

Table 2 also shows that Content and Context have much higher recalls than Area and Path do, but their accuracies are relatively lower than Path's. This is because the candidate extracting strategies for Content and Context are very tolerant; they generate many more candidates than Path and Area, and have a higher potential for finding the correct block. However, the results also indicate that the more candidates they find, the more noise is included. Distinguishing the correct block from all the candidates becomes a hard work for these two patterns.

Generally, we find that the results are far from optimal. Considering Data2 which includes more URLs, the best pattern (Path), is able to extract correct blocks for approximately 82 % of the extraction cases (versions), but can only return perfect results for approximately 60 % of pages.

From Table 2, we find block-level recall is much lower than version-level recall, for both Data1 and Data2. As discussed, block-level recall is much stricter than version-level recall, because block-level recall requires determining the correct candidates for all versions of a page.

Because each pattern is far from optimal, we further investigate whether the extraction accuracy can be improved by combining patterns. We calculate the recall on the candidate set of **combined patterns**. For a combined pattern of pattern A and pattern B, its recall candidate set is the **union** set of A's candidate set and B's candidate set. By analyzing the recall of combined patterns, we could determine whether we have a higher potential for finding correct blocks. Note that a combination of A and B does not guarantee a higher recall than the sole use of A or B. For example, for a version of a page, if both A and B fail to extract the correct candidates, or both of them include the correct block in their candidates, then the combination of the two will not improve recall on the candidate set (the recall will keep the same). The improvement of recall may occur when A and B are complementary (i.e., B finds the correct block in its candidates whereas A fails, or vice versa). The results are shown in Table 3, which shows that most combined patterns yield a higher recall on the candidate set than any sole use of each individual pattern. The combination of all patterns, namely APNX in Table 3, has very high recall values for both datasets. This means that our proposed patterns are complementary.

5 Combined pattern approaches

According to the results in Section 4, we find that each single pattern is far from optimal in block tracking. We also reveal that the combinations of these patterns could improve the recall on the candidate set, and hence could potentially improve the final extraction accuracy. In this section, we propose two combination models: a simple linear combination, and an adaptive combination model.

5.1 Combined pattern model (CPM)

For the CPM, we generate the same patterns as introduced in Section 4. We keep these patterns unchanged and use them for all coming versions. As discussed, we combine patterns to yield a higher extraction accuracy. For a new version, we use the four patterns to extract candidates independently and generate a set containing all these candidates. We then calculate a combined weight for each candidate in the set. We view all patterns as equally useful, and use a brief linear combination to calculate the overall similarity score,

$$Score = Sim_a + Sim_p + Sim_n + Sim_x \tag{1}$$

Recall that Sim_a, Sim_p, Sim_n, and Sim_x are the similarities between the candidate and each pattern, which were introduced in Section 4.2.

Table 3 Recall on the candidate set for combined patterns. The area, path, contet, and context patterns are respectively denoted as A, P, N, and X

	Data1		Data2	
Pattern	Block	Version	Block	Version
AP	0.736	0.905	0.646	0.855
AN	0.860	0.936	0.773	0.904
AX	0.910	0.972	0.838	0.937
PN	0.933	0.981	0.837	0.943
NX	0.949	0.992	0.886	0.951
PX	0.983	0.989	0.883	0.957
APN	0.938	0.982	0.858	0.956
ANX	0.961	0.996	0.907	0.968
APX	0.983	0.995	0.899	0.967
PNX	0.994	0.9991	0.927	0.976
APNX	0.994	0.9997	0.937	0.983

5.2 Adaptive pattern model (APM)

The CPM uses fixed patterns obtained from the initial labeled page. When a page evolves over time, the initial patterns may become invalid and then no matched candidates can be found after a period of time. By aggregating other patterns, there may be a chance to fix the failed patterns. The CPM uses fixed weights to combine the patterns. In investigating the data, we find that different content blocks have different changing styles. For example, some block paths are quite stable, but their positions constantly change. The CPM cannot adjust pattern weights to reflect the different changing styles. We propose an APM to solve this problem. The basic idea is to take the identified block in past versions as new labeled samples, and then combine these with the initial labeled sample to adapt the patterns and their weights. This is similar to the idea of pseudo-relevance feedback.

The process is described as follows. First, with the latest adapted patterns, we generate candidates, rank them, and output the optimal match for a new version of the page. Second, we add the new block into the historical sample set. Finally, we adapt patterns and their weights according to the historical samples. The number of kept historical samples can be set in different values. In this paper, we use all historical samples and treat each sample equally. We plan to investigate whether these historical samples are equally useful for adapting patterns (e.g., whether new samples are more useful than older samples) in a future work.

5.2.1 Pattern adaption

We want to adapt each pattern when a new block is extracted. In this section, we introduce an adaption method for each pattern.

Area Pattern Adaption. For a new sample, the area pattern does not usually completely overlapped with the original area. To describe a stable shape, we must keep the overlaps and handle the un-overlapping parts. Our strategy is to combine areas into a *vague* pattern: if the distance of one coordinate between the new area and the original area is within a threshold (set to 10 in our experiments), we view it as an overlap and use the mean as the new value for the coordinate; if not, the coordinate is assigned to empty (use $*$ to denote empty values in the samples) to indicate that this coordinate is not fixed. Note that the new adaptive area pattern contains empty values ($*$), which is not considered in Section 4.2. In APM, we extract and weight candidates by using only nonempty components in the pattern.

An example of area pattern adaption and weighting is shown as follows. Given $A_1 = (11, 650, 639, 263, 302, 39)$ and $A_2 = (418, 648, 230, 268, 351, 83)$, we get $A = (*, 649, *, 265, *, *)$. Thus, the block has fixed right and top coordinates, and any blocks whose area matches these two components will be extracted as a candidate. For $Can.A = (12, 651, 639, 364, 403, 39)$: there is one match in A's two nonempty coordinates, so we get $Sim_a = (1)/(2) = 0.5$.

Path Pattern Adaption. To merge the path pattern from the new sample into the original pattern, we use the longest common top (LCT) and longest common bottom (LCB) to describe the multi paths' common part. Parts of multi paths are only viewed as common if and only if they share identical tags. A common part adopts its mismatched indexes by using $*$. To avoid the overlap in LCT and LCB, we generate the common path twice by using different priorities: top-down (full LCT with the remainder LCB), or bottom-up (full LCB with the remainder LCT). We then select the path with a higher non-empty count of tags and indexes. Given $P_1 = body[1]/div[1]/div[2]/span[1]/a[1]$ and $P_2 = body[1]/div[1]/div[1]/div[2]/span[1]/a[2]$, we merge $LCT = body[1]/div[1]$ $/div[*]$ with $LCB = div[1]/div[2]/span[1]/a[*]$. We generate the path for top-down as $P_{td} = LCT + LTB_{rest} = body[1]/div[1]/div[*]/.../span[1]/a[*]$ and its nonempty components $(2 + 2 + 1) + (2 + 1) = 8$, and for bottom-up as $P_{bu} = LCT_{rest} + LCB = body[1]/.../div[1]/div[2]/span[1]/a[*]$ with a nonempty count of (2)+(2+2+2+1) = 9. So the adapted path pattern is $P = P_{bu} = body[1]/.../div[1]/div[2]/span[1]/a[*]$, where /.../ stands for any middle sub-path. Hence, P extends its candidates to identical top and bottom paths, ignoring their middle parts. Note that for a tag with $index = *$, an identical tag with any index value is a match.

To compute the similarity of path pattern, we consider both equal values and $*$. For $Can.P = body[1]/div[2]/span[1]/a[1]$ and $P = body[1]/div[1]/...$ $/div[*]/span[1]/a[1]$, we calculate match scores for top-down as $(2 + 1) + (2 + 2) = 7$, and for bottom-up as $(2+2+2)+(2) = 8$. Therefore, we get $Sim_p = max(7, 8)/max(4 \times 2, 5 \times 2) = 0.8$.

Content Pattern Adaption. An adaptive content pattern reflects the stability of the inner content of a block. It is represented by the shared elements $E \in \{N_1 \bigcap N_2 \bigcap ...\}$ of content patterns. For $N_1 = \{\langle``SPORTS", /a\rangle, \langle``Which"...\rangle,$ $\langle``J"...\rangle\}$, and $N_2 = \{\langle``SPORTS", /a\rangle, \langle``Wade"...\rangle, \langle``B"...\rangle\}$, we easily get $N = \{\langle``SPORTS", /a\rangle\}$. The adapted N is more stable and extracts fewer candidates with higher reliabilities. The method for computing the similarity between a candidate and an adaptive content pattern remains the same as in Section 4.2. Note that this pattern could be an empty set ($|N| = 0$), and then $Sim_n = 0$ for all candidates.

Context Pattern Adaption. Similar to adaptive content pattern, adaptive context pattern indicates the common contexts $\left\{X_1 \bigcap X_2 \bigcap ...\right\}$ of context patterns. If $X = \emptyset$, then its similarities for all candidates equal 0. Otherwise, this pattern similarity is computed as described in Section 4.2.

Example 2 shows the adaptive patterns for tracking the headlines (marked with a red border) in Figure 1, according to the original patterns from the original version and the new patterns from the new version.

Example 2 Adaptive Patterns for tracking the headlines in Figure 1

$A_{ori} = (11, 650, 639, 263, 302, 39)$
$P_{ori} = body[1]/div[2]/div[3]/span[1]/a[1]$
$N_{ori} = \{\langle``Latest:", /a\rangle,$ $\langle``\text{Bush's condition...}", /a\rangle\}$
$X_{ori} = \{``\text{December 27...}", ``\text{New York,NY}", ``34°", ``/", ``40°", ``°F", ``°C", ``5 - \text{day}", ``\text{EDITORS' PICKS}", ...\}$

$A_{new} = (418, 648, 230, 268, 351, 83)$
$P_{new} = body[1]/div[2]/div[3]/div[1]/ul[1]/li[1]/span[1]/a[1]$
$N_{new} = \{\langle``\text{Schwarzkopf, Desert...}", /a\rangle\}$
$X_{new} = \{``\text{December28...}", ``\text{New York,NY}", ``39°", ``/", ``29°", ``°F", ``°C", ``5 - \text{day}", ``\text{EDITORS' PICKS}", ...\}$

$A_{ori}, A_{new} \Rightarrow A_{ada} = (*, 649, *, 266, *, *)$
$P_{ori}, P_{new} \Rightarrow P_{ada} = body[1]/div[2]/div[3]/.../span[1]/a[1]$
$N_{ori}, N_{new} \Rightarrow N_{ada} = \emptyset$
$X_{ori}, X_{new} \Rightarrow X_{ada} = \{``\text{New York,NY}", ``/", ``°\text{F}", ``°\text{C}", ``5 - \text{day}", ``\text{EDITORS' PICKS}", ...\}$

5.2.2 Adjusted pattern weights

Over time, adaptive patterns could be very different from the initial patterns of a page. Some patterns remain stable and specific. Some patterns become more general (or vague) to capture additional dynamics. Typically, a specific pattern is effective in tracking blocks, but is sensitive to pattern changes. By contrast, a general pattern is not very precise in tracking blocks, but is robust to pattern changes. Hence, it is necessary to assign different weights to different patterns in our adaptive model.

$$Score = W_a * Sim_a + W_p * Sim_p + W_n * Sim_n + W_x * Sim_x \tag{2}$$

Algorithm 1 Adjust Pattern Weights

```
Input: updated adaptive $Patterns\ A, P, N, X$;
       all $Candidates$, including the new $Sample$, from the new version;
       unupdated pattern weights $W_a, W_p, W_n, W_x$;
Output: adjusted pattern weights $W_a, W_p, W_n, W_x$
  for all $Can \in Candidates$ do
    $Can.Sim_a \leftarrow$ the similarity between $A$ and $Can.A$
    $Can.Sim_p \leftarrow$ the similarity between $P$ and $Can.P$
    $Can.Sim_n \leftarrow$ the similarity between $N$ and $Can.N$
    $Can.Sim_x \leftarrow$ the similarity between $X$ and $Can.X$
    $Can.Score = W_a * Can.Sim_a + W_p * Can.Sim_p + W_n * Can.Sim_n + W_x * Can.Sim_x$
  end for
  repeat
    for all $pat \in Patterns$ do
      $maxSim = Max(Can.Sim_{pat}), \quad Can \in Candidates \setminus Sample$
      if $Sample.Sim_{pat} = 0$ and $W_{pat} \neq 0$ then
        $W_{pat} = 0$
      else if $Sample.Sim_{pat} > maxSim$ and $W_{pat} < maxSize$ then
        $W_{pat} = W_{pat} + stepSize$
      else if $Sample.Sim_{pat} = maxSim$ and $W_{pat} > midSize$ then
        $W_{pat} = W_{pat} - stepSize$
      else if $Sample.Sim_{pat} < maxSim$ and $W_{pat} > 0$ then
        $W_{pat} = W_{pat} - stepSize$
      end if
    end for
    for all $Can \in Candidates$ do
      $Can.Score = W_a * Can.Sim_a + W_p * Can.Sim_p + W_n * Can.Sim_n + W_x * Can.Sim_x$
    end for
    $maxScore = Max(Can.Score), \quad Can \in Candidates \setminus Sample$
  until $(Sample.Score > maxScore)$
```

To determine the pattern weights, we apply a simple rule: the new weights should enlarge the ratio of scores between the newest sample and the candidates. Considering performance, we use a hill-climbing algorithm [23] to find a local optimal solution for the problem and use the step size to adjust pattern weights. A pattern will get a higher weight if it helps to distinguish the newest sample from other candidates. A pattern valuing other candidates more, will reduce its weight. We observe that a specific pattern typically receives a higher weight than a general pattern because a general pattern is usually too vague to differentiate the sample among candidates.

Algorithm 1 shows the progress of adjusting pattern weights. First, we calculate the similarity between each updated adaptive pattern and each candidate from the newest version. Second, we compare the similarity of the newest sample with the other candidates, and adjust each pattern weight until the sample has the maximal score. Recall that the newest sample is selected from the previous patterns with their previous weights. More specifically, for a pattern, if the sample has an advantage over the other candidates, we increase the weight to enlarge the advantage in the overall similarity score; if the sample is at a disadvantage, we decrease the weight to reduce its influence on the score. In particular, for a irrelative pattern (e.g., $N = \emptyset$ or $Sim_n = 0$), we set its weight as 0. To be consistent with the CPM, we set the initial weight as 1 in the first version. We set $stepSize = 0.1$, $maxSize = 2$, and $midSize = 0.8$ in our experiments.

As an example, we adjust the weights for the adaptive patterns in Example 2. The initial weights W_a ,W_p ,W_n, and W_x in the original version are 1, 1, 1, and 1, respectively. After pattern adaption in the new version, A_{ada} and X_{ada} cannot distinguish the new sample from some candidates, P_{ada} distinguishes the sample from the others, and N_{ada} becomes invalid. Their updated weights are 0.9, 1.1, 0, and 0.9.

5.2.3 Discussion

One assumption behind the adaptive model is that changes between the versions are moderate and evolutional. This model does not well handle a dramatic version changes because new samples would be wrongly identified. Dramatic changes are also not handled well by the CPM and other existing models. Fortunately, Web pages typically change in an evolutional way, which is proven in our later experiments. If the adaptive model detects a large change in adaptive patterns or pattern weights, it can give an alert that the newest sample might be a mismatch.

Another advantage of the adaptive model is that it can naturally incorporate human feedback into the system. If a user finds that the returned block is wrong, he or she can simply label the correct block on the page. The system can then use the new sample to learn a new adaptive model. In our experiments in Subsection 6.5, we show that by incorporating human feedback, the adaptive model can be further improved.

6 Experimental results

In this section, we experiment with our proposed models (CPM and APM). We compare four single patterns as baseline approaches (Area, Path, ConteNt, ConteXt), because some of them are adopted as independent methods for tracking blocks in existing models (e.g., Area corresponds with “Visual Lenses” in [1] and Path is the same as “Structure Lenses” in [1] and “Direct Path Finding” in [16]). We also implement two existing algorithms for comparison. The first algorithm is extended from Textual Lenses in [1] and uses the equal tag path to extract candidates and then text similarity to rank candidates. We denote it as TextL. The second algorithm, which is denoted with EditD, is the tree edit distance algorithm used in [16].

6.1 Overall results

The overall accuracy results of all approaches are shown in Table 4. CPM and APM significantly outperform the four single patterns and two existing approaches, in terms of block-level and version-level accuracy. In Data1, the version-level accuracies of CPM and APM are 0.95 and 0.97, respectively. Data2 is larger and contains more diverse pages. Our two algorithms still work well, with a version accuracy reaching approximately 0.9 for both methods.

Although block-level accuracy is harder to achieve, our two models show significantly improved results over existing methods. APM can correctly track the block over all versions for approximately 90 % of human-created blocks in Data1, and approximately 80 % in Data2. These numbers are reasonably good, but indicate that there is still a potential for improving the adaptive model (such as implementing more complementary patterns, or designing an improved adaption method).

We find that APM is slightly superior to CPM on version-level accuracy. After APM adapts patterns and pattern weights according to the recent results, it can correctly identify some blocks that CPM cannot (1,662 and 202 block versions in Data2 and Data1 respectively). At the same time, if the recent results are incorrect, using them for pattern adaption can misguide APM in obtaining incorrect results (1,305 and 66 block versions in Data2 and Data1, respectively). However, APM is clearly superior to CPM on block-level accuracy; it improves 215 (6.79 %) blocks and impairs 49 (1.55 %) in Data 2, and improves 16 (8.99 %) blocks and impairs 5 (2.81 %) in Data 1. The reason for this is that APM adapts patterns and weights to keep track of the correct block for some pages, especially when the changes between the versions are moderate. There are always more blocks improved than impaired. Because block-level accuracy measures the overall success of tracking a block, we believe APM has an even greater advantage over CPM on users' perceived quality in real applications.

Table 4 shows that TextL outperforms the four simple patterns in terms of block level accuracy. Because TextL considers both path and text patterns, this actually indicates that combining path and content patterns can improve tracking robustness. EditD is worse than

Table 4 Overall accuracy result. † indicates the result is significant different (t-test,$p < 0.01$) from the results of all baselines (Area, Path, ConteNt, ConteXt, TextL, and EditD)

	Data1		Data2	
Method	Block	Version	Block	Version
Area	0.281	0.577	0.295	0.567
Path	0.674	0.854	0.599	0.816
ConteNt	0.444	0.543	0.366	0.478
ConteXt	0.421	0.581	0.491	0.662
TextL	0.697	0.830	0.634	0.792
EditD	0.612	0.754	0.583	0.739
CPM	$0.843^{\dagger}$	$0.949^{\dagger}$	$0.741^{\dagger}$	$0.898^{\dagger}$
APM	$0.904^{\dagger\circ}$	$0.967^{\dagger}$	$0.793^{\dagger\circ}$	$0.904^{\dagger}$

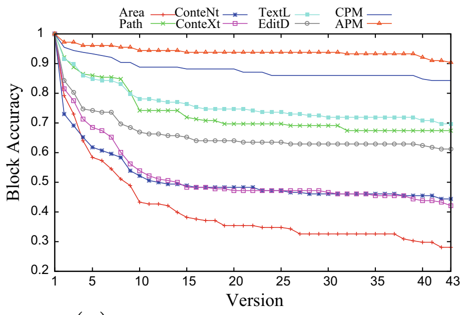
$^{\circ}$ indicates the result is significant different (t-test,$p < 0.01$) from the result of CPM

Path, which indicates that a stable inner structure is not as useful as a direct path. This is because there may be multiple blocks with similar structures. For example, in Figure 1, the structures of the three news blocks at the bottom (NEWS, SPORTS, and ENTERTAINMENT) are identical. It is difficult for EditD to select a correct block because their tree edit distances to the pattern might be similar (or the same).

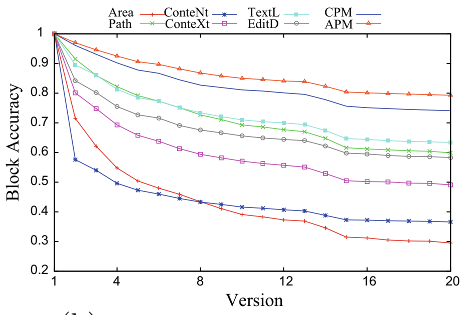
6.2 Stability over time

Figure 4 shows the accuracies of proposed methods on different page versions over time. It is clear that APM and CPM are much more stable than the six baselines, both in terms of version and block accuracies. The accuracies of the baselines quickly decreases over time, which indicates that these baselines quickly become less useful because of pages changes.

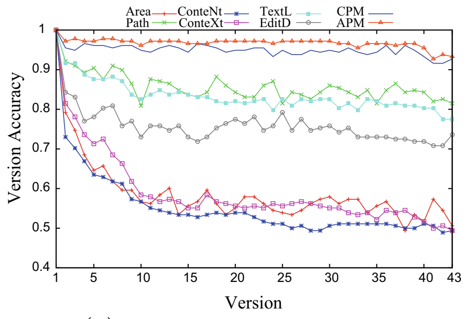
Figure 4a shows an unusual phenomenon of accuracy fluctuation over time. In Data1, the version accuracies of most approaches, except for APM, fluctuate frequently. After checking the data, we find that this phenomenon is primarily caused by the use of templates on some websites, whereby several templates are used interchangeably to present pages. For example, a news website adds a breaking-news block when a big event happens, but removes it afterwards. The patterns identified from the initial labelled block become invalid when a new template is applied, but return to work when the old template is reapplied. Because APM can automatically detect and adapt to such changes, it is much more stable in version accuracy than the other approaches.



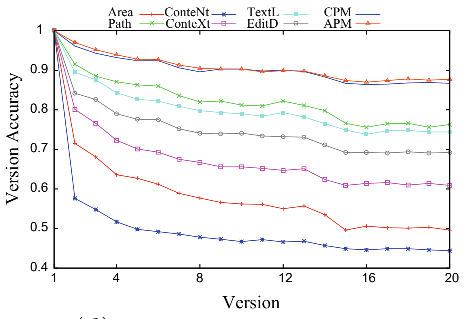
(a) Version-level accuracy on Data1



(b) Version-level accuracy on Data2



(c) Block-level accuracy on Data1



(d) Block-level accuracy on Data2

Figure 4 Accuracy of the approaches over time

6.3 Stability over changed blocks

In this subsection, we discuss the performance of all algorithms over the changed blocks, to determine whether the advantage of our algorithms is retained under different block changing scenarios.

In a preliminary analysis of the changed blocks, we find that 92.7 % of blocks in Data1 have changed over time according to three change metrics: DOM tree path (32.6 % of blocks), visual location (69.7 % of blocks), and text (76.4 % of blocks). For the changed blocks in Data1, 29.1 % of blocks are viewed as having low changes because they have only one change metric altered over time; 49.1 % of blocks with two change metrics altered are treated as having medium changes; and 21.8 % of blocks are viewed as having high changes because all three change metrics are altered. In Data2, 80.6 % of blocks have changed according to their DOM tree path (40.1 % of blocks), visual location (70.5 % of blocks), or text (51.3 % of blocks), where 29.8 % of blocks have low changes, 39.8 % of blocks have medium changes, and 30.4 % of blocks have high changes.

Table 5 compares the block accuracy results of all algorithms under different changing scenarios. The results show that the performance of all algorithms decreases with large changes, but APM always performs the best. More specifically, in Data1, APM outperforms all the baselines by more than 17 points in medium changes and more than 13 points in high changes; in Data2, APM outperforms all the baselines by more than 24 points in medium changes, and more than 15 points in high changes. This indicates that our proposed model is more effective than the baseline methods, particularly when the blocks have two or more change metrics altered over time.

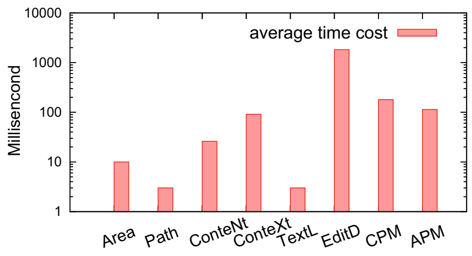
Our proposed models outperform all the baseline models in the two datasets, except for the low-change case in Data1. In analyzing the low-change cases, we find that the DOM tree path is rarely changed alone; it usually changes with visual location or text. Because there are no blocks with DOM tree path changed alone in Data1, Path successfully tracks all these blocks for low changes.

6.4 Performance

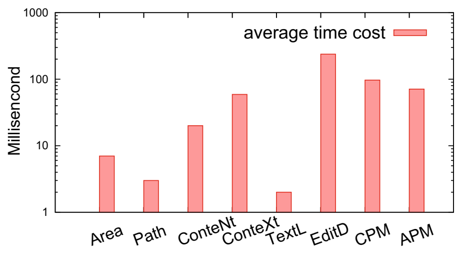
Because we hope to use the algorithms to track web blocks in real time, we compare their performance in Figure 5. Area, Path, ConteNt, ConteXt, and TextL use simple and fixed

Table 5 Block accuracy comparison over changed blocks. Best results are in bold

	Data1			Data2		
Change	Low (29.1 %)	Medium (49.1 %)	High (21.8 %)	Low (29.8 %)	Medium (39.8 %)	High (30.4 %)
Area	0.771	0.000	0.000	0.369	0.044	0.000
Path	1.000	0.728	0.000	0.942	0.557	0.000
ConteNt	0.458	0.457	0.417	0.518	0.308	0.179
ConteXt	0.604	0.346	0.222	0.665	0.406	0.134
TextL	0.708	0.679	0.611	0.842	0.547	0.273
EditD	0.750	0.605	0.361	0.754	0.520	0.241
CPM	0.979	0.815	0.667	0.961	0.703	0.372
APM	1.000	0.901	0.750	0.986	0.804	0.427



(a) Average Time Cost on Data1



(b) Average Time Cost on Data2

Figure 5 Performance of tracking a block in a version

patterns and thus are very fast. EditD uses the edit distance to locate blocks and is the slowest baseline, especially when the inner structure of a block is complicated. CPM and APM are slower than the fast algorithms, but their average block-tracking processing time in a version is approximately 100 ms, which is sufficient for most real-time applications.

The execution times of some methods (ConteXt, TextL, CPM, and APM) for Data2 are slightly smaller than those for Data1. One of possible reasons for this is that Data2 has more static blocks (19.4 %) than Data1 (7.3 %) (see Subsection 6.3), which never change and are easy to follow. Moreover, the blocks in Data1 are more complicated than those in Data2 (see Subsection 3.2), and EditD is quite sensitive to block complexity. It is reasonable that EditD spends much more time to track a block in Data1 than in Data2.

6.5 Effect of human feedback

As mentioned in Subsection 5.2.3, APM can incorporate user feedback into the tracking process. When a block is wrongly detected in the newest version, users can label the correct block instead. We conduct experiments to test the effectiveness of user feedback. Table 6 shows the results of tracking accuracy when 1−5 feedback labels are provided. Overall, feedback is very effective in improving the tracking accuracy, especially for page accuracy. The first feedback label produces the largest improvement margin. The capability of incorporating user feedback is critical for real applications. Sometimes web pages change dramatically, so user feedback can help algorithms to learn the correct track.

Table 6 Results with user feedback

Data	Data1		Data2	
APM/Acc	VerAcc	PageAcc	VerAcc	PageAcc
No-Feedback	0.967	0.904	0.904	0.793
Feedback1	0.967	0.949	0.913	0.858
Feedback2	0.979	0.949	0.926	0.875
Feedback3	0.984	0.961	0.933	0.881
Feedback4	0.984	0.966	0.938	0.886
Feedback5	0.988	0.978	0.942	0.889

Because of the limitations of space, in the remaining parts of this section, we only report the results for Data2. All experiments using both Data1 and Data2 resulted in similar trends.

6.6 Effect of different blocks sizes and complexities

We determine whether our models could consistently outperform the baselines in tracking all types of blocks. We use our algorithms on blocks with various sizes and complexities. Block size is represented by its visual size, calculated by its width * height in pixel; block complexity is the number of inner element nodes contained in a block. Figure 6 shows that

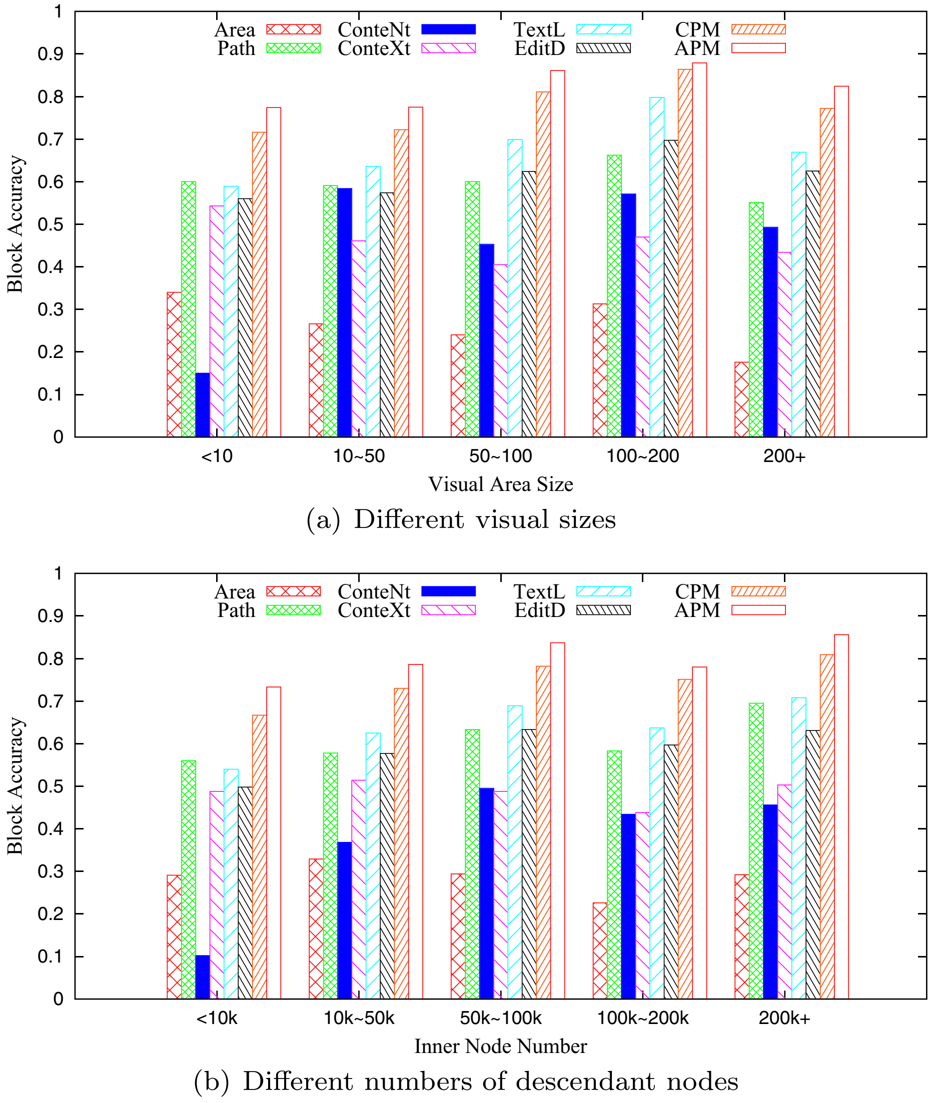


Figure 6 Block level accuracy on blocks with different visual sizes or different number of descendant nodes in Data2

Table 7 Different pattern combinations in Data2. The area, path, content, and context patterns are respectively denoted as A, P, N, and X in the different pattern combinations

	CPM		APM	
Method	Block	Version	Block	Version
AX	0.623	0.813	0.743	0.868
AP	0.602	0.825	0.763	0.894
AN	0.670	0.843	0.762	0.884
PX	0.706	0.872	0.789	0.899
NX	0.659	0.817	0.748	0.867
PN	0.730	0.894	0.785	0.901
ANX	0.709	0.873	0.773	0.890
APX	0.710	0.877	0.793	0.902
APN	0.723	0.893	0.788	0.902
PNX	0.738	0.898	0.792	0.903
APNX	0.741	0.898	0.793	0.904

our proposed models, CPM and APM, consistently outperform other models for all ranges, and APM always performs the best. This indicates that our models are general and can be used under different situations for tracking different types of blocks. The figure also shows that ConteNt is sensitive to block size; it does not work well when the block is small. When a block is too small, or contains only a few inner nodes, there is insufficient content for identifying the block. ConteXt is superior than ConteNt for tracking small blocks, because it can usually find sufficient context. Area and Path pattern are insensitive to block size and block complexity.

6.7 Effect of different combinations

Table 7 shows the performance of different pattern combinations for CPM and APM. We find that the path and content patterns are very strong features in CPM; their performance is close to the result of combining the four patterns. The area and context patterns are less useful in CPM.

In APM, the adaptive path is a strong feature. Each combination with the path is superior to those without. Context changes from a weak baseline in CPM to a strong pattern in APM. This confirms our observation that most blocks can be located with a stable context. An essential finding is that APM achieves consistently high accuracy with any combination of features, even when only two features are combined. Even the weakest combination of two patterns, Area and Context, attains a higher block accuracy than CPM's combination of all four patterns.

7 Conclusions and future work

In this paper, we implemented and studied four patterns used to track blocks in Web pages. We proposed two models, namely the Combined Pattern Model (CPM) and Adaptive Pattern Mode (APM), to combine these patterns to more accurately and reliably track blocks. The experimental results showed that these two models outperformed the use of a single pattern, with the APM performing the best. In the future, we will explore additional patterns that can provide complementary information to the existing patterns. We are also interested in enhanced adaptive models that can further improve the reliability of block tracking.

Acknowledgments This work was partially supported by the National Basic Research Program of China (973 Program) [No.2014CB340403], the Fundamental Research Funds for the Central Universities, the Research Funds of Renmin University of China [No. 14XNLF05 and No. 15XNLF03], the National Culture Science and Technology Promotion Plan, the National Natural Science Foundation of China [No.61502501], and the secondary network prototype system development project by Xinhua News Agency.

References

1. Adar, E., Dontcheva, M., Fogarty, J., Weld, D.S.: Zoetrope: interacting with the ephemeral Web. In: Proceedings of the 21st annual ACM symposium on User interface software and technology, UIST 08, p. 239C248, CA, USA (2008)
2. Adar, E., Teevan, J., Dumais, S.T.: Resonance on the Web: Web dynamics and revisitation patterns. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 09, p. 1381C1390, MA, USA (2009)
3. Adar, E., Teevan, J., Dumais, S.T., Elsas, J.L.: The Web changes everything: Understanding the dynamics of Web content. In: Proceedings of the 2nd ACM International Conference on Web Search and Data Mining, WSDM 09, p. 282C291, Barcelona, Spain (2009)
4. Agrawal, N., Ananthanarayanan, R., Gupta, R., Joshi, S., Krishnapuram, R., Negi, S.: Eshopmonitor: A Web content monitoring tool. In: Proceedings of the 20th International Conference on Data Engineering, ICDE 04, p. 817C820, MA, USA (2004)
5. Anderson, C.R., Horvitz, E.: Web montage: A dynamic personalized start page. In: Proceedings of the 11th International Conference on World Wide Web, WWW 02, p. 704C712, Hawaii, USA (2002)
6. Boyapati, V., Chevrier, K., Finkel, A., Glance, N., Pierce, T., Stockton, R., Whitmer, C.: Changedetector: A site-level monitoring tool for the www. In: Proceedings of WWW 2002, p. 570C579, Hawaii, USA (2002)
7. Cai, D., Yu, S., Wen, J.R., Ma, W.Y.: Vips: a vision-based page segmentation algorithm. In: Microsoft Technical Report, p. MSRCTRC2003C79 (2003)
8. Cai, D., Yu, S., Wen, J.R., Ma, W.Y.: Block-based Web search. In: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 04, p. 456C463, Sheffield, United Kingdom (2004)
9. Cho, J., Garcia-Molina, H.: The evolution of the Web and implications for an incremental crawler. In: Proceedings of the 26th International Conference on Very Large Data Bases, VLDB 00, p. 200C209, Cairo, Egypt (2000)
10. Dontcheva, M., Drucker, S.M., Salesin, D., Cohen, M.F.: Changes in Webpage structure over time. In: UW CSE Technical Report (2007)
11. Dontcheva, M., Drucker, S.M., Wade, G., Salesin, D., Cohen, M.F.: Summarizing personal Web browsing sessions. In: Proceedings of UIST 2006, p. 115C124, Montreux, Switzerland (2006)
12. Douglis, F., Ball, T., Chen, Y.f., Koutsofios, E.: The at&t internet difference engine: Tracking and viewing changes on the Web. World Wide Web **1**(1), 27C44 (1998)
13. Fetterly, D., Manasse, M., Najork, M., Wiener, J.: A large-scale study of the evolution of Web pages. In: Proceedings of WWW 2003, p. 669C678, Budapest, Hungary (2003)
14. Freire, J., Kumar, B., Lieuwen, D.: Webviews: Accessing personalized Web content and services. In: Proceedings of WWW 2001, p. 576C586, Hong Kong (2001)
15. Greenberg, S., Boyle, M.: Generating custom notification histories by tracking visual differences between Web page visits. In: Proceedings of Graphics Interface 2006, GI 06, p. 227C234, Quebec, Canada (2006)
16. Han, J., Han, D., Lin, C., Zeng, H.J., Chen, Z., Yu, Y.: Homepage live: Automatic block tracing for Web personalization. In: Proceedings of WWW 2007, p. 1C10, Alberta, Canada (2007)
17. Hupp, D., Miller, R.C.: Smart bookmarks: automatic retroactive macro recording on the Web. In: Proceedings of UIST 2007, p. 81C90, Rhode Island, USA (2007)
18. Kushmerick, N.: Wrapper induction for information extraction. Ph.D. thesis. University of Washington (1997)

19. Liu, B., Grossman, R., Zhai, Y.: Mining data records in web pages. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 03, p. 601C606, Washington, D.C (2003)
20. Liu, B., Zhai, Y.: Net - a system for extracting web data from flat and nested data records. In: Proceedings of the 6th International Conference on Web Information Systems Engineering, WISE 05, p. 487C495, New York, NY (2005)
21. Liu, L., Pu, C., Tang, W.: Webcq - detecting and delivering information changes on the Web. In: Proceedings of the Ninth International Conference on Information and Knowledge Management, CIKM 00, p. 512C519, Virginia, USA (2000)
22. Muslea, I., Minton, S.N., Knoblock, C.A.: Active learning with strong and weak views: A case study on wrapper induction. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI 03, p. 415C420, Acapulco, Mexico (2003)
23. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. 3rd. Prentice Hall Press, Upper Saddle River, NJ, USA (2009)
24. Sugiura, A., Koseki, Y.: Internet scrapbook: automating web browsing tasks by demonstration. In: Proceedings of UIST 1998, p. 9C18, California, USA (1998)
25. Teevan, J., Dumais, S.T., Liebling, D.J.: A longitudinal study of how highlighting Web content change affects peoples Web interactions. In: Proceedings of CHI 2010, p. 1353C1356, Georgia, USA (2010)
26. Teevan, J., Dumais, S.T., Liebling, D.J., Hughes, R.L.: Changing how people view changes on the Web. In: Proceedings of UIST 2009, p. 237C246, BC, Canada (2009)
27. Zhai, Y., Liu, B.: Web data extraction based on partial tree alignment. In: Proceedings of WWW 2005, p. 76C85, Chiba, Japan (2005)
28. Zhai, Y., Liu, B.: Extracting Web data using instance-based learning. World Wide Web **10**(2), 113C132 (2007)
29. Zhao, H., Meng, W., Wu, Z., Raghavan, V., Yu, C.: Fully automatic wrapper generation for search engines. In: Proceedings of WWW 2005, p. 66C75, Chiba, Japan (2005)