Author manuscript, published in "TWAW (Temporal Workshop on Web Archiving) (2011)"

Deriving Dynamics of Web Pages: A Survey*

Marilena Oita
INRIA Saclay – Île-de-France
& Télécom ParisTech; CNRS LTCI
Paris, France
marilena.oita@telecom-paristech.fr

Pierre Senellart
Institut Télécom
Télécom ParisTech; CNRS LTCI
Paris, France
pierre.senellart@telecom-paristech.fr

inria-00588715, version 1 - 26 Apr 2011

ABSTRACT

The World Wide Web is dynamic by nature: content is continuously added, deleted, or changed, which makes it challenging for Web crawlers to keep up-to-date with the current version of a Web page, all the more so since not all apparent changes are significant ones. We review major approaches to change detection in Web pages and extraction of temporal properties (especially, timestamps) of Web pages. We focus our attention on techniques and systems that have been proposed in the last ten years and we analyze them to get some insight into the practical solutions and best practices available. We aim at providing an analytical view of the range of methods that can be used, distinguishing them on several dimensions, especially, their static or dynamic nature, the modeling of Web pages, or, for dynamic methods relying on comparison of successive versions of a page, the similarity metrics used. We advocate for more comprehensive studies of the effectiveness of Web page change detection methods, and finally highlight open issues.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*

General Terms

Algorithms, Experimentation, Measurement

Keywords

Change monitoring, Web archiving, Timestamping

1. INTRODUCTION

The World Wide Web challenges our capacity to develop tools that can keep track of the huge amount of information that is getting modified at speed rate. Web archiving crawlers [25], especially, need the ability to detect change in Web content and to infer when a Web page last changed. This ability is fundamental in maintaining the coherence of the crawl, in adjusting its refresh rate, in versioning, and to allow a user to retrieve meaningful temporal data. The understanding of the dynamics of Web pages, that is, how fast the Web content changes and what the nature of these changes is, its implications on the structure of the Web, and the correlation with the topic of pages are also popular subjects in the research literature [4].

In addition to being of paramount importance to Web archiving, the subject of change detection is of interest in various applications and domains, such as: large-scale information monitoring and delivery systems [18, 15, 24, 17, 23] or services[1], Web cache improvement [11], version configuration and management of Web archives [30], active databases [18], servicing of continuous queries [1]. Research has focused on finding novel techniques for comparing snapshots of Web pages (a reference Web page and its updated version) in order to detect change and estimate its frequency. Change can however be detected at various levels: there are various aspects of dynamics which must be considered when studying how Web content changes and evolves.

The majority of works have seen the change detection problem from a *document-centric* perspective, as opposed to an *object-centric* one. By object or entity we mean here any Web content, part of a Web page, that represents meaningful information per se: image, news article, blog post, comment, etc.. Comparatively, little effort has been put on making the difference between relevant changes and those that might occur because of the dynamic template of a Web page (ads, active layout, etc.), i.e., its boilerplate [21].

We study in this article some of the strategies that have been established in different settings, with the aim at providing an overview of the existing techniques used to derive temporal properties of Web pages.

There is a large body of work on the related problem of change detection in XML documents, particularly for purposes of data integration and update management in XML-centric databases. However, the solutions developed for XML documents cannot be applied without serious revisions for HTML pages. The model assumptions made for XML do not really hold for HTML. Indeed, the HTML and XML formats have a key difference: while an XML page defines the nature of the content by its *meta* tags, HTML tags define

*This research was funded by the European Research Council grant Webdam FP7-ICT-226513.

Copyright 2011 for the individual papers by the papers' authors. Copying permitted only for private and academic purposes. This volume is published and copyrighted by its editors.
TWAW 2011 March 28, 2011, Hyderabad, India

[1]Examples of these include `http://timelyweb.en.softonic.com/`, `http://www.urlywarning.net/`, `http://www.changealarm.com/`, `http://www.changedetect.com/`.

inria-00588715, version 1 - 26 Apr 2011

mainly presentational aspects of content within. In addition to the challenges that exist in XML documents, Web pages add some others by their lack of formal semantics, a fuzziness regarding the formatting, by the embedding of multimedia and scripts, etc.. Separate approaches commonly need to be adopted and the research done on XML documents is beyond the scope of our paper, although we mention some works on XML [37, 14] that have particular relevance to Web page change detection. We refer the reader to [13] for a survey of XML change detection algorithms.

We focus in this survey on deriving dynamics of **HTML documents**.

Change detection mechanisms can either be *static*, estimating the date of last modification of content from the Web page itself (its code, semantics or neighbors), or *dynamic*, by comparing successive versions of a Web page. The structure of this article reflects these dimensions. In Section 2, we present static approaches to timestamping Web pages, while Section 3 introduce dynamic methods. We then analyze in Section 4 the different models of a Web page used by existing techniques for comparing successive versions. Similarity metrics used in dynamic methods are independently investigated in Section 5. We briefly describe statistical modeling approaches to estimate change frequency in Section 6. We conclude with a discussion of some remaining open questions.

2. STATIC APPROACHES: TIMESTAMPING

This section deals with methods for inferring temporal properties of a Web page in a *static* manner as opposed to the commonly used *dynamic* computation of the difference between successive versions of a given Web page. The goal here is to infer the creation or the last modification date of a Web page or, possibly, of some parts of it. We study sources of data that can be useful for that purpose: metadata, the content of the Web page itself, or its graph neighborhood. The canonical way for timestamping a Web page is to use the `Last-Modified` HTTP header. Unfortunately, studies have shown this approach is not reliable in general [12]. We describe next why this happens in practice and other techniques for timestamping Web pages.

2.1 HTTP metadata

HTTP/1.1, the main protocol used by Web clients and servers to exchange information, offers several features of interest for timestamping, the foremost of which are the `ETag` and `Last-Modified` HTTP response headers. Entity tags (or ETags) are unique identifiers for a given version of a particular document. They are supposed to change if and only if the document itself changes. Servers can return this with any response, and clients can use the `If-Match` and `If-None-Match` HTTP requests headers to condition the retrieval of the document to a change in the ETag value, avoiding then to retrieve already known contents. `If-Modified-Since` and `If-Unmodified-Since` provide conditional downloading features, in a similar way as for ETags. Even when conditional downloading is not possible, Etags and HTTP timestamps can be retrieved by a Web client without downloading a whole Web page by making use of the `head` HTTP method. The problem is that while this information is generally provided and is very reliable for static content (e.g., static HTML pages or PDF), it is most of the time missing or changed at every request (the timestamp given is that of the request, not of the content change) when the content is dynamic (generated by content management systems, etc.). Some CMSs do return correct HTTP timestamps, such as MediaWiki[2], but they seem to be a minority.

In [12], Clausen presents an experimental study of the reliability of Etags and HTTP timestamps on a collection of a few million Danish Web pages. He finds out that the best strategy for avoiding useless downloads of versions of Web pages already available in a Web archive is to always download when the ETag server is missing, and otherwise download only if the `Last-Modified` header indicates change. This rather counterintuitive result yielded in this experimental study an almost perfect prediction of change, and a 63% accuracy in predicting non-change. Given that the majority of Web servers run some version of the open-source Apache[3] HTTP server [26], it would be interesting to see whether this strategy is correlated with some inherent behavior of this software. Furthermore, repeating this experiment on a larger scale and with a more recent set of Web pages would be of interest.

HTTP also provides the `Cache-Control` and `Expires` response headers. This information is often given, but with a zero or very low expiration delay, which means that nothing interesting can be derived from it. In some specific and controlled environments (e.g., Intranets), it might still be useful to look at these two pieces of information to estimate the refresh rate of a Web page.

2.2 Timestamps in Web content

Content management systems, as well as Web authors, often provide in the content of a Web page some human-readable information about its last date of modification. This can be a global timestamp (for instance, preceded by a "Last modified" string, in the footer of a Web page) or a set of timestamps for individual items in the page, such as news stories, blog posts, comments, etc. In the latter case, the global timestamp might be computed as the maximum of the set of individual timestamps. It is actually quite easy to extract and recognize such information, with keyword selection (*last*, *modification*, *date*, etc.) or with entity recognizers for dates (built out of simple regular expressions). However, this timestamp is often quite informal and partial: there is sometimes no time indication, and most of the time no timezone. To the best of our knowledge, no formal study of the precision reached by extracting timestamps from Web content has been carried out.

2.3 Semantic temporal associations

In addition to these timestamps provided to humans, documents on the Web may include additional semantic timestamps meant for machines. No mechanism for this exists in HTML *per se*, but the HTML specification [36] allows for arbitrary metadata in the form of `<meta>` tags, one particular profile of such metadata being Dublin Core[4] whose `modified` term indicates the date of last modification of a Web page. Both content management systems and Web authors occasionally use this possibility. Web *feeds* (in RSS or Atom formats) also have semantic timestamps, which are quite reliable, since they are essential to the working of applications that exploit them, such as feed readers. In some cases, external semantic content can be used for dating an HTML Web

[2] `http://www.mediawiki.org/`
[3] `http://www.apache.org/`
[4] `http://dublincore.org/documents/dcmi-terms/`

inria-00588715, version 1 - 26 Apr 2011

page: for instance, an RSS feed containing blog entries can be mapped to the corresponding Web page, in order to date individual items [29]. Another case is that of *sitemaps* [35]. Sitemaps are files that can by provided by the owner of a Web site to describe its organization, so as to improve its indexing by Web search engines. Sitemaps allow for both timestamps and change rate indications (*hourly*, *monthly*, etc.), but these features are not often used. Very few content management systems produce all of this, although it would have been the ideal case: the download of a single file would suffice to get all timestamping information about the whole Web site.

2.4 Using the neighborhood

It is possible to use the graph structure of the Web to help timestamping Web pages: [28] uses the neighboring pages of a Web page to estimate its timestamp. When no source of reliable timestamps is found for a given page using one of the technique described above, its timestamp is set to some form of average of the timestamps of pages pointed to and by this page. The inherent assumption is that pages linked together tend to have a similar update patterns. The precision is not very high, but better than nothing when no other information is available.

3. DYNAMIC METHODS

When static techniques do not give adequate results, there is still the possibility of comparing a Web page with its previous version in order to determine (sometimes in a rough way) an equivalent of `Last-Modified`. The timestamp that gives the last modification of a Web page can be inferred then as the date when change has been detected.

Nevertheless, for a precise estimation, a *just-in-time* crawl of versions is needed. In reality, the frequency of change is quite difficult to estimate because Web pages have different patterns of change. In general, many factors determine variations in the frequency of change for a given Web page: the CMS, the subject, the time of the year, even the time of the day, etc.

Estimating the Web pages' frequency of change is the subject of many studies [16, 2, 27, 10, 19]. Their results are however heavily dependent on the technique of detecting change that they have used.

There are two parameters that influence the process of determining the dynamics:

1. the **frequency of change** is not known in advance, but if we do not crawl the Web pages on time, we miss versions and the timestamp detected will be then imprecise;

2. the **change detection technique**, which heavily relies on the similarity metrics and on the model (that can capture more or less types of changes) and the filtering of dynamic elements that influence the frequency without being truly relevant (and which occur quite often in Web pages because of AJAX applications or advertisements).

A method of filtering irrelevant content is to know what is important rather than trying to filter what is unimportant. If we knew in advance the frequency of crawl, then we would know when change occurs and therefore set timestamps for new versions. This is unfortunately not the case, but we need however to crawl frequently enough (better too frequently than not frequent enough). Once we have these versions, we can detect if there is any change that happened in the interval of time that represents the interval of crawl. Based on this, timestamps can be derived with good approximation.

The majority of works consider that they have an access to the versions and they focus on detecting change efficiently. Few make a semantic distinction between changes by disregarding insignificant ones. Next, we present some general notions about changes: what kind of changes can occur in Web pages, which have been identified in our studied approaches and which have not, and finally we give an insight into how change is actually represented.

3.1 Types of changes

We summarize here the types of changes detected by different approaches. There are however other, more sophisticated types that are sometimes mentioned, but not treated. For instance, behavioral changes are mentioned in [38]; they occur in active HTML elements like scripts, embedded applications and multimedia. These new forms of Web content have a big impact on the Web today, but they require a more complex modeling.

All considered approaches detect changes in content.

Works that model the HTML page as a tree, including page digest encoding, usually detect also structural and attribute changes. However, differences exist in the coverage of cases for these types of changes. For example, MH-Diff [9] detects also *move* and *copy* structural changes, which is an improvement over the traditional detection of *insert*, *delete* and *update*. Structural (or layout) changes occur when the position of elements in the page is modified. Attribute (or presentation) changes are related to the representation of information, for instance changes in the font, colors or captions. For capturing structural and attribute changes, the model has to be aware their existence; this implies a more complex model which influences generally in a negatice manner the performance. Unlike flat-file models of Web pages, the output of content change detection in hierarchical models is more meaningful: the type of node in which the content change occured can also be identified.

There are also *type* changes, that are modifications which come about when the HTML tags change: e.g., a **p** tag which becomes a **div**. Type changes can be detected by [31] which uses the page digest encoding that provides a mechanism for locating nodes of a particular type (see further).

Semantic types of change capture the meaning of content that has changed. They are defined in SCD [23], a pioneer work in this direction.

Changes are sometimes captured in a quantitative manner rather than in a qualitative one. In contrast with the qualitative way, where the change is described (in a delta file) or visualized in a comparative manner, quantitative approaches estimate the amount of change of a specific type. More specifically, all approaches that use the similarity formula defined in CMW [17] do not reconstruct the complete sequence of changes, but give a numerical value of it. In this case, supposing a threshold of change, we can determine if a page has changed or not, which actually represent more an oracle response, still useful in the majority of applications.

3.2 The representation of change

There are various ways to present the difference between

inria-00588715, version 1 - 26 Apr 2011

two documents. Changes are usually stored in a physical structure generically called *delta file* or *delta tree*. The format of storing the change for RMS [38] consists in specialized set of arrays that capture the relationships among the nodes and the changes that occur both in structure and content. Systems for monitoring change like [24, 18] have typically a user interface and present changes in a graphical way. HTML*diff* merge the input Web page versions into one document that will summarize all the common parts and also the changed ones. The advantage is that the common parts are displayed just once, but on the other hand, the resulting merged HTML can be syntactically or semantically incorrect. Another choice linked to change presentation is to display only the differences and omit the common parts of the two Web pages. When the documents have a lot of data in common, presenting only the differences could be better, with the drawback that the context is missing. The last approach is to present the differences between the old and new version side by side.

These presentation modes are used in combination, rather than being the unique choice for a given system. For example, [24, 18] are presenting the results of the change monitoring service using a hybrid approach: presentation modes are combined depening on the type of change that is presented.

4. HTML DOCUMENT MODELS

This section contains an overview of the models that are considered in the quest of detecting changes in Web documents. The modeling step is a key one as it will determine the elements on which the comparison algorithms operate.

We first discuss the "naïve" approach, that is, to consider Web pages as flat files (strings); then we describe tree models, a popular choice in the literature. We explore also some approaches that are based on tree models, but which are essentially transforming the two versions to be compared in a bipartite graph on which specific algorithms are applied. Finally, we present the *Page Digest* design of Web pages, a manner of encoding that clearly separates structural elements of Web documents from their content, while remaining highly compact.

4.1 Flat-files

Some early change detection systems model Web pages as flat files [15]. As these models do not take into account the hierarchical structure of HTML documents and neither the characteristics of the layout, they can detect only *content changes* – and this without making any semantic difference in the content.

Some works [34] try to filter first irrelevant content by using heuristics on the type of content and regular expressions. After this basic filtering, the Web page content is direcly hashed and compared between versions. Unfortunately, we can never filter all kind of inconvenient content, especially when its manner of encoding or type get more complex.

4.2 Trees

A natural approach is to represent a Web page as a tree using the DOM model. By taking into account the hierarchies, *structural* and *attribute* changes can be detected besides *content* changes.

Differences between tree models appear in their *ordered* characteristics and in the level of granularity on which change is detected: node, branch, subtree or "object". We will further discuss these aspects.

First, the modeling of a Web page into a tree requires a preprocessing step of cleaning. This is a significant one because it corrects missing or mismatching, out-of-order end tags, as well as all other syntactic ill-formedness of the HTML document. A tree is constructed first by filtering the "tag soup" HTML into an XML document and second, by manipulating the result in order to obtain a workable tree structure using implementations of the DOM standard (that usually employ XSLT and XPath). Sometimes also an initial pruning of elements is done: [22] is filtering out scripts, applets, embedded objects and comments.

Many works do not specify how they realize the cleaning, so either they assume it to be done in advance, or they solve this issue by using an HTML cleaning tool. HTML Tidy[5] is well-suited for this purpose and mentioned in [3].

There are however cases [23] when the tree model does not need to be cleaned: as the semantics of tags (value, name) is leveraged in the technique, the structure does not have to be enforced.

Ordered trees.

The *ordered* characteristic of trees implies that the order of appearance of nodes is considered in the algorithm and therefore included in the model.

RMS algorithm [38] stores the level (*depth*) of a node, information which will be used in the tree traversal and parsing. The SCD [23] algorithm uses branches of ordered trees to detect *semantic* changes. The notion of branch is used to give the context of a node and is formalized as an ordered multiset, in which the elements are designated by the *tag name* of the HTML non-leaf node, or its content (text), if it represents a leaf node. The order is very important in this model because of the *data* hierarchy considered (e.g., `book.author.name.Eminescu` vs. a markup hierarchy like `div.p.b.PCDATA`). In this model, if we change the order, we change the semantics of hierarchies or this semantics becomes incoherent.

Ordered tree model is also used in [20].

Unordered trees.

The unordered labeled tree model does not consider the order of appearance of elements in the tree as relevant, instead only the parent-child relationships are captured.

It is mentioned in MHDiff [9] that the change detection problem for unordered trees is harder that for ordered ones. Like [17, 9], [22] constructs a weighted bipartite graph from the two trees given as entry. In these models, the order has not an importance on the final structure for which further processing will be done, therefore this feature is not captured.

[3] delimits and encodes subtrees; these are generated by analyzing the level of a node in the unordered tree. From the root, when we arrive at level % 3, a subtree is created; the node at (level % 3 + 1) becomes the *local* root of the following subtree, and this iteratively until the end of the hierarchy. Each subtree is marked with the tag name of its local root and indexed based on its start and end node identifiers. A hashtable will finally map metadata about nodes (like tag name, content, path to the root, attributes pair values, etc.).

[5] `http://tidy.sourceforge.net/`

inria-00588715, version 1 - 26 Apr 2011

4.3 Bipartite graph

The bipartite graph model is a model derived from unordered trees: it consists in two independent sets (acquired after tree pruning), each corresponding to a version, that are connected by cost edges. As set elements, subtrees are chosen over nodes in [22, 17]. The assumption [17] is that we might be more interested in which subtree the change occurs, than in which specific node.

The cost of an edge represent the cost of the edit scripting needed to make a model entity (node or subtree) of the first set *isomorphic* with one of the second set. The similarities between all subtrees in the first tree and all those of the second tree are computed and placed in a cost matrix. Having this matrix, the Hungarian [6] algorithm is used to find in polynomial time a minimum-cost bijection between the two partitions. This algorithm is typically used in linear programming to find the optimal solution to the assignment problem.

CMW [17] as well as MH-Diff [9] algorithm, are based on transforming the change detection problem in one of computing a minimum cost edge cover on a bipartite graph. Optimizations are brought out in [22] in comparison with the work in [6], but the general aim and similarity metrics remain the same as in [3].

4.4 Page Digest

The page digest model[6] has been adopted in [31] and [38], and represents a more compact encoding of data than HTML and XML formats, while preserving all their advantages. To construct this model, some steps are performed: counting the nodes, enumerating children in a depth-first manner (in order to capture the structure of the document), and content encoding and mapping for each node - encoding which will preserve the natural order of text in the Web page.

SDiff [31] is a Web change monitoring application that uses a digest format that includes also tag type and attribute information. RMS [38] also uses this model, although without giving many details. The advantages of the Page Digest over the DOM tree model are enumerated in [31]. We mention minimality and execution performance: the reduction of tag redundancy gives a more compact model, therefore a faster document traversal. The algorithms developed on this model run in linear time without making too many heuristics or restrictions, while capturing also a large palette of changes.

5. SIMILARITY METRICS

Similarity metrics are used in dynamic methods of detecting change, in the matching stage of the two versions modelled as described in section 4.

For string models, (content) change is identified when the data strings are discovered to be partially unsimilar. For tree models that have various dimensions, the problem gets more complex.

The aim is to identify model elements that are essentially the same, but which have been affected by change. Model elements that are the identical are pruned because they have basically not changed between versions; also, totally dissimilar model elements do not represent instances of the object that has evolved, so will not be included in further processing steps. Essentially, only *approximatively similar* model elements will be further studied. A typical matching is based on comparisons of attribute values of the model elements. If they have the same properties, then they are similar.

We will describe next the types of similarity metrics used for change detection in different settings or applications, for the studied cases.

[6] `http://www.cc.gatech.edu/projects/disl/PageDigest/`

5.1 String matching techniques

Approaches that compare two Web pages modeled as flat files (i.e. strings), rely on hash-based methods, edit distance metrics or techniques based on the longest common subsequence. We do not exhaustively cover all the possiblities, but rather present some of them that we have encountered in the analyzed works.

Naïve: Jaccard.

One simple technique for change detection in text is to count the number of words that are common (regardless of their position) for two string sequences and to divide the result by the number of distinct words. Another possiblity is to divide by the length of the first string, as done in [30].

Hash-based methods.

In this category, we mention [8, 20, 3]. The method of [8] uses shingling, which is a flexible method of detecting changes between two strings. It is usually referred to as w-shingling, where w the denotes the number of tokens in each shingle in the set. A *shingle* is a contiguous subsequence (w-gram) of the reference string. For the two strings to be compared, their shingles are hashed; if these strings have a lot of shingle values in common, then they are similar.

Another method is to use *signatures*. A signature of a string is a function of its hashed value. When the model is hierarchical, the signature of a node is computed based on its terminal path. For a formatting (or non-leaf) node, the signature represents the sum of the signatures of its children, until leaf nodes, where changes are actually detected. In [20], only nodes that have different signatures from those in the original version will be compared. Change detection algorithms that employ signatures have the disadvantage that false negatives are possible: change exists, but a different signature for it does not. It obviously depends on the careful choice of the space of hashing, and eventually on the application: if it can tolerate or not false results.

Another hash-based approach is presented in [3], where the change is detected at atomic element level using a direct hash.

Longest common subsequence.

A **diff** is a file comparison program that outputs the differences between two files. Diff tools are based on computing the longest common subsequence between two strings [32]. For instance, HTML*Diff* uses GNU diff utility adapted for HTML page change detection. This program treats Web pages as strings and, after processing, highlights the changes directly in a merged document; as mentioned in [15], HTML*Diff* can consume significant memory and computation resources, and this might have an influence on the scalability of the tool. Tree models of Web pages also use diff techniques, but not at HTML page level, but at subtree (or node) content level, which has the advantage of making the computation less complex. For instance, WebCQ [24] uses HTML*Diff* to

inria-00588715, version 1 - 26 Apr 2011

detect change at *object* level. Here, the object represents in fact a part of a Web page specified for monitoring by the user, either by means of regular expressions or by marking elements of the HTML DOM tree like a table, list, link etc..

Another example of system that uses GNU diff tool is WebVigiL [18].

Edit scripting on strings.

The edit distance between two strings of characters represents the number of operations required to transform one string into another. There are different ways of defining an edit distance, depending on which edit operations are allowed: delete, insert, etc.. In string edit scripting, the atomic element is a single character and the cost is usually unitary, for every edit operation defined.

Root Mean Square.

Another notable way [38] of compute similarity is to use RMS(Root Mean Square) value, i.e., the quadratic mean. RMS represents a statistical measure for the magnitude of a varying quantity and permits therefore to quantify the change. If its value is small, then the difference between the compared elements is not significative. Often used in engineering to do an estimation of the similarity between a canonical model and an empirical one (in order to see the precision of the experiment), RMS formula needs numeric values of the model parameters. In the HTML context, ASCII values of the Web document's text characters are utilized in the canonical formula. Although a good idea, RMS measure applied for the ASCII values of each character has some drawbacks: first it does not take into account the hierarchies (it considers that every character has equal influence, independently of its position in the page), and second, it cannot take into account the semantics of context. Variants of this measure are presented in [39].

5.2 Matching of hierarchical models

Edit scripting on trees.

has the same formal definition as for strings, excepting for the fact that the basic entities are here tree elements (nodes or subtrees). Also, edit operations can occur at different levels, depending on the types of changes considered. As a consequence, cost models of edit operations become more complex. Each edit operation has a cost associated (cost proportional to the complexity of the operation, or based on heuristics of the model), so the execution of an edit script as a sequence of operations will return a cumulated cost. The similarity measure can be then computed as the inverse of this total cost. The interpretation is the following: less *unimportant* modifications we do to the first structure of data in order to make it isomorphic with its version, the more similar the two structures are.

This problem of determining the distance between two trees is referred to as the *tree-to-tree* correction problem and this is more in depth covered in [5]. Some works [9] report that edit scripting with moving operations is NP-hard. Usually, every model introduces some kind of model or computational heuristics that make the problem (a little) less difficult. As an example, [23] computes an edit scripting on branches. It can be also done on subtrees, rather than on complete trees, for the same complexity reasons. Concerning the cost of change operations, every technique makes its own decisions. MH-Diff [9] for example defines the costs as being moderated by some constants, all depending on the type of change.

Quantitative measures of change.

Various works [17, 3, 22] use a composed measure of similarity that tries to better adapt to the specific of the types of changes considered. The change is measured by a formula that incorporates three sub-measures of specific similarity: on the content (**intersect**: the percentage of words that appear in both textual content of subtrees), attributes (**attdist**: the relative weight of the attributes that have the same value in the model elements) and on the types of elements considered in the path (**typedist** emphasizes differences on the tag names when going up in the hierarchy). The final measure incorporates all above-defined types of similarity, together with some parameters that are meant to influence the importance of certain types of changes over others. The advantage of this measure is that it captures the symbiosis of different types of changes that occur in a certain way independently: content changes in leaf nodes, attribute changes in internal nodes; the third submeasure is more focused on the position of nodes in the structure.

Another quantitative measure of change is proposed in [23]. Here, a weighted measure that determines the magnitude of the difference between two ordered multisets (i.e., branches) is employed. In an ordered multiset, the weight of the **i**th node is defined as $(2^i)^{-1}$ (where i represents the depth of an element of the branch considered). Finally, the quantity of change is measured by computing the sum of the weights of the nodes that appear in the symmetric difference set.

6. STATISTICALLY ESTIMATING CHANGE

6.1 Motivating estimative models

We have seen the multitude of domains that are interested in the change detection issue. The general aim is to get an idea about the dynamics of a certain type of content, at a given granularity.

In Web crawler-related applications, the interest is more in whether a Web page has changed or not, in order to know if a new version of a Web page shall be downloaded or not. Only a binary response is needed; this does not happen because it is not interesting to make a distinction between the different types of changes that can occur, but because current crawlers treat information at Web page level. In this case, an estimation of the change frequency is as effective as explicitly computing it, as we show in Section 3. Indeed, if archive crawlers were more aware of the semantics of data they process, they could clearly benefit of a broader, richer insight into the data and could develop different strategies related to storage and processing. An estimation of the change rate, although not very descriptive (we usually do not know where the change appeared or its type), is still useful, especially when we can imagine a strategy that combines estimative and comparative methods of deriving dynamics. For this reason, we shortly present some of the existing statistical approaches.

6.2 Poisson model

The study carried out in [10] reports the fact that changes that occur in Web pages can be modeled as a Poisson process. A Poisson process is used to model a sequence of *random*

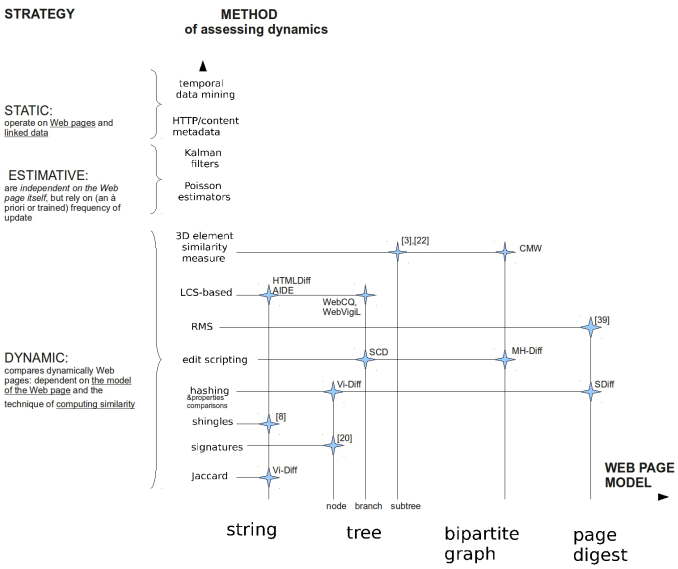


Figure 1: Summary of the presented approaches

events that happen *independently* with a fixed rate over time. Based on a(n ideally complete) change history, the frequency of change is estimated. The time-independence assumption in homogeneous models [10, 11] does not really capture the reality of the Web. The authors of [33] affirm that in the case of dynamic Web content (like blogs), the posting rates vary very much depending on different parameters. Hence, they propose an inhomogeneous Poisson model (which do not assume that events happen independently); this model learns the posting patterns of Web pages and *predicts* a recheck for new content.

The work in [11] formalizes some use cases where, by adapting the parameters of the Poisson model to the requirements of the application, a better accuracy of the estimation can be achieved. The situation when we do not have a complete change history of a Web page is treated, which is actually the real world case. Sometimes, we have only the last date of change or we at best just know that a change has occurred. Contributions are also related to the fact that the authors adapt the canonical model to interesting applications and feed different estimators, improving thus the technique for the considered cases.

6.3 Kalman filters

Other statistical approach to change detection is that of [7]. Here, the textual vector space model is employed to identify the patterns of the page and to train Kalman filters with these patterns. In the end, the change represents the event that does not match the prediction. The possible disadvantages of this method is that it assumes the linearity of the system (because of the the Kalman filter model, which is doing an exact inference in a linear dynamical system) and uses an incomplete vector space model (for complexity reasons).

7. OPEN QUESTIONS

We end this article by discussing several issues that have not been addressed yet and which are related to the deriving of temporal properties of Web pages. The selection of subjects reflects our personal vision on the topic.

Further studies on timestamping.

With the large number of sources of timestamping hints, it should indeed be possible to estimate the freshness of a Web page. An experimental study on the reliability of these sources, perhaps in specific contexts (a given Web server software, a given CMS, etc.), which could provide more insight into optimal strategies for timestamping Web pages, is still to be carried out.

Relevant change detection.

An important question when detecting changes is whether the changes are relevant to the interests or needs of the user or archivist. Web users are exposed to many advertisements and content becomes pre-fabricated, put in containers and delivered in a targeted way, so much more attention should be paid to the relevance factor. Additionally, Web pages have some components that are more dynamic than others; it is not possible to say if the dynamics come from irrelevant content (think of ads changing at every request) or precisely because the content is very informative (e.g. frequently updated news). To make the distinction between these, techniques that define and extract *semantics* (e.g., by looking at the topic and linked data) might be used as a filtering method or just for adding more significance to the change detection results. The cleaning of the Web page or its segmentation into semantic blocks is of interest in many information extraction fields and, although we mention only [40, 21, 29], there exists a large number of works that treat this subject.

As an observation, there exists a subtle difference in the use of the term "meaningful" in the various works that we have studied. While some works [23] use it to emphasize the fact that more types of changes are detected, other approaches [30] use it as synonym to "relevant", from the content point of view. Vi-Diff [30] uses the VIPS algorithm [40] to get from a Web page an hierarchy of semantic blocks and detect changes only from this perspective, hopefully ignoring all boilerplate content. However, a deeper insight into the relevance aspect is mentioned as future work; the authors of [30] talk about using machine learning techniques for this issue, which would be an interesting line of research. Usually, heuristics [40] are employed to get a measure of relevance because having a generic source of knowledge which can be automatically used for this task is very difficult.

Recently, [29] used the concepts that can be extracted from the description of Web feeds to get the content of interest from Web pages. However, this kind of semantics can only be obtained in the case of Web pages that have feeds associated.

With the emergence of the Semantic Web, we envision new ways of distinguishing timestamps or of filtering irrelevant content from Web pages and therefore more efficient methods for deriving dynamics of Web pages.

8. REFERENCES

[1] S. Abiteboul. Issues in monitoring web data. In *Proc. DEXA*, 2002.

[2] E. Adar, J. Teevan, S. T. Dumais, and J. L. Elsas. The Web changes everything: Understanding the dynamics of Web content. In *Proc. WSDM*, 2009.

[3] H. Artail and K. Fawaz. A fast HTML Web change detection approach based on hashing and reducing the

inria-00588715, version 1 - 26 Apr 2011

inria-00588715, version 1 - 26 Apr 2011

number of similarity computations. *Data Knowl. Eng.*, 66(2), 2008.

[4] R. Baeza-Yates, C. Castillo, and F. Saint-Jean. Web dynamics, structure, and page quality. In M. Levene and A. Poulovassilis, editors, *Web Dynamics*. Springer, 2004.

[5] D. T. Barnard, G. Clarke, and N. Duncan. Tree-to-tree correction for document trees. Technical Report 95-372, Queen's University, Kingston, Ontario, Canada, 1995.

[6] D. P. Bertsekas and D. A. Castañon. Parallel asynchronous Hungarian methods for the assignment problem. *INFORMS J. Computing*, 5(3), 1993.

[7] P. L. Bogen, II, J. Johnson, U. Karadkar, R. Furuta, and F. M. Shipman, III. Application of Kalman filters to identify unexpected change in blogs. In *Proc. JCDL*, 2008.

[8] A. Z. Broder. On the resemblance and containment of documents. In *Proc. SEQUENCES*, 1997.

[9] S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *Proc. SIGMOD*, 1997.

[10] J. Cho and H. Garcia-Molina. The evolution of the Web and implications for an incremental crawler. In *Proc. VLDB*, 2000.

[11] J. Cho and H. Garcia-Molina. Estimating frequency of change. *ACM TOIT*, 3(3), 2003.

[12] L. R. Clausen. Concerning Etags and datestamps. In *Proc. IWAW*, 2004.

[13] G. Cobéna and T. Abdessalem. A comparative study of XML change detection algorithms. In *Service and Business Computing Solutions with XML*. IGI Global, 2009.

[14] G. Cobéna, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In *Proc. ICDE*, 2002.

[15] F. Douglis, T. Ball, Y.-F. Chen, and E. Koutsofios. The AT&T Internet difference engine: Tracking and viewing changes on the Web. *World Wide Web*, 1(1), 1998.

[16] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of Web pages. In *Proc. WWW*, 2003.

[17] S. Flesca and E. Masciari. Efficient and effective Web page change detection. *Data Knowl. Eng.*, 46(2), 2003.

[18] J. Jacob, A. Sanka, N. Pandrangi, and S. Chakravarthy. WebVigiL: An approach to just-in-time information propagation in large network-centric environments. In M. Levene and A. Poulovassilis, editors, *Web Dynamics*. Springer, 2004.

[19] A. Jatowt, Y. Kawai, and K. Tanaka. Detecting age of page content. In *Proc. WIDM*, 2007.

[20] H. P. Khandagale and P. P. Halkarnikar. A novel approach for web page change detection system. *Intl. J. Comput. Theory Eng.*, 2(3), 2010.

[21] C. Kholschutter, P. Fankhauser, and W. Nejdi. Boilerplate detection using shallow text features. In *Proc. WSDM*, 2010.

[22] I. Khoury, R. M. El-Mawas, O. El-Rawas, E. F. Mounayar, and H. Artail. An efficient Web page change detection system based on an optimized Hungarian algorithm. *IEEE TKDE*, 19(5), 2007.

[23] S.-J. Lim and Y.-K. Ng. An automated change-detection algorithm for HTML documents based on semantic hierarchies. In *Proc. ICDE*, 2001.

[24] L. Liu, C. Pu, and W. Tang. WebCQ: Detecting and delivering information changes on the Web. In *Proc. CIKM*, 2000.

[25] J. Masanès, editor. *Web Archiving*. Springer-Verlag, 2006.

[26] Netcraft. January 2011 Web survey. `http://news.netcraft.com/archives/2011/01/12/january-2011-web-server-survey-4.html`, 2011.

[27] A. Ntoulas, J. Cho, and C. Olston. What's new on the Web? The evolution of the Web from a search engine perspective. In *Proc. WWW*, 2004.

[28] S. Nunes, C. Ribeiro, and G. David. Using neighbors to date Web documents. In *Proc. WIDM*, 2007.

[29] M. Oita and P. Senellart. Archiving data objects using web feeds. In *Proc. IWAW*, 2010.

[30] Z. Pehlivan, M. Ben Saad, and S. Gançarski. A novel Web archiving approach based on visual pages analysis. In *Proc. IWAW*, 2009.

[31] D. Rocco, D. Buttler, and L. Liu. Page digest for large-scale Web services. In *Proc. CEC*, 2003.

[32] S. C. Sahinalp and A. Utis. Hardness of string similarity search and other indexing problems. In *Proc. ICALP*, 2004.

[33] K. C. Sia, J. Cho, and H.-K. Cho. Efficient monitoring algorithm for fast news alerts. *IEEE TKDE*, 19(7), 2007.

[34] K. Sigurðsson. Incremental crawling with Heritrix. In *Proc. IWAW*, 2005.

[35] sitemaps.org. Sitemaps XML format. `http://www.sitemaps.org/protocol.php`, 2008.

[36] W3C. HTML 4.01 specification. `http://www.w3.org/TR/REC-html40/`, 1999.

[37] Y. Wang, D. J. DeWitt, and J.-Y. Cai. X-Diff: An effective change detection algorithm for XML documents. In *Proc. ICDE*, 2003.

[38] D. Yadav, A. K. Sharma, and J. P. Gupta. Change detection in Web pages. In *Proc. ICIT*, 2007.

[39] D. Yadav, A. K. Sharma, and J. P. Gupta. Parallel crawler architecture and web page change detection. *WSEAS Transactions on Computers*, 7(7), 2008.

[40] S. Yu, D. Cai, J.-R. Wen, and W.-Y. Ma. Improving pseudo-relevance feedback in Web information retrieval using Web page segmentation. In *Proc. WWW*, 2003.