Efficient and effective Web change detection

S. Flesca a, E. Masciari b,*

a DEIS, University of della Calabria, Via P. Bucci 41/C, 87036 Rende, Italy
b ICAR-CNR, Via P. Bucci 41/C, 87036 Rende, Italy

Received 24 April 2002; accepted 20 November 2002

Abstract

In this paper we present a new technique for detecting changes in Web documents. The technique is based on a new method to measure the similarity of two documents, that represent the actual and the previous version of the monitored page. The technique has been effectively used to discover changes in selected portions of the original document.

The proposed technique has been implemented in the CMW system providing a change monitoring service on the Web. The main features of CMW are the detection of changes on selected portions of web documents and the possibility to express complex queries on the changed information. For instance, a query can require to check if the value of a given stock has increased by more than 10%. Several tests on stock exchange and auction web pages proved the effectiveness of the proposed approach.

© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Update monitoring; Continuous queries; WWW tools

1. Introduction

Due to increasing number of people that use the Web for shopping or on-line trading, services for searching information and identifying changes on the Web have received renewed attention from both industry and research community. Indeed, users of e-commerce or on-line trading sites
frequently need to keep track of page changes, since they want to access pages only when their information has been updated.

Several systems providing change monitoring services have been developed in the last few years [4–6,12,14–16,22,23]. Generally, these systems periodically check the status of the selected web pages, trying to identify how the page of interest has been modified. The lack of a fixed data structure makes the problem of detecting, efficiently and effectively, meaningful changes on the web a difficult and interesting problem. Most of the web change detection systems developed so far are not completely satisfactory since they are only able to check if a page has been modified unregarding the type of changes occurred. For instance, the system Netmind can only detect changes on a selected text region, a registered link or image, a keyword, or the timestamp of the page [14]. There are several cases where this behaviour is not satisfactory: consider an auction on-line web page (e.g., eBay, Auckland, etc.), the user wants to be alerted only if a change occurs in one of the items he wants to buy, i.e. if the quotation of an article has been changed or if new items of the desired kind are available on the site.

Change detection systems should provide the possibility of specifying the changes the user is interested in: to select the region of the document of interest, the items inside the region whose changes have to be monitored, and conditions on the type of changes which must be detected. Systems detecting changes on HTML pages with fixed structure are not able to satisfy these kind of user needs since the page regions considered depend on the user’s request. Current techniques for detecting document differences are computationally expensive and unable to focus on the portion of the page that is considered interesting for the user [2,3,7,26].

A technique able to detect changes with a reasonable degree of efficiency and accuracy is necessary. The general problem of finding a minimum cost edit script that transforms a document into its modified version is computationally expensive (NP-hard) [1–3]. Techniques based on the computation of an approximate tree matching [17–21] are not completely satisfactory for this goal, since the type of changes they detects is too weak. However, in many application contexts, like the one considered in the above example, users are only interested in the changes made and not in the sequence of updates which produce the new document. For the on-line trading example, users are interested in the change on the quotation of a stock or in the insertion of new stocks, regardless of other changes to the page.

The approach presented in this paper differs from all the previously cited approach. The main idea underling our technique is that of paying attention only to how much the changes have modified the document under observation, instead of looking for the exact sequence of changes that produced the new document. Our technique represents the document as a tree and permits the user to focus on specific portions of it, e.g. sub-trees. In particular the aim of our technique is that of efficiently monitor changes on small portion of a web page, for instance an item of a list that represents the information on a particular flight. However, it will be clear in the following that our technique is able to detect even sophisticated changes, like that produced by move operations.

The paper also describes the architecture of a system, called CMW, which allows to create web update triggers that monitors web page of interest (WebTriggers). A web update triggers allow the CMW users to specify the type of changes they are interested in, and the actions that has to be performed when changes are detected.
The main contributions of this paper are:

- the definition of a new efficient technique that allows users to measure Web document differences in a quantitative way,
- the definition of a language to specify web update triggers,
- a system for change detection on the web.

2. Web changes monitoring

In this section we define an efficient technique for the efficient detection of interesting changes in web documents. We are mainly interested in changes that add, delete or update information contained in specific portions of a Web page. To better understand our approach to the change detection problem, let us consider a web page containing information about the staff of the ‘Database and Artificial Intelligence’ (DBAI) group at the Technical University of Vienna. We want to detect changes that involve people in the staff of DBAI. However not all the possible changes have to be managed in the same way: a room change or the change of a telephone number is not relevant for an user interested in the recruiting or dismissal of a member of the staff. Fig. 1 shows the changes occurred in the DBAI staff page in the last year.

A change detection system must provide facilities that allows users to monitor the changes that happen to this list of people and react in different way to the different changes. Therefore, we

Fig. 1. Two versions of the DBAI Staff page.
propose a technique that permits to detect changes only in a specific area of a web page, depending on the user interest. To specify the information that has to be monitored, the user selects the region of the document of interest (a sub-tree of the document tree) and names some items inside the region (sub-trees of the previously selected sub-tree) whose changes have to be monitored, he can also specify a complex condition on the type of changes that have to be detected. As an example, consider a user that wants to be notified when the price of a stock is quoted at more than 110% of its old value. The user selects the region of interest (e.g., the row of a table containing the information about the stock quote), the value of the stock and specifies that he wants to be alerted to if this value increases by more than 10%. The system has to identify first the region of interest (i.e., the portion of the document that is most similar to the region selected in the old version) and to verify, for each item the associated conditions.

To retrieve the sub-tree of interest in the updated document it is necessary to define a similarity measure between document sub-trees. The similarity of two trees is defined by considering the similarities of their sub-trees. It is worth noticing that the use of the minimum cost edit script to detect changes is not feasible for this kind of application since it is computationally expensive when complex edit operations are considered [2]. However, our technique use a similar change model: it can be seen as the computation of an edit script characterized by an explicit null cost for the “move” operation. If a node of the tree is moved, it cause a change only on the information associated to its ancestors.

The similarity measure is defined by considering the complete weighted bipartite graph \((N_1, N_2, E)\) where \(N_1\) and \(N_2\) are the nodes of the two sub-trees and the weight of each edge \((x, y) \in E\) represents the similarity of \(x\) and \(y\). The similarity of two trees is defined by considering the associations of nodes (edges of the bipartite graph) which give the maximum degree of similarity. The constructed association is then used to obtain quantitative information about changes.

2.1. Document model

Several different data models for Web documents have been recently proposed. For instance, the WWW Consortium (W3C) has proposed a “generic” model, named Document Object Model (DOM), which defines a set of basic data structures that enable applications to manipulate HTML and XML documents.

In this work we represent structured documents as unordered labelled trees [1–3], i.e. we do not consider the order of appearance of elements in the document, but only the containment of elements inside each other. Generally each node of the tree corresponds to a formatting HTML element in the document. Observe that practical model for representing HTML document (e.g., Java Swing HTML/DOM) label each node with its tag name. Formatting elements specify that the text enclosed by a pair of start and end tags have to be formatted in a certain way (\(<\text{table}>\ldots/<\text{table}>\), \(<\text{ul}>\ldots/<\text{ul}>\ldots). This assumption follow from the observation that formatting elements are generally used to distinguish some information by emphasizing it. Moreover, the information associated to each element is given by the attributes specified in its start tag, the text enclosed by its start tag and its end tag, and the type of tag characterizing the element.

The document model used by our technique, permit to represent all of these informations. It is defined in a formal way as follows. We assume the presence of an alphabet \(\Sigma\) of content strings, of a set of element types \(\tau\), that contains the possible structuring markup, and a set of attribute names \(A\).
Definition 1 (Document tree). A document tree is a tuple $T = \langle N, p, r, l, t, a \rangle$, where

1. $N$ is the set of nodes of the tree,
2. $p$ is the parent function associating each node (except the root $r$) of the tree with its parent,
3. $r$ is the distinguished root of $T$,
4. $l$ is a labelling function from leaf ($T$) to $\Sigma^+$,
5. $t$ is a typing function from $N$ to $\tau$ and
6. $a$ is an attribute function from $N$ to $A \times \Sigma^*$.

Thus, a document tree is an unordered tree whose nodes (elements) are characterized by their markup type and the associated set of attribute-value pairs. Leaf nodes have associated the actual textual content of the document. Given a document tree $T$, whose root is $r$, and a node $e_n$ of $T$, we denote with $T(e_n)$ the sub-tree of $T$ rooted at $e_n$.

However, to be suitable for a change detection tools, this model should be extended by associating more information to internal nodes of the tree. We define two functions characterizing an element w.r.t. the whole document tree, type $(e_n)$ and $w(e_n)$. If $r, e_2, \ldots, e_n$ is the path from the root $r$ to the element $e_n$, type $(e_n) = t(r)t(e_2)\cdots t(e_n)$, whereas $w(e_n) = \{s|s$ is a word $1$ contained in $l(e) \wedge e \in$ leaf $(T(e_n))\}$. We also define $a(e_n)$ as the set of attributes associated to $e_n$. Essentially $w(e_n)$ is a set of words contained in the various text strings associated to the leaves of the sub-tree rooted at $e_n$, and type $(n)$ is the concatenation of type label in the path starting from the root of the tree and ending in $e_n$, i.e. the complete type of the element.

Example 1. Consider the portion of an HTML document shown in Fig. 2. It corresponds to the HTML document tree shown in the same figure, where for each node are reported the corresponding HTML tag, and attributes (text is not shown for nonleaf elements). We have the following values for the root element $r$ of this sub-tree: $w(r) = \{\text{This, is, an, example}\}$, type $(r) = \{\text{table}\}$, $a(r) = \{\emptyset\}$. For the node $p$ relative to the first paragraph, say $p$, we have: $w(p) = \{\text{This, is}\}$, type $(p) = \{\text{table.tr.td.p}\}$, $a(p) = \{A\}$.

Our change detection algorithm works on this tree structure exploiting the information contained in each node. Observe that each node keeps track of its level in the tree (the type name), its content and the content of its sons and its attributes. Experimental results show that this information is the only one necessary to estimate the similarity of document elements.

2.2. A tree similarity measure

To effectively detect changes in a selected portion of a web page, we first have to retrieve this portion of the document in the new document version. Since in the new version of the web page text can be added or removed before and after the portion of the document we are interested in, we cannot rely on its old position in the document tree to perform this task, and consequently, we

---

1 A word is a substring separated by blank to the other substring.
have to find the portion of the new document that is the most similar to the old one. One possibility to perform this task is to compute the minimum cost edit script between the original document and its current version [2,3]. However these techniques are not completely satisfactory for this task. Indeed, a system that builds an edit script producing the new document from the old one, does not pay particular attention to the monitored information. Thus, acting in this way, this information could be mismatched. Moreover the information to be monitored usually represents a small piece of the original page (i.e., an item of a list) thus looking directly for it in such a case can result in a significant performance improvement.

We have to define a similarity measure between document (sub)trees in a way that it should be possible to compute it efficiently, and the measure must be normalized, allowing the comparison of different pairs of trees and the selection of the most similar one. In order to define the similarity between sub-trees, we first associate each element of the selected sub-tree to its current version in the new sub-tree, and then consider the similarity degree of the two sub-trees w.r.t. this association. So, we first have to define a measure of similarity between single elements and then use it to define a similarity measure between whole (sub)trees.

Given a document tree $T = \langle N, p, r, l, t, a \rangle$ and an element $r'$ of $N$, the characteristic of $r'$ ($\psi(r')$) is a triple $\langle \text{type}(r'), a(r'), w(r') \rangle$. The similarity measure of two elements is defined on the basis of the similarity between each component of the characteristics of the elements being considered. We define the following functions measuring similarity between the different components of element characteristics. Given two trees $T_1$ and $T_2$, and two nodes $r_1 \in T_1$ and $r_2 \in T_2$ we define:

$$\text{intersect} \left( w(r_1), w(r_2) \right) = \frac{|w(r_1) \cap w(r_2)|}{|w(r_1) \cup w(r_2)|}$$

$$\text{attdist} \left( a(r_1), a(r_2) \right) = \frac{\sum_{a_i \in \{a(r_1) \cap a(r_2)\}} \text{Weight}(a_i)}{\sum_{a_i \in \{a(r_1) \cup a(r_2)\}} \text{Weight}(a_i)}$$
possible transformations, since it is not probable that complex transformations correspond to an effective document transformation. Also we do not want to consider all the possible associations can be considered valid since node association has the same value in \( r_1 \) and \( r_2 \) w.r.t. all the attributes in \( r_1 \) and \( r_2 \). The attributes are weighted differently by using the Weight function because some attributes are generally considered more relevant than other, for instance the attribute “href” is considered more relevant than formatting attributes, like “font”. The Weight function could be defined by simply associating to each different tag a weight corresponding to the relevance of the tag in the chosen model.

The definition of the function \( \text{intersect}(w(r_1), w(r_2)) \) returns the percentage of words that appear in both \( w(r_1) \) and \( w(r_2) \). The function \( \text{attdist}(a(r_1), a(r_2)) \) is a measure of the relative weight of the attributes that have the same value in \( r_1 \) and \( r_2 \) w.r.t. all the attributes in \( r_1 \) and \( r_2 \). The attributes are weighted differently by using the Weight function because some attributes are generally considered more relevant than other, for instance the attribute “href” is considered more relevant than formatting attributes, like “font”. The Weight function could be defined by simply associating to each different tag a weight corresponding to the relevance of the tag in the chosen model.

The definition of the function \( \text{typedist}(\text{type}(r_1), \text{type}(r_2)) \) take care of the difference between the complete types of element, differences in the tag names are considered less relevant going up in the containment hierarchy of the two elements. Consider for example the tree in Fig. 2 we assume that a change in the first level (e.g., deleting a row in the table) has a great influence when characterizing the structure of the document w.r.t. a change in the paragraph contained in it (e.g., the leaf in the previous tree). In the given formula \( \text{suf} \) represents the length of the common suffix between \( \text{type}(r_1) \) and \( \text{type}(r_2) \) and \( \text{max} \) denotes the maximum cardinality between \( \text{type}(r_1) \) and \( \text{type}(r_2) \). The similarity between two document elements is defined as follows.

**Definition 2 (Element similarity).** Given two document trees \( T_1 \) and \( T_2 \) and two elements \( r_1 \in T_1 \) and \( r_2 \in T_2 \) having characteristics \( (\text{type}(r_1), a(r_1), w(r_1)) \) and \( (\text{type}(r_2), a(r_2), w(r_2)) \), the similarity of \( r_1 \) and \( r_2 \) \( (\text{CS}(r_1, r_2)) \) is defined as

\[
\text{CS}(r_1, r_2) = -1 + 2 \times (\alpha \times \text{typedist}(\text{type}(r_1), \text{type}(r_2))) + \beta \times \text{attdist}(a(r_1), a(r_2)) + \gamma \times \text{intersect}(w(r_1), w(r_2)))
\]

where \( \alpha + \beta + \gamma = 1 \).

The value of \( \alpha, \beta, \gamma \) can be selected on the basis of the type of changes to detect (see Section 5.1). Clearly the similarity coefficient takes values from the interval \([-1, 1]\), where \(-1\) corresponds to the maximum difference and \(1\) to the maximum similarity. A element that is deleted (respectively inserted) is assumed to have similarity \(-1/k\) with elements of the new (respectively old) document, where \( k \) is a fixed constant that defaults to 2 (this value has been chosen by experimental evidence).

### 2.3. Detecting document changes

Once we have defined element similarity, we can complete the definition of our technique. To compare two document sub-trees, we consider the complete weighted bipartite graph \( (\langle N_1, N_2 \rangle, E) \) where \( N_1 \) and \( N_2 \) are, respectively, the nodes of the two sub-trees; the weight of each edge \((x, y) \in E \) is \( \text{CS}(x, y) \). We use this weighted graph to establish what is the association between elements belonging to the old and new version of the document that gives the maximum similarity degree. Obviously not all the possible associations can be considered valid since node association must correspond to an effective document transformation. Also we do not want to consider all the possible transformations, since it is not probable that complex transformations correspond to
2.3.1. Document mappings

All the possible changes that can occur in a document must correspond to an association between the nodes in the document tree of the original pages and the nodes in the document tree of the newest pages.

As stated above, not all the associations can be considered valid. For instance, if we do not permit paragraph splitting or joining then only one to one associations are valid. In our model, we do not consider *glue or copy* operations [2,3] since they seem to be not relevant in this context. Indeed, in most web change systems these kind of operations are not considered because they happen very rarely unless we consider relevant the movement of a portion of the document inside the same page, while they are relevant when considering different kind of documents such as book chapters or web guide. Moreover we make the assumption that we do not want to reconstruct the complete sequence of changes but we only want to give a quantitative measure of the changes occurred in the document.

In general we are interested in associations that correspond to some type of *editing* of the document that add, change or delete some meaningful information in the document, for instance the text of a paragraph or the destination of a hypertext link. Before defining valid *edit mapping* we introduce some notation. Given two document trees $T = \langle N, p, o, r, l, t \rangle$ and $T' = \langle N', p', o', r', l', t' \rangle$. A *Tree Mapping* from $T$ to $T'$ is a relation $M \subseteq N \times N'$, such that $(r, r') \in M$. Given two document trees $T$ and $T'$, a tree mapping $M$ from $T$ to $T'$ and a node $x$ in $N$, we denote with $M_x$ the set of nodes of $N'$ associated with $x$ in $M$; analogously, given a node $y$ in $N'$ we denote with $M_y$ the set of nodes of $N$ associated with $y$ in $M$.

**Definition 3** (*Edit mapping*). Given two document trees $T = \langle N, p, o, r, l, t \rangle$ and $T' = \langle N', p', o', r', l', t' \rangle$. An edit mapping $M$ from $T$ to $T'$ is a tree mapping such that $\forall x \in N$ if $|M_x| > 1$ then $|M_y| = 1$ for each $y$ in $M_x$.

Intuitively if $|M_x| > 1$ the original node has been split while if $|M_y| > 1$ many nodes in the original tree have been merged. A mapping between two trees $T$ and $T'$ is said to be Simple if it associates each node in $T$ with at most one node in $N'$ and each node in $T'$ with at most one node in $N$. Given two document trees $T$ and $T'$ and a tree mapping $M$ we denote with $\text{ext}(M)$ the set of mappings $M'$ from $T$ to $T'$ such that $M \subseteq M'$. The number of valid edit mapping may be very large due to the completeness of the graph, but we can strongly reduce the number of edges to be considered for the mapping. This can be done by considering the edges that have a weight greater than a predefined threshold.

**2.3.1.1. A cost model for mapping.** Once we have defined the valid association between document trees we have to define the cost of these associations, that is: if we consider the nodes in the new sub-tree as the new version of the associated nodes in the original sub-tree, how similar can we consider the new document sub-tree to the old one?
Since an element of a given document tree can be associated to a group of elements in the new document, to define document similarity with respect to an edit mapping we need to define first the notion of element similarity.

The similarity of two documents \( D_1 \) and \( D_2 \) is computed on the basis of the similarity between elements of \( D_1 \) and \( D_2 \). The similarity of one element in \( D_1 \) and the group of elements in \( D_2 \) is computed using the weights of the arcs in the bipartite graph selected by the edit mapping.

**Definition 4.** Given two document trees \( T_1, T_2 \) and two sub-trees \( T'_1, T'_2 \) and an edit mapping \( M \) from \( T'_1 \) to \( T'_2 \), the similarity of \( x \in N_1 \) w.r.t. \( M \) is defined as follows:

\[
\text{Sim}_M(x) = \begin{cases} \frac{\text{avg}_{(x,y) \in M} \text{CS}(x,y)}{|\{(x,y) \in M\}| > 0} & \text{if } \{|(x,y) \in M| > 0\} \\ -\frac{1}{k} & \text{otherwise} \end{cases}
\]

Thus, given a bipartite graph \((N_1, N_2, E)\), Definition 4 computes the similarity of a node \( x \) in \( N_1 \) by considering the average of the similarities of the pairs of elements \((x, y)\) for each \( y \) related to \( x \) by the edit mapping. Using the previous definition we can now define the concept of similarity among document sub-trees.

**Definition 5.** Given two document trees \( T_1, T_2 \), two sub-trees \( T'_1 \) of \( T_1 \), \( T'_2 \) of \( T_2 \) and an edit mapping \( M \) from \( T'_1 \) to \( T'_2 \), the similarity of \( T'_1, T'_2 \) w.r.t. \( M \) is defined as follows:

\[
\text{Sim}_M(T'_1, T'_2) = \frac{\sum_{x \in N_1 \cup N_2} \text{Sim}_M(x)}{|N'_1| + |N'_2|}
\]

Finally, we define document sub-tree similarity considering the similarity obtained by the edit mapping that maximizes the similarity between two sub-trees.

**Definition 6 (Tree similarity).** Given two document trees \( T_1, T_2 \), two sub-trees \( T'_1 \) of \( T_1 \), \( T'_2 \) of \( T_2 \), and letting \( M \) be the set of possible mappings from \( T'_1 \) to \( T'_2 \). The similarity coefficient of \( T'_1, T'_2 \) is defined as

\[
\text{Sim}(T'_1, T'_2) = \max_{M \in M} \text{Sim}_M(T'_1, T'_2)
\]

### 2.3.2. Searching for the most similar sub-tree

Once we have defined similarity between document sub-trees we can approach the problem of detecting document changes. Here we consider only simple changes, i.e. changes that are detectable using a simple mapping; these changes are insertion, deletion, or textual modification. Note that some types of move operations are also detectable; in particular changes that move an element \( e \) from the sub-tree rooted in the parent of \( e \) to another sub-tree that is not contained in \( T(e) \) and does not contain \( T(e) \).

Before presenting the technique used to detect changes we introduce the concept of similarity graph, that will be used in the algorithm searching for the most similar sub-trees. Given two document trees \( T_1 = (N_1, \{p_1, r_1, l_1, t_1, a_1\}) \) and \( T_2 = (N_2, \{p_2, r_2, l_2, t_2, a_2\}) \), the similarity graph associated to \( T_1, T_2 \) (denoted \( WG(T_1, T_2) \)) is a weighted bipartite graph \( (N', E) \) where \( N' = N_1 \cup N_2 \) and \( E \) is the set of weighted edges defined as follows: \( E = \{(x, y, \text{CS}(x,y)) | \forall x \in N_1, y \in N_2\} \).
Furthermore given a similarity graph \( WG(T_1, T_2) = (N, E) \), we define the projection of a similarity graph on a set of nodes \( N' \subseteq N \) as \( \pi_{N'} WG(T_1, T_2) = (N', \{(x,y,c) | (x,y,c) \in E \land (x \in N' \lor y \in N')\}) \), i.e. the sub-graph representing the piece of the document to be monitored. To define the algorithm that for a given sub-tree finds the most similar sub-tree in the new document, we refer to the maximum matching problem. Indeed, given two document sub-trees \( T_1 = (N_1, p_1, r_1, l_1, t_1, a_1) \) and \( T_2 = (N_2, p_2, r_2, l_2, t_2, a_2) \), the following lemma assures that we can use the Hungarian algorithm to compute a simple edit mapping between two sub-trees.

**Lemma 1.** Given two document trees \( T_1, T_2 \) and two sub-trees \( T_1', T_2' \), and a simple edit mapping \( M \) between \( T_1' \) and \( T_2' \), then \( \text{Sim}_M(T_1', T_2') = \text{Sim}(T_1', T_2') \) if \( M \) is a maximum weight matching on \( WG(T_1', T_2') \).

**Proof.** By definition, we know that the maximum similarity simple edit mapping will contain only simple associations between nodes in \( T_1' \) and \( T_2' \), i.e., for each node at most one edge leaving it will exist. This assures that for each node \( x \), with an outgoing edge belonging to the mapping \( M \), the following equivalence \( \text{avg}_{(x,y) \in M} CS(x,y) = CS(x,y) \) holds.

Therefore,

\[
\sum_{x \in N_1 \cup N_2} \text{Sim}_M(x) = \sum_{(x,y) \in M} CS(x,y) \quad \square
\]

Since computing the maximum similarity edit mapping corresponds to computing the maximum weighted matching on \( WG \) we can use the Hungarian algorithm to compute the maximum edit mapping in the algorithm for change detection. Indeed, the Hungarian algorithm [9] computes correctly the optimal complete matching of a complete bipartite graph having a complexity of \( O(N^3) \). Moreover the following lemma assures that we can prune the new document tree by deleting irrelevant sub-trees when looking for the sub-tree maximally similar to the one being monitored.

**Lemma 2.** Let \( T_1 = (N_1, p_1, r_1, l_1, t_1, a_1) \), \( T_2 = (N_2, p_2, r_2, l_2, t_2, a_2) \) be document trees and \( T_1' = (N_1', p_1', r_1', l_1', t_1', a_1') \) a sub-tree of \( T_1 \). Given a sub-tree \( T_2'' = (N_2', p_2', r_2', l_2', t_2', a_2') \) of \( T_2 \) with \( |N_2'| > (k+2) \times |N_1'| \) there exists a sub-tree \( T_2' = (N_2', p_2', r_2', l_2', t_2', a_2') \) of \( T_2 \) with \( |N_2'| \leq (k+2) \times |N_1'| \) such that \( \text{Sim}(T_1', T_2') > \text{Sim}(T_1', T_2'') \).

**Proof.** Let us consider as sub-tree \( T_2' \) an empty sub-tree \( (N_2' = \emptyset) \). \( \text{Sim}(T_1', T_2') \) is equals to \(-|N_1'|/k\) since we must consider the cost of the deletion of all the nodes in \( T_1' \) (remember that deleting a node costs \(-1/k\)).

Consider now the sub-tree \( T_2'' \). Since \( |N_2''| > |N_1'| \), any edit mapping can associate only \( |N_2''| - |N_1'| \) nodes because each node in \( N_1' \) can be associated to a single node in \( N_2'' \). In the best case the associated nodes have similarity degree 1, thus we have a similarity \( \text{Sim}(T_1', T_2'') = |N_1'| - (|N_2''| - |N_1'|) / k \). However, since \( |N_2''| > (k+2) \times |N_1'| \)

\[
\text{Sim}(T_1', T_2'') \leq |N_1'| - \frac{(k+2) \times |N_1'| - |N_1'|}{k} = \frac{k \times |N_1'| - (k+2) \times |N_1'| + |N_1'|}{k} = \frac{-|N_1'|}{k} = \text{Sim}(T_1', T_2')
\]

Thus, the lemma holds. \( \square \)
The algorithm that finds the most similar sub-tree is shown below.

**Algorithm 1 (Finding the most similar sub-tree).**

**INPUT** A stored document sub-tree \( T_0 = (N_0, p_0, r_0, l_0, t_0, a_0) \) and a new document \( T_1 = (N_1, p_1, r_1, l_1, t_1, a_1) \)

**OUTPUT** a sub-tree \( T_2 \) of \( T_1 \) to be monitored and a mapping \( M \) among \( T_0, T_2 \)

Construct the similarity graph \( G = WG(T_0, T_1) \)

Set \( \text{Max Sim} = 0, \ M = \emptyset \)

For each \( n \in N_2 \) do begin

let \( T(n) = (N_2, p_2, r_2, l_2, t_2, a_2) \)

if \( |N_2| \leq |N_1| \times 4 \) then begin

\( G' = \pi_{N_1 \cup N_2} G \)

\( M' = \text{Maximum matching}(G') \)

\( AS = \text{Sim}(T_0, T(n)) \)

if \( AS > \text{Max Sim} \) then \( \text{Max Sim} = AS, \ T_2 = T(n), \ M = M' \)

end

end

return \( M, T_2 \)

The Algorithm 1 searches for the sub-tree most similar to the one being monitored in the new document and returns the simple edit mapping for which these two sub-trees have the maximum similarity value. We now states the complexity of our algorithm.

**Theorem 1.** Given a document tree \( T_1 = (N_1, p_1, r_1, l_1, t_1, a_1) \) and a sub-tree \( T_0 = (N_0, p_0, r_0, l_0, t_0, a_0) \) to be monitored. Let the document tree \( T_2 = (N_2, p_2, r_2, l_2, t_2, a_2) \) be the new version of \( T_1 \), Algorithm 1 finds the most similar sub-tree in \( O(|N_2| \times |N_1|) \) time.

**Proof.** Lemmas 1 and 2 assure the correctness of the algorithm. To verify the complexity bound consider that for each sub-tree \( T_2 = (N_2, p_2, r_2, l_2, t_2, a_2) \) with \( |N_2| \leq 4 \times |N_1| \) (by using the default value \( k = 2 \), we must compute the Hungarian Algorithm to find the maximum matching between \( T_0, T_2 \). By Lemma 2 this can be done in \( O(|N_1|^3) \) time and this operation has to be performed for a number of sub-trees that is proportional to \( |N_2| \). \( \square \)

This result emphasize the efficiency of our technique since in general \( N_1 \ll N_0 \) and \( N_1 \ll N_2 \). The efficiency is also confirmed by the execution times of the experimental tests, as reported in Section 5.

3. Web update queries

To better exploit the change detection technique defined in the previous section, we need to provide the possibility of specifying general conditions on data being observed. For instance, reconsider the DBAI staff page shown in Section 2; an user may want to be alerted only if a new
professor is recruited. A trigger that perform this task requires to monitor the professors list, and to check if a new piece of information that contains an email link is added.

In this section we introduce a language to specify this type of triggers, named *Web update queries* (or *web trigger*). A web update query allows the user to select some specific portions of the document, that will be monitored (we refer to these portions as *target-zones*). These are the portions of the document where the information which is considered relevant is contained. Inside this zone you can specify a set of sub-zones, named targets. When specifying the trigger condition, the user can ask for verification if the information in a target has been modified. Usually, each target is a leaf of an HTML Tree that is considered relevant by the user.

Once the user has specified the targets of the web update query, he can choose the preferred notification method, e-mail or in site notification (an alert in the personal monitoring application). Finally the user can set the time interval for query evaluation and the validity period of the web update query. Web update queries are expressed using the syntax sketched in Fig. 3.

In Fig. 3 (HTML sub-tree) represents a sub-tree of the document representation discussed in the above section. Note that web update queries should be specified using a visual interface, since it is the best way to specify the (HTML sub-tree) involved in the trigger definition. The non terminal symbol *(target-condition)* in the trigger syntax table represents simple boolean conditions that can be used in the *when* clause. In particular when specifying conditions in the *when*
clause it is possible to access both the old and new target values using the NEW and OLD properties of target items, as shown in the example below.

Furthermore you can cast target item type from the predefined string type to number and date type. Using the CREATE WebTrigger command a user can create a web trigger on the CMW personal server, that handles change detection on his behalf. Once the query has been created the user can set the values for the parameters $a$, $b$, $c$ previously described by using a frame that is automatically prompted to the user. The server maintains a local copy of the target zones to be monitored and a list of the target predicates that can fire user notification.

For instance a trigger that check for the dismissal of Prof. Leone is the following CREATE
WebTrigger dbaiTrigger ON "DBAI staff" STAFF INSIDE http://www.dbai.tuwien.ac.at/staff/CHECK "LEONE" STAFF.LEONE NOTIFY BY alert WHEN STAFF.LEONE DELETED BETWEEN 17-11-2000 AND 17-11-2001 EVERY 1440

4. System architecture

In this section we describe the evaluation process of web update queries in the CMW system that allows users to specify and execute web update queries using a visual interface. Change detection results are shown when triggers are raised.

The system is implemented in java, and HTML documents are represented using the swing document libraries, which uses a document model (Swing DOM) very close to the document model discussed above. The architecture of the system is reported in Fig. 4 and consists of five main modules are: the change monitoring service, the query engine, the change detection module, the query builder and the change presentation module.

The system is composed of two main applications, a visual query editor, that handles query specification, and an active query engine, that evaluates web update queries. The system maintains an object store where the objects describing the currently active web update queries are serialized. Each query object maintains information about the list of target zones (document sub-trees) referred to in the query, and for each target zone the list of targets contained inside that zone.

Next we describe the behaviour of CMW by explaining how each module behaves.

4.1. Change monitoring service

This module is devoted to schedule user queries evaluation. Essentially it maintains a queue of active Web update queries and select those queries whose condition has to be checked. It deserializes query objects out of the query database and sends them to the query engine to be evaluated.

4.2. Query engine

The query engine coordinates the whole evaluation process of a web update query. For each web source (URL) involved in the query, it sends the different target zones relative to specified
URL to the change detection module. Then it waits to receive the resulting edit mapping for each target zone. Once the edit mappings has been received it checks the conditions specified in the when clause and, if true, it sends a notification request to the change presentation module.

Notification requests contain change notification objects, one for each target zone in the query. A change notification object contains (1) the references to the old and new document trees, and (2) an edit mapping for each target zone.

To speed up query evaluation, the query engine schedules the change detection process of the various documents involved in the queries. Indeed, to avoid delays caused by slow downloads, it activates several instances of the change detection modules and if possible starts evaluating the when clause as soon as change detection results became available.

4.3. Change detection module

This module performs the real task of change detection on a web page. First it downloads from the internet the new version of the page of interest, and then it tries to find the portions of the new web page that are the most similar to the portions of the old document that appear in the web update query.

For each one of the target zones, it runs Algorithm 1 once for each sub-tree of the new document, selecting the one having the highest similarity value. The selected sub-tree and its edit mapping, are associated to the target zone. Note that since all the target zones inside the same
document are processed by the same instance of the change detection module the document is downloaded, parsed and analyzed only once.

4.4. Query builder

An user should not to be aware of the internal representation of an HTML document. For this reason, the system provides a visual interface that supports query formulation. In particular, a user has to first select the desired zones inside the documents that have to be monitored, and then for each zone he selects the target items. The query builder interface used for the selection of target zone is shown in Fig. 5.

Next the user defines the when clause, the type of notification and the query validity interval, according to the syntax for Web Trigger specified above. The interface used to specify valid time interval is shown in Fig. 6 and it is a step of the web trigger composition.

Fig. 5. The CMW query builder interface.
4.5. Change presentation module

This module, when a trigger raises a notification request, presents the pieces of information being observed and the changes which have occurred placed side by side to the user. This structure allows easy recognition and navigation of changes detected by the system as shown in Fig. 8.

Example 2. Consider an auction site and assume that we have a bid on the item “AMD Athlon”, we are interested in monitoring its price and we want to be advised if someone tries to surpass our offer.

By using the system CMW it is possible to monitor the specific item. In particular, the system takes the two versions of the Web pages as input and produces as output a measure that quantify changes that have been made to the portion of the page of interest. To choose this portion we select the piece of document containing the row of the item “AMD Athlon”, naming it AMDtarg, and then we select the portion of text containing its price and name it AMDtarg.price. Finally we write the Web trigger shown in Fig. 7. In the trigger we use the terms “AMD Athlon” and “Price” to identify the HTML sub-tree associated to the whole item and to its price.

When a change is detected we are alerted so that we can verify which change has occurred. Fig. 8 shows the output produced by the system for the web trigger of Fig. 6.
The system discovers the change occurred in the price and displays the pages side by side scrolling the old page and the new page to compare them at the points where the change has occurred (the changed item is underlined).

5. Experimental results

Before presenting the results of the experiments we have performed to prove the effectiveness of our approach, we have to clarify one aspect of the algorithm that has remained unexplained in the above discussion: what are the values of the parameters $a$, $b$, and $c$ used in the definition of the element similarity formula.

To assign effective values to them, we performed some experiments to observe the variation of change detection accuracy depending on the values of these parameters. Once these parameters have been chosen in the best possible way, according to the tests performed, we test the real effectiveness of the approach by also performing other experiments to measure the execution time and the detection accuracy of the algorithm when applied to various categories of Web documents.

5.1. Tuning parameters

As explained above the element similarity function is essentially a weighted average of the three functions ($intersect$, $attdist$, $typedist$) weighted by the coefficients $a$, $b$ and $c$. We performed several experiments on a set of 10 Web documents, relating to auction bid and stock quotes.
In particular, for each web page in the set we selected five modified versions and executed three different web update queries on it. For each triple of values \((a, b, c)\), we report the average accuracy obtained in the experiments, i.e. the accuracy level is an average of the results of the experiments, where successful experiments are weighted 1 and failing experiments are weighted 0. (In Fig. 9 each graphic reports for a given value of \(c\) the variation of \(a, b\) values can be obtained considering that \(a + b + c = 1\).) The results of the sensitivity analysis leads to the following values for the weight factors: \(a = 0.1, b = 0.1, c = 0.8\). These are the default values used in the system.

However, in this test set we want to be notified about the variation in bid prices and stock quotation, practically we are more interested in changes in the contents of the items than in structural changes. So, in this case the function that gives best results when a high weight is assigned to it is the \textit{intersect} function that is weighted by \(c\) coefficient.

Similar experiments have been performed on a set of pages relative to on-line newspaper and libraries, in this case the information we are interested in is relative to structural (attribute) changes so we observe that \textit{attdist} and \textit{typedist} should have a greater value. Thus, we choose to provide the user with a multiple predefined choice on the selection of these weights, depending on the type of the web update query (structural, semantic, mixed).

5.2. Monitoring pages results w.r.t. time

In this section we present some statistics relative to the execution time of web update queries. We performed several experiments on different kind of web pages and in this section we show the experiments on 10 test pages as reported in Fig. 10 and we made 100 tests on each of them. This pages are particularly interesting since they contain an high number of items rapidly changing.
reported (1) the average time interval which elapsed between the check request and the end of query evaluation, and (2) the detection accuracy as the number of successful changes detected. All the test of this section have been run on a Pentium IV 1700 MHz with 512 MB of Ram. Note that the execution time does not include the download time of the document since tests are performed on local copies of the pages, predownloaded from the Web site, so in this case our system is working like a personal web update server. Note that the size of the pages being monitored is very high, this cause that the graph has $N_1 > 1000$ and $N_2 > 1000$ but the algorithm works very quickly as well. The changes that are not detected are all *false positive*, i.e. some changes have been made but they are not reported. As a further example Fig. 11 shows the output of the system when multiple pages are monitored at the same time. In this figure we show the list of some items being monitored at the same time. We refer to this list as *watch-list* and it contains the information relative to the web pages that contain the item being monitored, the start date for monitoring and the last check time. When the time interval expires a statistic is reported, if relevant changes are detected then we refer these as *major changes*.

### 6. Related works

The problem of identification and notification of changes in semi-structured data [13,24,25] has been deeply investigated and many techniques and system aiming to provide these kind of services are available. Some techniques approaches the change detection problem by identifying the sequence of changes that transform a document in its modified version, these techniques are computationally expensive therefore there is no known application of this techniques to web change detection. An efficient system implemented to verify changes in documents is the HTMLDiff [8] system. In this system comparisons can be made locally or over http or ftp protocols, it displays the changes found in a single screen shot, it can analyze folders and produce HTML reports for both single-file and folder analysis. This system is particularly suited for

<table>
<thead>
<tr>
<th>URL</th>
<th>Portion Monitored</th>
<th>Time (seconds)</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://eBay.co.UK">http://eBay.co.UK</a></td>
<td>DVD</td>
<td>3</td>
<td>98</td>
</tr>
<tr>
<td><a href="http://eBay.co.UK">http://eBay.co.UK</a></td>
<td>LAPTOPS</td>
<td>2</td>
<td>97</td>
</tr>
<tr>
<td><a href="http://eBay.co.UK">http://eBay.co.UK</a></td>
<td>GSM MOBILE</td>
<td>5</td>
<td>94</td>
</tr>
<tr>
<td><a href="http://eBay.co.UK">http://eBay.co.UK</a></td>
<td>AERONAUTICS</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td><a href="http://eBay.co.UK">http://eBay.co.UK</a></td>
<td>COMMEMORATIVE</td>
<td>1</td>
<td>97</td>
</tr>
<tr>
<td><a href="http://eBay.co.UK">http://eBay.co.UK</a></td>
<td>SPORT</td>
<td>3</td>
<td>99</td>
</tr>
<tr>
<td><a href="http://eBay.co.UK">http://eBay.co.UK</a></td>
<td>FILM</td>
<td>4</td>
<td>99</td>
</tr>
<tr>
<td><a href="http://eBay.co.UK">http://eBay.co.UK</a></td>
<td>ROCK HARD</td>
<td>2</td>
<td>95</td>
</tr>
<tr>
<td><a href="http://eBay.co.UK">http://eBay.co.UK</a></td>
<td>CDs</td>
<td>2</td>
<td>99</td>
</tr>
<tr>
<td><a href="http://eBay.co.UK">http://eBay.co.UK</a></td>
<td>SCIENCE &amp; NATURE</td>
<td>1</td>
<td>92</td>
</tr>
</tbody>
</table>

Fig. 10. Time and accuracy of the CMW system.
Microsoft Word documents. Available systems for web change detection belongs to one of the following categories:

- systems focused on URL changes;
- systems that provide change detection facilities on specific application domains;
- systems for change detection over wrapped Web pages.

The system *NetMind* [14] surely belongs to the first category: it provides a text-based and keyword-based change detection and notification services. An improvement provided by the same vendor is the Mind-it notification service that allows users to monitor a specific URL and to customize the monitoring task by signing the portion of the document that are not relevant to the user and then should not be monitored. Another system for URLs monitoring is *SQURL* by Bluesquirrel (Search and Query URL’s) that is the basis for the new product *Webwhacker* [23] that allows user to verify changes made to the structure of the web site being monitored. *WWWfetch* is a system that allows users to express various boolean condition on the URLs of the pages being monitored.

![Fig. 11. Results for page monitoring.](image-url)
A very useful tool that belong to the second category is the SIFT service by Stanford that monitor the Web to notify user when changes are made to the news previously chosen. Another system belonging to this category is the Etrade alert facility by Amazon, it is a new book notification service.

The main systems that work on wrapped Web pages is the C3 project at Stanford [3] and OpenCQ [10,11]. The WebCQ system [12] integrates most of this features allowing users to specify sentinel over the document to monitor specific condition, uses a proxy service and grouping techniques for trigger processing.

The CMW system behaves differently from all the previously mentioned systems. CMW triggers provide the user an high degree of flexibility, thus it is usable on a wide range of application domains. Furthermore triggers can be defined using a simple visual interface and the user is never required to deal with the details of the underling HTML structure. Finally it works very well even if the page size is great.

7. Conclusion

The need for systems that provide Web change detection facilities has become essential due to the fast rate of change of the information on the WWW. In this paper we have proposed a new technique which allows the efficient detection of Web document differences, in a quantitative way.

Our technique, rather than being based on computing an edit script that produces the updated version of the whole document, focuses on the detection of changes in a specified portion of the document. Using this technique has been possible to define a language that permits to express complex queries on web documents’ changes.

The CMW system we have developed is a personal web update monitoring service. It is composed of a query editor, that permits to specify web update queries in a fully visual and interactive way, and a query engine, that manages web query execution. Future work will be devoted to the design and implementation of a Web based prototype of the system.

References

Sergio Flesca received his Ph.D. in computer science engineering at the University of Calabria, Italy. Currently he is an Assistant Professor at the Engineering Faculty at the University of Calabria. He was a visiting researcher at the Computer Science Department of the Vienna University of Technology. His research interests include deductive and active databases, semi-structured data, XML query languages and HTML/XML wrapper languages. He is also actively involved in the Italian Lixto Research Lab.

Elio Masciari received his Ph.D. in computer science engineering at the University of Calabria, Italy. Currently he is a Researcher at the ICAR institute of the Italian National Research Council. His research activity is mainly focused on techniques for the analysis and mining of data, text and web mining, XML query languages and XML structural similarity. He is also involved in several research project in the data mining field.