Detecting Semantically Correct Changes to Relevant Unordered Hidden Web Data

Vladimir Kovalev Sourav S. Bhowmick

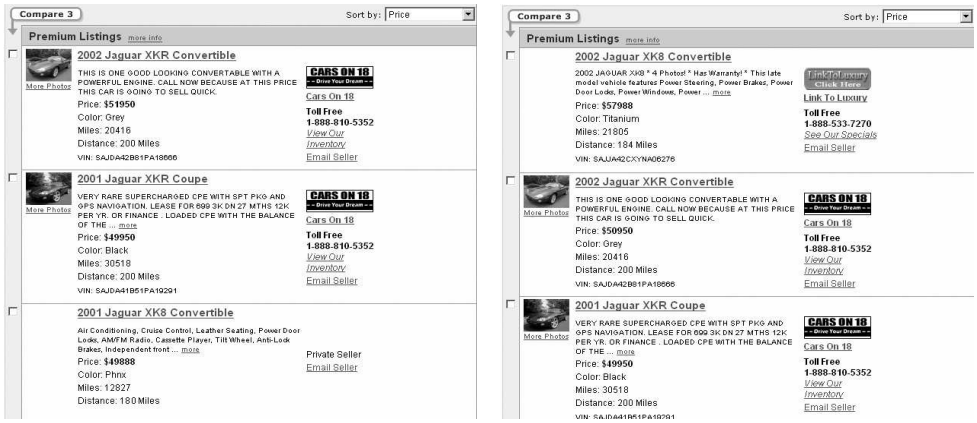
School of Computer Engineering
Nanyang Technological University, Singapore
assourav@ntu.edu.sg

Abstract. Current proposals for XML change detection use structural constraints to detect the changes and they ignore semantic constraints. Consequently, they may produce *semantically incorrect* changes. In this paper, we argue that the semantics of data is important for change detection. We present a semantic-conscious change detection technique for the hidden web data. In our approach we transform the unordered hidden web query results to XML format and then detect the changes between two versions of XML representation of the hidden web data by extending X-Diff, a published unordered XML change detection algorithm. By taking advantage of the semantics, we experimentally demonstrate that our change detection approach runs up to 7 times faster than X-Diff on real life hidden web data and always detect changes that are semantically more correct than those detected by existing proposals.

1 Introduction

Data in the hidden web can change any time and in any way. These changes are reflected on the results of the queries posed on the hidden web. Hence the problem of detecting and representing changes to hidden web data in the context of such query results is an important problem. Let us illustrate with an example.

Consider the `AutoTrader.com` which is one of the largest car Web sites with over 1.5 million used vehicles listed for sale by private owners, dealers, and manufacturers. The search page is available at `http://www.autotrader.com/findacar/index.jtmpl?ac_afflt=none`. Figures 1(a) and 1(b) represent snapshots of results returned for searching for *Jaguar* cars on 2nd and 5th July, 2003 respectively. The results are presented as a list of cars. Each result contains car details such as `Model`, `Year`, `Price`, `Color`, `Seller`, `Vehicle Identification Number (VIN)`, etc. Given the query results and our understanding of its semantics, we may wish to state the following constraints. First, the `VIN` in the results uniquely identifies a particular car entity. Note that VIN may not be present for some results. However, if it exists then it will not be removed from the subsequent versions of query results involving the particular car. Second, the `year` of manufacturing and the `model` of each car do not get modified in different versions. That is, a "mercedes" cannot be updated to "jaguar" or if the manufacturing year of a car is "2001" then it cannot be modified to "2002" or any other year in the subsequent versions. Similarly, the `seller` attribute of a car does not get modified in different versions for the same car entity. Furthermore, as underlying information related to the *Jaguar* cars has changed during this time period, the



(a) 2 July, 2003. (b) 5 July, 2003.

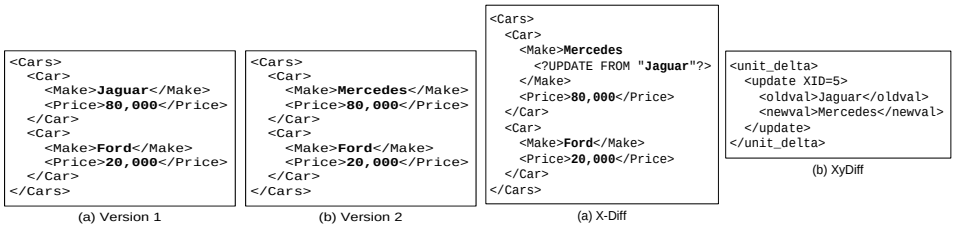
Fig. 1. The results of searching for *Jaguar* cars.

query results in Figure 1(b) contain the relevant changes that have occurred between 2nd July, 2003 and 5th July, 2003. Particularly, the first car in Figure 1(b) has been inserted, the last car in Figure 1(a) has been deleted, and the price of the first car in the older version has been updated from "$51950" to "$50950" during this period.

In this paper, we consider the problem of detecting the above types of changes automatically taking structural as well as a broad class of semantic constraints into account. Note that our goal is to detect and represent the changes that are relevant to a user's query, not any arbitrary change to the hidden web data. Also, we assume that the hidden web query results are unordered.

In our approach, we do not directly compare the two versions of hidden web query results (Figures 1(a) and 1(b). This is because hidden web data is HTML-formatted and every hidden web site generates it in its own fashion. Thus it becomes extremely difficult and cumbersome to develop a generalized technique that can be used for change detection to the hidden web data. Consequently, it is important to develop a technique for transforming the hidden web data to a more structured format so that we can develop such generalized technique for the hidden web data. Hence, the hidden web query results are transformed into XML format before they are used for change detection.

Since there are several recent efforts in the research community to develop change detection algorithms for XML documents [1, 5], an obvious issue is the justification for designing a separate algorithm for detecting changes to the XML representation of hidden web data. We argue that although such algorithms will clearly detect syntactically correct changes, they fail to detect *semantically correct* changes. For example, these algorithms may detect that the `model` of a car element has been updated from "mercedes" to "jaguar". However, this is semantically incorrect! We elaborate on this further in Section 2.



(a) Sample documents. (b) Edit scripts.

Fig. 2. Change detection.

We have developed a semantic-conscious change detection algorithm called HW-Diff to address the above issue. As we assume the query results to be unordered, we have extended *X-Diff* [5], a published unordered XML change detection algorithm to implement HW-Diff using Java. *To the best of our knowledge, this is the first approach that address an important limitation of state-of-the-art XML change detection algorithms by making them semantic-conscious.* Our experimental results on four real data sets show that the HW-Diff detects more semantically correct changes compared to the X-Diff algorithm. Furthermore, it runs up to 7 times faster than X-Diff due to the exploitation of semantic constraints.

2 Related Work

XyDiff [1] is designed for detecting changes in ordered XML documents. X-Diff [5] is designed for computing the deltas for two unordered XML documents. However, these algorithms in many cases produce *semantically incorrect* changes. Let us illustrate this limitation with a simple example. Figure 2(a) shows the old and new versions of an XML document. The results returned by X-Diff and XyDiff are depicted in Figures 2(b). Observe that both the algorithms detect that the `Make` of the first car is updated from *Jaguar* to *Mercedes*. However, in reality the car whose `Make` is *Jaguar* definitely cannot be updated to *Mercedes*. Hence, the results generated by X-Diff and XyDiff are semantically incorrect. The correct types of changes that should be detected here are *deletion* of the first `car` element in Figure 2(a)(i) and *insertion* of a new car (the first `car` element in Figure 2(a)(ii)). HW-Diff address this limitation by extending an existing algorithm (X-Diff) with semantic constraints. By incorporating such semantic knowledge in a general-purpose change detection algorithm, we can dramatically decrease the number of matching possibilities between the two versions of the document and increase the performance of the algorithm. Hence, HW-Diff shows better response time compared to X-Diff.

3 Semantic Constraints in Hidden Web Query Results

In this section, we present the semantic constraints in hidden web query results that are captured in the XML representation of the query results using HW-Stalker [2, 3].

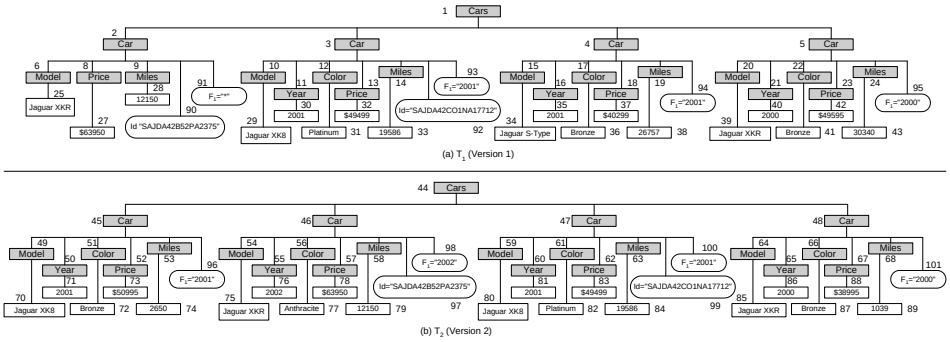


Fig. 3. Partial XML trees of results in Figures 1.

Identifier: Some elements in a set of query results can serve as a unique identifier for the particular result, distinguishing them from other results. For example, the `VIN` uniquely characterizes every `Car` in the query results from a car database. These elements are called *identifiers*. In this work we assume that the *identifier*, being assigned to a particular query result, does not change for this result through different versions of the query results. However, it is possible for the *identifier* to be missing in a result. Also, if an *identifier* is specified (not specified) for a result in the initial version of the query results or when the result appeared for the first time, then it will remain specified (not specified) throughout all versions, until the result is deleted. This reflects the case for most web sites we have studied. HW-Stalker allows specifying only one *identifier* for each result. As each result is transformed into a subtree in the XML representation of the hidden web query results, the *identifier* of a particular node in the subtree is modeled as an XML attribute with name `Id` and the *identifier* information as value. We now illustrate with an example the usefulness of the *identifiers* in change detection.

Reconsider the query results in `autotrader.com`. Figure 3 shows partial XML trees of the results in Figure 1. The `Car` nodes in T_1 and T_2 have child attributes with name `Id` and value equal to the `VIN`. Intuitively, if we wish to detect the changes between the two versions of the query results, then we can match the `Car` nodes between two subtrees by comparing the `Id` values. For instance, the node 2 in T_1 matches the node 46 in T_2 and the node 3 in T_1 matches the node 47 in T_2 (same `VIN` values). However, the nodes 2 and 3 do not match the node 45 as it does not have any *identifier* attribute.

Facilitator: One or more elements in the result of the hidden web query result set can serve as non-unique characteristics for distinguishing the results from one another. This is particularly important when the results do not have any *identifier* attribute. Two results that have the same characteristics (same attribute/value pair) can be matched with each other. While results that have different characteristics can not be matched with each other. Examples of types of such characteristics are: the `Year` or `Model` of a `Car` node in the query results from car trading site. These non-unique elements are called *facilitators*. Note that these elements may not identify a result (entity) uniquely. But they may

provide enough information to identify results that do not refer to the same entities.

We allow specifying any number of *facilitators* on a node. The *facilitators* are denoted by node attributes with names F_1, F_2, ... , F_n for all n *facilitator* attributes specified for a particular node. If a node does not have a *facilitator* attribute (the subelement may be missing) then the *facilitator* value is set to "`*`". Note that the *facilitator* attribute for a node can appear in any version of the query results, but once it appears we assume that it will not disappear in the future versions. As we never know which *facilitator* may appear for a node in the future, a node with missing *facilitator* attribute should be matched with nodes having *facilitators*. The reader may refer to [3] for guidelines the user may follow to choose *facilitators*. In [4], an algorithm is discussed for automatic discovery of facilitators from hidden web data.

Reconsider the Figure 1. We can find several candidates for *facilitators*, i.e., `Color`, `Year`, or `Model`. However, based on the semantic constraints introduced in Section 1, it is reasonable to use the `Year` or `Model` as the *facilitator*. Figure 3 shows the *facilitators* for various nodes. There is one *facilitator* specified: the `Year` as an attribute with name F_1 for every `Car` node. Note that if a `Car` node does not have an subelement `Year` then F_1 is set to "`*`". Now let us match the node 45 in T_2 with all the nodes in T_1. Observe that node 45 does not have a `VIN`. Therefore, it cannot match with nodes 2 and 3. Hence we do not need to compare node 45 with these nodes. Using F_1 we also observe that the node 45 cannot match with node 5 as the facilitators do not match. We can see that the node 45 only matches node 4 in T_1 as it does no have any `VIN` and its F_1="2001". However, this is not sufficient information to confirm whether these two nodes represent the same car entity. But if we use both the `Year` and `Model` as facilitators then we can answer this question by comparing the `Model` of node 45 with that of node 4. As the `Model` of nodes 4 and 45 are not identical, we can say that these nodes do not represent the same car entity. Thus, we can state that the node 45 is inserted in T_2 as none of the car entities in T_1 matches the car entity described by node 45.

4 Change Detection

4.1 HW-Signature

In order to detect the changes between two trees, we first need to find a matching of corresponding nodes in the two trees. Obviously, matching every node in the first tree to every node in the second tree is an inefficient solution. In X-Diff, this problem is addressed by using the concept of *node signature* [5]. The node signature compares not only the node *type* (text, element, and attribute nodes) and the node *name* (e.g., `car` and `price` nodes) of two nodes, but also their ancestors to determine the nodes that are to be matched. It is obtained by concatenating the names of all its ancestor with its own name and type. However, this notion of signature ignores semantic constraints embedded in the data. Hence, we extend the definition of signature in X-Diff by incorporating the semantic constraints associated with the hidden web query results. We call this

extended notion of signature as *HW-signature*. Hereafter, in this paper signature refers to the X-Diff version of node signature and HW-signature refers to the signature used in HW-DIFF.

The HW-signature imposes semantic constraints by adding the notion of *identifiers* and *facilitators* in the definition of signature. It is used later in our change detection algorithm HW-DIFF to facilitate the process of matching nodes between the trees representing different versions of the hidden web query results. It not only reduces the number of nodes that are to be matched to one another, but also facilitates more semantically accurate matching compared to the signature.

We first introduce the notion of *HW-node signature* which is one of the basic component of HW-signature. The *HW-node signature* of a node is concatenation of the name of the node and its *identifiers* and *facilitators* (if any).

Let us define some terms for our exposition. Given a node x in the DOM tree T, let $Type(x)$ and $Name(x)$ be the node type of x and the node name of x (including the attributes) respectively. Also, if node x contains *identifiers* and *facilitators* then let $IdValue(x)$ and $FValue(F_i, x)$ be the values of the *identifier* and the *facilitator* attribute F_i of x respectively. Then,

Definition 1. [**HW-node Signature**] *Suppose x is an element node and $\mathcal{F}$ be a set of facilitator attributes on x. Then the* ***HW-node signature*** *is defined as follows: $HW-node(x) = Name(x)[Id(x)][Fac(x)]$ where (1) $[Id(x)] = /Id/IdValue(x)$ if identifier attribute is defined for node x, otherwise $[Id(x)]=\varnothing$; (2) If $\mathcal{F} \neq \varnothing$ then $[Fac(x)] = [F_1(x)][F_2(x)]\ldots[F_n(x)]$ where $[F_m(x)] = /F_m/FValue(F_m, x)$ $\forall$ $1 \leq m \leq n$ and $F_m \in \mathcal{F}$. Otherwise, $[Fac(x)] = \varnothing$.* ∎

For example, consider the two trees in Figure 3. Based on the above definition, $HW-node(3)$=`Car/Id/`AJDA42C01NA17712/F_1/2001 and $HW-node(52)$=`Price`.

Definition 2. [**HW-Signature**] *Let x be an element node. Then* ***HW-Signature****(x)=/HW-node(x_1) /HW-node(x_2)/.../HW-node(x_n)/ HW-node(x)/Type(x), where x_1 is the root of T, (x_1,x_2,...,x_n,x) is the path from x_1 to x. If x is a text node, then* ***HW-Signature(x)****=/HW-node(x_1)/HW-node(x_2) /.../HW-node(x_n)/Type(x).* ∎

Definition 3. [**HW-node Equality**] *Let x and y be the element nodes in T_1 and T_2. Let I_x and I_y be the identifiers of x and y. Let $\mathcal{F}_x$ and $\mathcal{F}_y$ be sets of facilitators of x and y respectively and $|\mathcal{F}_x| = |\mathcal{F}_y|$. Let $P = \{(F_{x1}, F_{y1}), (F_{x2}, F_{y2}) \ldots (F_{xk}, F_{yk})\}$ be the set of pairs of facilitators such that $F_{xi} \in \mathcal{F}_x$, $F_{yi} \in \mathcal{F}_y$, $FValue(F_{xi}, x) \neq$ "$*$", and $FValue(F_{yi}, x) \neq$ "$*$" $\forall$ $1 \leq i \leq k$. Then* ***HW-Node****(x)=****HW-Node****(y) iff (1) signature(x)=signature(y); (2) $IdValue(x) = IdValue(y)$; and (3) If $|P| \neq \varnothing$ then $FValue(F_{xi}, x) = FValue(F_{yi}, y)$, $\forall$ $1 \leq i \leq |P|$.* ∎

In Figure 3, *HW-node(3)=HW-node(47)*, but *HW-node(2)≠HW-node(47)* as they have `Id` attributes with different values. Note that for all nodes that have a facilitator F where $FValue(F, x)$ = "$*$" the last condition in the above definition is ignored. That is, only first two conditions are sufficient for HW-node equality.

Definition 4. [**HW-signature Equality**] *Let x and y be element nodes in trees T_1 and T_2 respectively. Let x_1 be the root of T_1 and (x_1,x_2,...,x_n,x) is the path from x_1 to x. Let y_1 be the root of T_2 and (y_1,y_2,...,y_n,y) is the path from y_1 to y. Then,* ***HW-signature(x)=HW-signature(y)*** *iff Type(x)=Type(y) and HW-node(x_i)=HW-node(y_i) $\forall\ 1 \leq i \leq n$.* ∎

Observe that the above definition of HW-signature equality combines the criteria of matching nodes by their signatures and matching nodes based on the semantic constraints in the hidden web data. If two nodes have identical HW-signatures then they have identical signatures. However, the inverse is not always true.

4.2 Minimum Cost Matching

In this section we introduce the notion of matching. Formally,

Definition 5. [**Matching**] *A set of node pairs (x, y), M, is called a* ***matching*** *from tree T_1 to tree T_2 iff (1) $\forall$ (x,y) $\in M$, $x \in T_1$, $y \in T_2$, HW-signature(x)=HW-signature(y); (2) $\forall$ (x_1,y_1) $\in M$, (x_2,y_2) $\in M$, x_1=x_2 iff y_1=y_2 (one-to-one); (3) Given (x,y) $\in M$, suppose x' is the parent of x, y' is the parent of y, then (x',y') $\in M$ (preserving ancestor relationships).* ∎

Observe that the notion of matching is defined in the same way as in [5]. The key difference is in Criteria (1): we use the HW-signature as basic criterion for matching nodes instead of signature as used in X-Diff. Criteria (2) and (3) in this definition prevent children being matched if their ancestors are not matched. These criterion reflect the integrity of XML segments.

Based on a matching M from T_1 and T_2, we can generate an *edit script*. It can be shown using the same method as in [5] that there is a *minimum-cost matching* that corresponds to a *minimum-cost edit script*. Note that we use the same notion of minimum cost edit script as defined in [5] except for that fact that instead of using the signature, we use the notion of HW-signature. Observe that every matching in our approach is also a matching in X-Diff. However, the inverse is not always true. Thus, the X-Diff approach always has equal or more number of matchings compared to our approach. As a particular matching defines a particular edit script, the X-Diff approach chooses the minimum-cost edit script from more number of scripts compared to our approach.

4.3 Algorithm HW-Diff

The algorithm HW-Diff takes as input the old and new versions of the XML representation of the hidden web query results D_1 and D_2. It returns as output the edit script E for transforming D_1 to D_2. As the algorithm is an extension of the X-Diff algorithm, it can be best described by the following three phases. Note that we do not present the pseudocode here as it is similar to X-Diff except that we use the HW-signature instead of the signature to match nodes.

Phase 1: Parsing and Hashing Phase: In this step, the algorithm parses the input XML documents to trees, and assigns the HW-signatures to each node and computes the XHash values [5] for all the nodes in both trees. After this step,

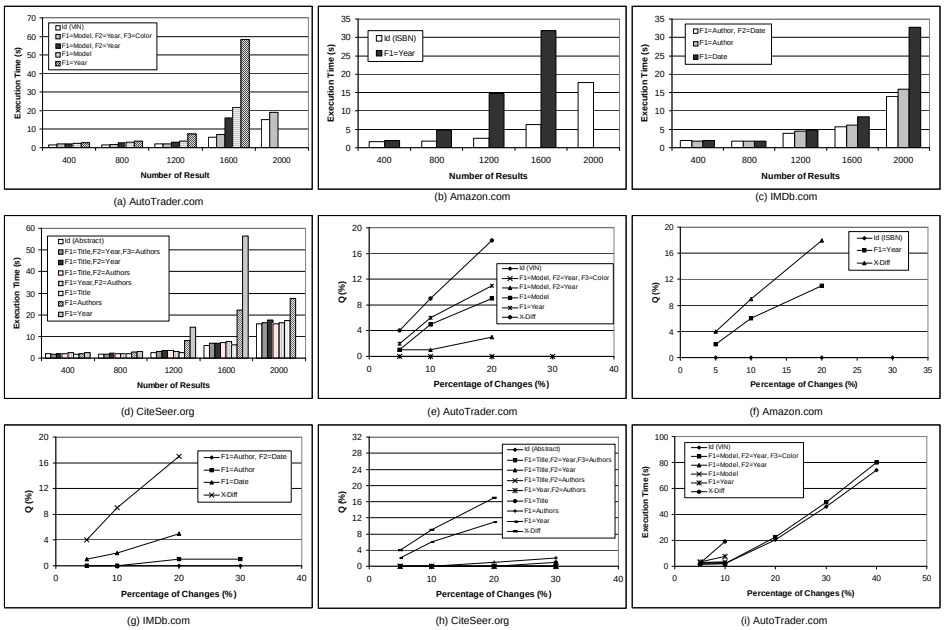


Fig. 4. Performance study.

the algorithm checks if the two trees are equivalent by comparing the XHash values of the roots.

Phase 2: Matching Phase: In this step, the algorithm generates the *minimum-cost matching* between two trees by computing the editing distances between the nodes with equal HW-signatures on each level of the trees, going from the leaf nodes to the root nodes. The steps for computing the *minimum-cost matching* in our approach are the same as the steps of computing it for the X-Diff algorithm except that we only compute distances between the nodes with equal HW-signatures instead of equal signatures. As the number of nodes with equal HW-signatures is always less or equal to the number of nodes with equal signatures, the number of distances between the nodes to be calculated in the X-Diff approach is equal or more than the number of distances to be calculated using the HW-Diff approach. These two numbers are identical only when there are no *facilitators* or *identifiers* specified in the query results.

Phase 3: Edit Script Generation Phase: In this phase, we generate a minimum-cost edit script for changes to the hidden web data based on the minimum cost matching found in the matching phase. This step is similar to X-Diff.

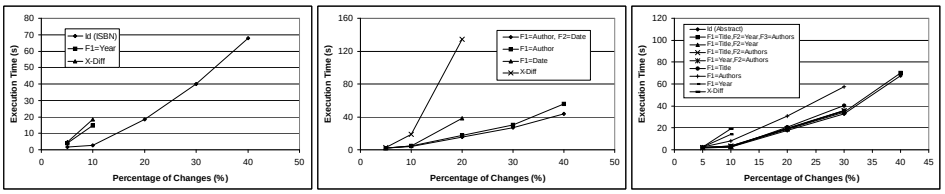
5 Performance Evaluation

We have implemented HW-Diff using Java. All the experiments have been performed on a Pentium 4 CPU 2.4 GHz with 512 MB of RAM. We used Microsoft Windows 2000 Professional as operating system. We use the data from the following four hidden web sites for our experiments: `AutoTrader.com`, `Amazon.com`,

`IMDb.com`, and `CiteSeer.org`. We generated a data set for the experiments based on the results of a set of queries. We created a data set containing files with 400, 800, 1200, 1600, and 2000 results for each site. We monitored these sites for a period of 6 months and archived the query results for our change detection process. We also implemented a program that generates semantically meaningful changed versions of these files with 5%, 10%, 20%, 30%, and 40% of changes of all three types of changes (insert, update, and delete) equally distributed. Note that in this experimental set up and data set, the Java implementation of X-Diff (downloaded from www.cs.wisc.edu/∼yuanwang/xdiff.html) cannot detect the changes when the percentage of change is more than 20% due to lack of memory (`java.lang.outofmemory.error`). For some data set it cannot detect the changes when more than 10% data has changed. Similar situation arises for HW-DIFF if there are too many equal values for the facilitators.

Execution Time vs Semantic Attributes: Our first experiment is to evaluate the affect of the selection of attributes in the query results as *identifiers* and *facilitators* on the change detection time. We applied HW-DIFF to the data with different combinations of attributes selected as the *facilitators* and the *identifier*. We used the data set described above with 10% changes. Figures 4(a) to 4(d) show the results for different hidden web sites. HW-DIFF demonstrates best performance when we use the *identifier* attribute. If no *identifier* attribute can be modeled for a particular result set then several *facilitator* attributes help us to achieve similar performance. Observe that for small sets of query results, the execution time is almost the same and is actually defined by the parsing time. However, as the result size increases, sometimes a single *facilitator* attribute results in significant increase in the change detection time compared to other attributes due to the frequent appearance of results having identical values for this attribute in the sample data. For example, Figure 4(a) shows that if we only use the `Year` of the `Car` as *facilitator* attribute then the change detection time increases significantly, but not so much when we select the `Model` as *facilitator* attribute. This is because more number of cars in the two versions of the query results have the same manufacturing `Year` compared to the number of cars having the same `Model`.

Semantic Incorrectness of Changes: This experiment evaluates the *semantic incorrectness* of the changes detected by HW-DIFF compared to X-Diff. Let N be the total number of results in the change detection delta file D. Let M of these results contain semantically incorrect changes. Then, the *semantic incorrectness* of changes (denoted as Q) is defined as $Q=\frac{M}{N} \times 100\%$. As in the previous experiment, we applied HW-DIFF to the data with different combinations of attributes selected as the *facilitators* and the *identifier*. The data for X-Diff is same but without any *facilitators* or *identifier*. For each web site, we used the data set with 800 results and 5%, 10%, 20%, 30%, and 40% changes. Figures 4(e) to 4(h) show the results for the different hidden web sites. As in the previous experiment, HW-DIFF demonstrates best performance when the *identifier* attribute is used. In this case the semantic incorrectness is reduced to 0% for all cases. If no *identifier* attribute can be modeled for a particular result



(a) Amazon.com (b) IMDb.com (c) CiteSeer.org

Fig. 5. Performance study (Contd.).

set then several *facilitator* attributes help us to achieve similar performance. For example, for `AutoTrader.com` the combination of `Model`, `Year`, and `Color` helps to achieve similar performance. Observe that for all cases HW-Diff outperforms X-Diff as far as detection of the number of semantically correct changes. In fact, the number of incorrect changes detected by X-Diff approximately equals the number of changes in the document for all the cases. Also observe that even though single *facilitator* attribute leads to increase in detection of semantically incorrect changes, it is still less than X-Diff for the given sites.

Execution Time vs Percentage of Changes: Our last experiment measures the execution time on different percentage of changes. For each web site, we used the data set with 1200 results and 5%, 10%, 20%, 30%, and 40% changes. Figures 4(i) and 5 show the results for different hidden web sites. The affects of the *identifiers* and *facilitators* are the same as the previous two experiments. Observe that for all the sites HW-Diff outperforms X-Diff (up to 7 times) especially when the percentage of changes is more than 10%.

6 Conclusions

In this paper, we presented a technique to detect semantically correct changes to the hidden web query results. Our work is motivated by the problem that existing change detection algorithms do not exploit the semantic constraints of the data. In our approach, the unordered hidden web query results are transformed to XML format using the HW-Stalker algorithm [2] and then detect the changes between the two versions of XML representation of the hidden web query results. We propose an algorithm HW-Diff that extends X-Diff, a published change detection algorithm for unordered XML, by incorporating the semantic constraints associated with the data. HW-Diff detects more semantically correct changes compared to X-Diff and runs up to 7 times faster than X-Diff for the given data set.

References

1. G. Cobena, S. Abiteboul, A. Marian. Detecting Changes in XML Documents. In *ICDE* , San Jose, 2002.
2. V. Kovalev, S. S. Bhowmick, S. Madria. HW-Stalker: A Machine Learning-based Approach to Transform Hidden Web Data to XML. In *DEXA*, Zaragoza, Spain, 2004.
3. V. Kovalev, S. S. Bhowmick, S. Madria. HW-Stalker: A Machine Learning-based System for Transforming QURE-Pagelets to XML. To appear in *Data and Knowledge Engineering Journal (DKE)* , Elsevier Science, 2005/2006.
4. V. Kovalev, S. S. Bhowmick. Mining Facilitators and Identifiers from Hidden Web Query Results. *Technical Report* , CAIS-06-2005, 2005.
5. Y. Wang, D. DeWitt, J-Y Cai. X-Diff: A Fast Change Detection Algortihm for XML Documents. *ICDE* , India, 2003.