

Revisiting Web Data Extraction Using In-Browser Structural Analysis and Visual Cues in Modern Web Designs

Alfonso Murolo^(✉) and Moira C. Norrie

Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland
{alfonso.murolo,norrie}@inf.ethz.ch

Abstract. Recent trends in website design have an impact on methods used for web data extraction. Many existing methods rely on structural analysis of web pages and, with the introduction of CSS, table-based layouts are no longer used, while responsive design means that layout and presentation are dependent on browsing context which also makes the use of visual clues more complex. We present DeepDesign, a system that semi-automatically extracts data records from web pages based on a combination of structural and visual features. It runs in a general-purpose browser, taking advantage of direct access to the complete CSS3 spectrum and the capability to trigger and execute JavaScript in the page. The user sees record matching in real-time and dynamically adapts the process if required. We present the details of the matching algorithms and provide an evaluation of them based on the top ten Alexa websites.

Keywords: Data extraction · Wrapper induction · Browser

1 Introduction

As the web rapidly evolved into a vast data source covering any and every topic, it was natural that researchers strove to develop efficient ways of programmatically extracting data from web pages. Thus, the research area of web data extraction emerged and many different techniques have been proposed over the last two decades. As more and more web pages started to be generated dynamically based on queries to underlying content management systems and application databases, the term *deep web* was introduced to refer to the huge amounts of dynamic data contained in web pages that could not be indexed by traditional search engines. Web data extraction involved generating wrapper code that could generate queries and extract that data from the resulting web pages.

Instead of hard-coding wrappers, *wrapper induction* techniques were developed to enable wrappers to be generated based on patterns detected within dynamically generated web pages. These wrappers typically enabled data to be gathered from sets of web pages and converted into a tabular structure which could then be handled as relational data. Wrapper induction systems can be divided into two general categories: those which attempt to extract entities at

the page level, and those which extract records from lists of entities contained within a page using similarity measures [1].

While most previous work has focused on the use of structural cues for detecting patterns within and across sets of web pages, some techniques have combined structural cues with visual ones such as the coordinates of bounding boxes, the width and height of elements, and the use of different font faces and sizes [2–4]. However, with the introduction and evolution of the CSS standards in recent years, the information that can be gleaned from CSS stylesheets in conjunction with HTML markup has increased enormously. At the same time, CSS has enabled designers to move away from table-based layouts and many of the structural cues used in previous web data extraction techniques have been eradicated from HTML due to the separation of concerns supported by the new HTML and CSS standards. Further, the widespread adoption of responsive design techniques to enable websites to adapt to different viewing contexts means that previous techniques that made use of visual cues such as the position or size of elements to locate data or labels may no longer function in all contexts since these are no longer fixed but rather controlled by CSS media queries. Last but not least, the extensive use of JavaScript nowadays also needs to be taken into account as it may be responsible for generating or updating data, structure and presentation and can therefore have a major impact on data extraction.

Our goal was to develop a tool that would enable users to extract data as they browse. One possible use of such a tool would be for users to collect and add data to their personal libraries while browsing as proposed in Sift [5], but without requiring the presence of semantic markup. It has also been shown that such a tool can support web development by enabling database code plus data to be generated based on data extracted from digital mockups of a website [6].

We therefore decided to revisit web data extraction techniques with a view to developing methods that could not only cope with modern standards and practices, but also exploit them in wrapper induction. The main contributions of this paper are the following:

- We propose a hybrid approach that performs extraction at the record level using a combination of structural, content and visual features based on current web technologies and design practices. The approach requires users to label parts of an example record but this was kept to a minimum.
- The approach can be implemented as a web application that runs in a general-purpose browser and takes advantage of direct access to the page, the DOM and the CSS3 rules applied at runtime as well as the capability to trigger and execute JavaScript in a page. We present a specific tool, called DeepDesign, which was implemented as a browser extension for Chrome which allows users to see the matching process in real-time, adjusting the input if necessary until they are satisfied with the results. The extraction process is executed in each page directly, rather than delegating it to an external system, and therefore is completely realized with web technologies.

- An evaluation of our approach based on the top ten websites according to the Alexa¹ ranking and a comparative evaluation against previous work.

The methods presented are based on earlier work where we showed how custom post types for WordPress² could be generated based on sample content used in digital mockups [6]. As well as adapting the approach to allow data to be extracted from any website, we have improved the algorithms by taking more visual cues into account so that data record detection is more reliable.

We first provide a review of related work on web data extraction in Sect. 2 before going on to present an overview of our approach in Sect. 3. Details of our matching algorithm and its reliability improvements are presented in Sect. 4 and we discuss the implementation of the DeepDesign tool in Sect. 5. We then present an evaluation of our algorithm in Sect. 6. Concluding remarks and an outline of future work are given in Sect. 7.

2 Related Work

There has been a lot of research on the problem of extracting data records and performing *wrapper induction* from web pages in the past two decades. A wrapper embeds some form of extraction rules to match content based on a pattern detected in a web page. The detection of the patterns may involve different features: most use structural cues [1], while others use visual indicators, content-related measurements or some combination of these. We will outline the various approaches used, focussing on those closest to our own work.

The survey by Chang et al. [7] classifies methods using many different dimensions including the degree of automation and the difficulty of the task addressed. Here, we choose to classify techniques based on two dimensions: the level at which data extraction is performed and the integration with the browser. The level of data extraction can be either at the page level or the record level. Techniques that work at the page level typically take multiple pages containing data records of the same type as input and infer patterns for extracting data records from individual pages. Those that work at the record level typically take one or more pages containing multiple occurrences of data records of the same type and try to detect repeating patterns in individual pages that can be used to extract records from that page.

Among the most important systems that work at the page level is *NoDoSE* [8], which can generate wrappers and extract content to be stored in a database in a supervised way. The tool needs user-provided oracles to decompose entire web pages. Then, according to an inferred grammar, NoDoSE computes the field delimiters and extracts the attributes of each data record. This system is demanding for users as they have to provide various inputs to guide the creation of the extraction rules. NoDoSE was implemented as a separate system and is not integrated in any web-based system.

¹ <http://www.alexa.com>.

² <https://wordpress.org/>.

RoadRunner [9] also works at the page level, but data extraction is instead formulated as a decoding process. It requires a set of similar pages as input and assumes that the dynamic content will be different across pages. All dynamic parts are taken into account in guiding the generation of a wrapper. RoadRunner is divided into two modules. The *classifier* module clusters similar pages together and generates a wrapper based on their similarities and differences. The *labeller* module then tries to discover attribute names for data records within similar pages. While this is an interesting approach, it requires a set of similar pages rather than a single page as input. For the labeller to work, it requires the coordinates of the bounding boxes of the text nodes in the pages. It is a Java-based system and, although it tries to rebuild and re-render pages to locate the bounding boxes of candidate labels, it mainly performs a structural analysis.

An example of a record-level system is *IEPAD* [10] which uses so-called PAT trees to discover repeating patterns in pages. It implements a semi-automated approach and was later extended by *DeLa* [11] to remove the need for user interaction. Computation starts by detecting a *data-rich section* of the web page and then patterns in the DOM tree to generate regular expressions for future data extraction. As an extension of IEPAD, DeLa also works at the record level, aiming to detect multiple records from search page results. Again, the DeLa approach differs from our own in its requirement to have multiple input pages in order to detect data-rich sections. Moreover, it does not use any visual features, focusing only on content and structural analysis. DEPTA [12] also works at the record level to detect repeating patterns, but parses only the HTML tag strings verifying that the repeated record's tags have the same parent in the HTML tag tree. Using an algorithm called MDR-2, DEPTA processes the pages in three steps: first, it builds a DOM tree of the page, using the visual features of bounding boxes to compare those of child and parent elements. Second, a tree is built based on which bounding boxes contain those of other elements. Finally, data records are identified by analysing structural cues and the containment relationships of elements represented in the tree. DEPTA was integrated with the MSHTML API, which provides a rendering of each HTML element and builds the DOM tree for the system with the results in a spreadsheet.

A different approach that processes a document with similarly structured data without supervision is the one by Lu et al. [13]. This system performs alignment, clustering and annotation of sets of data records that appear as the results of queries to websites performed through forms. The alignment step at the start of their process identifies the data records. Then their fields are clustered in groups with different semantic meanings, for example names and titles. The second step attempts to automatically annotate the fields with labels based on heuristics. For example, if the data is in a table, the column headings are used as labels. While their method also takes into account some features considered in our system such as the contiguity and presentation style of data units, it only looks at six visual properties of text elements. The system accesses the rendered properties using a ViNTs [14] component as a browser renderer and extracting two critical cues to their system: the first is a tree structure that maps elements

to the original page DOM tree and the second is the coordinates of the bounding boxes for the elements along with the other visual properties.

ViWER [15] is another work that uses visual as well as structural features, also focussing on the sizes of the bounding boxes of elements, in conjunction with a tree matching algorithm, to detect data records. In contrast to these works which use visual cues alongside structural analysis, *ViDE* [16] mainly relies on visual features. *ViDE* first builds a so-called visual block tree using the coordinates of the bounding boxes of the data regions and of the contained elements. Second, each region in the tree is analysed based on a group of features such as content-related features, positioning, layout and a limited number of appearance features such as font faces and sizes. By doing block grouping and clustering, it then constructs the records and their fields. *ViDE* retrieves the rendered information through a programming interface that allows their system to be interfaced with an external browser renderer, in their case Internet Explorer. Unlike *ViDE*'s approach which moves radically away from structural cues, we aim to exploit both structural and visual cues, but to make more extensive use of visual cues and content-based features than in the combined methods proposed previously. Our approach is to first group elements by structural similarity and only then use visual cues to reconstruct records in the propagation of the labels.

Another important characteristic of methods for web data extraction is the role that the user plays in terms of interacting with the system and guiding the extraction process. *DEByE* [17] specifies an approach to user interaction which we share: a user provides example data records and the system then tries to find similar data records. In the case of *DEByE*, wrappers are generated from examples and these are then used to extract data from other web pages. Similar to our approach, *DEByE* works at the record level. However, our approach differs in a number of ways. First, we have tried to minimise the input required by the system by requiring users to mark up only parts of at most 2 examples. Second, our tool is in-browser while *DEByE* runs as a separate system. Third, and most important, our record extraction process works in the opposite way, first detecting the records and then propagating the fields.

OLERA [18] also operates based on user-specified examples. It runs as an independent system with an embedded browser. *OLERA* learns extraction rules at the record level from the example by first finding the information block of interest by interaction with the user, and then looking for similar blocks using approximate matching techniques and creating extraction patterns through string alignment techniques. The data is shown to the user through spreadsheets in the *OLERA* interface, and they can drill-down or roll-up through the data, allowing them to further manipulate the results. A third stage allows attribute names for the extracted data to be specified in the spreadsheet.

Previous work such as *Sift* [5] has also targeted the use case of extracting records from a web page while browsing. However, the *Sift* approach relies on semantic markup specified by Schema.org³ or microformats to aid the extraction process.

³ <http://www.schema.org>.

Finally, Thresher [19] is a previous project that also integrated a web induction system in a browser. Specifically, it was integrated in the Haystack information client, which acts as a browser for the semantic web. The system uses structural cues that take advantage of a user-assisted labelling procedure to perform the semantic web annotations. The user achieves two different goals at the same time: the first being the generation of the requested wrapper and the second the semantic annotation of content not previously annotated, thereby promoting the spread of semantic web technologies. Our approach differs by not only taking DOM-based features into account, but also content-related measurements and especially visual-based features, introducing the complete CSS3 spectrum to extract the data records.

3 DeepDesign

DeepDesign has been implemented as a Chrome extension and can therefore be loaded and executed in any page that the browser opens. Users can quickly access the tool while browsing whenever they find data of interest that they want to extract as structured data records. To do so, they simply click on the DeepDesign button which triggers the opening of a control panel for our tool as shown on the right of Fig. 1.



Fig. 1. A page from DBLP with the control panel of the matcher.

From the control panel, the user can start and fine-tune the detection process. The first button allows users to start the labelling process of a single example that will guide the matching process. Users annotate examples by clicking on any element that they consider to be part of the data record and giving it a label. We have tried to minimise the amount of annotations that users have to perform by letting them annotate only one element of a repeating field. Further, they do not need to specify whether a field may be repeating or optional. For example,

in the publications list from DBLP shown in Fig. 1, a user would be required to annotate just one author, one title and the pages in the proceedings. The system will propagate the annotations to similar fields in each record automatically, for example all authors of a publication.

Once an example has been annotated, the user can start the extraction process. Data records are matched based on a similarity metric that takes both structural and visual cues into account. Moreover, the annotations get propagated to all records and all similar fields that have been matched, within and across different records. The system dynamically highlights records in the page as the matching process proceeds so users can see results produced in real-time. Elements labelled by the system are underlined in green and the user can check these by hovering over the element to display the propagated label. This enables the user to quickly evaluate the quality of the matching process. If the results show the matching process has had limited success, they can stop the process and either adjust threshold parameters or annotate a second example before re-starting the extraction process. The three thresholds used in our algorithms will be explained in Sect. 4 where we describe the matching process in detail. For users who do not have a good understanding of the algorithm and the effect of these thresholds, they can easily experiment with different values on the sliders based on some general help guidelines.

In Sect. 4, we also explain in detail why a single example may not be sufficient to produce good results in some cases and how specifying a second example can significantly improve the quality of the matches.

Users can also control the use of CSS caching techniques that we introduced to boost the speed of the matching algorithm. When CSS caching is active, a DOM element will be decorated with an object representing its entire processed CSS style when the system first wants to find the style applied to that element rather than going through the browser and rebuilding it on each access. While this significantly improves the speed of the matching process, it can reduce the reliability of results in pages where JavaScript or CSS animations alter the visual properties. There is therefore a performance trade-off between speed and reliability and users have the option to disable caching before re-running the extraction process if the results generated do not meet expectations.

Once the extraction process is complete, the user can select what to do with the extracted data records. DeepDesign provides a storage area in the browser where records can be saved. The basic options are to store the records in the extracted format or send these to a web application using a message-based system that allows Chrome extensions to send content to any web page that is open. The web application might process the data directly or map it to another format before storing it. In this way, users can open another web application and send the matched records to it by reading them from the storage area in which they were saved. Additionally, the control panel allows the JSON representation of the matched records to be obtained which can then be read from external applications. In the case of the first scenario presented in the introduction, the external application could store it in the user's personal library of data gathered from

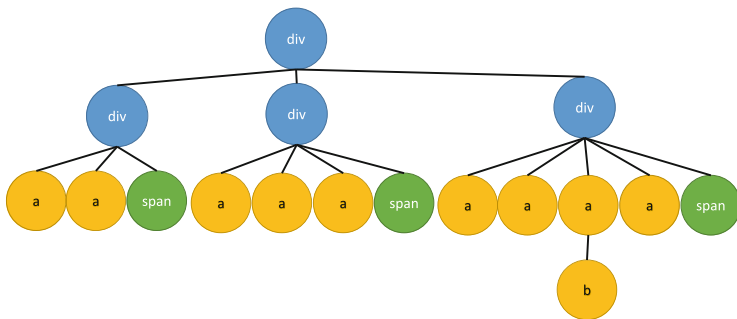


Fig. 2. A tree representing a list of some example publications.

websites. For the second scenario where data is extracted from a digital mockup as part of the process of developing a website, an external application could, for example, generate custom post types for the WordPress platform based on the schema of the extracted data as described in [6]. In the latter case, populating instances of those custom post types with the actual data used in the mockup and extracted by DeepDesign would be optional. The control panel interface to DeepDesign was implemented using HTML, CSS and JavaScript since Chrome allows extensions to have web-based interfaces that can communicate with the tool being executed in the page through message-based communication.

4 Matching Algorithm

The matching algorithm uses structural as well as visual cues. The input elements are the annotated fields of an example data record as discussed in the previous section and the DOM of the web page as well as the set of tuning parameters which we explain in detail later in this section. To explain how the algorithm works, we will use the example in Fig. 2 which represents a possible DOM for a list of publications similar to that shown in Fig. 1. The anchor elements, **a**, represent authors and link to their individual web pages. The **span** elements include the titles of the publications. We will assume that the only annotations provided initially by the user are the label “*author*” for the first anchor to the left, and the label “*title*” for the first **span** element.

Our algorithm works in four steps. We first aim to detect the boundaries of the annotated record, then look for subtrees with a similar structure in the page. Once these have been found, we propagate the labels from the example record to each of the matched records and then propagate them locally within the records. This means that the author and title labels of our example would be propagated to an author and title of all subtrees before detecting the co-authors within each record in a second step and propagating the author label to them. We will now discuss the details for each of these steps.

1. Record Boundaries Detection. The algorithm starts by detecting the boundaries of the annotated example record using a lowest common ancestor

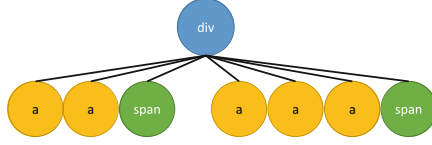


Fig. 3. A corresponding example tree with no common lowest common ancestor for the two records in the list.

(LCA) algorithm, which the annotated elements in the DOM as input. An LCA is the nearest node in the hierarchy that is an ancestor for all of the annotated elements. In Fig. 2, this corresponds to the first of the three `div` elements which is on the left branch and wraps the first publication.

However, this cannot always be guaranteed. There are cases where the records are siblings of the same parent, such as the example presented in Fig. 3. Here, the two publications are not wrapped by an LCA. First of all, we need to be able to detect such a case and, second, we have to be able to identify the record boundaries once such a case has been detected. This is an example where it is necessary for the user to provide two examples in order to detect that such a situation occurs and be able to demarcate records. The LCA of both records will be the same node, namely the one enclosing the entire list of records, which we will call $L2$. In Fig. 3, $L2$ would be the `div` element enclosing the two publications. Once we have detected such a situation, we can trigger our fault correction. Since there should be exactly one LCA per record, we need to locate the first and last nodes in the children of $L2$, which act as delimiters for the first data record. To do so, we traverse the elements of the first annotated example and their parents up to $L2$ and assign them a specific marker. Then the children of $L2$ are re-examined knowing what the start and end of a record look like from the perspective of its children. This is necessary because it is possible that the annotated elements are not direct children of $L2$. In the example of Fig. 3, it would be clear based on two examples that the first `a` element is the start of the first example annotated by the user and the first `span` element is the end of that record since they are direct children.

Making the assumption that all records start and end in the same way, we look for similar nodes among the children of $L2$ so we can find the boundaries of all records. The similarity is assessed according to the distance function $dist(x, y)$ which is a weighted sum of various distances, the style distance ratio Δ_s , the tree edit distance, the ratio between the length of the longest common subsequence in the text of nodes and their maximum length, and finally the Levenshtein distance [20] between the XPaths of the two elements. The style distance ratio is defined as:

$$\Delta_s = \frac{\Delta_{css}}{\max(\Delta_{css} + S, 1)}$$

where Δ_{css} is the number of CSS rules which differ between the two elements being considered, and S is the number of CSS rules which are similar. The style distance ratio Δ_s is also used in the later step of local label propagation. To

achieve the maximum reliability of this measurement, the CSS rules for the two elements being compared are retrieved from the computed style, which returns all the CSS properties, including the non-standard ones computed by the browser for each DOM element. The tree-edit distance instead is calculated through an approximation algorithm called pq-gram distance [21].

Once we have found candidates close enough to be matched with the start and end of the example record, respectively, we process these candidates to mark them as boundaries of records, enclosing them with a wrapper node which provides an LCA in cases where none is present.

2. Finding Similar Records. After the boundaries of the example record have been found, we have a DOM subtree similar to all other data records of the same type present in the page. To allow for variability across data records, we rely on the approximated tree edit distance provided by the pq-gram distance. To consider records as candidates, their distance has to be smaller than a threshold called the *tolerance factor*. This is the first of the thresholds that can be specified in the control panel of DeepDesign. At this stage, the middle and the right branch of Fig. 2 would be detected as possible candidates for record matching.

Up to this point, there is no information about the internal fields of the newly detected data records. For this reason, we now need to propagate user annotations automatically to the matched records.

3. Cross-Record Propagation. In general, this step aims at locating the elements that correspond to those annotated by the user in each of the matched data records. Once these have been located, the corresponding annotations can be migrated from the original example record to all of the matched records.

We introduce an intermediate step of candidate detection for each label, and a candidate selection step among those previously found. Candidate detection and selection make extensive use of visual cues, content-based measurements and structural analysis to ensure that only the most suitable nodes are considered for cross-record propagation. In our previous work [6], we addressed this step using an XPath-based approach where, for each annotation, we inferred the XPath of an annotated element relative to a data record, and accessed the corresponding element in each candidate data record. While this tree-to-tree mapping of XPaths is a fallback technique when not enough labels could be propagated with the candidate detection, this implementation assumes a regularity of record structures that is not always present in practice. For example, in the case of publications, fields can be optional and even variable in structure. We therefore decided on an alternative approach that could cope with record-related peculiarities that impact on the DOM structure and/or order of elements and could lead to faulty cross-record propagation.

We introduced a candidate detection step that locates which elements of the subtree are the best matches for an annotated element. This step starts by looking for all the elements in the local subtree, filtering out the non visible ones. For each label, the algorithm computes a list of candidates with the closest distance from the originally annotated element according to the previously presented distance function $dist(x, y)$. For text nodes, since visual cues can be

fewer, also the text and style similarities of the surrounding elements is considered.

Since multiple elements can have the same distance from the originally labelled one, the candidate detection step returns, for each label annotation, at most three sets of elements in the current subtree with the lowest distances to the labelled element in the example record. Thus, we obtain sets of candidate elements for each label, together with the corresponding distance from the original annotated element, that are good candidates for the cross-record propagation of that label. However, various issues can still affect these candidate elements. It is possible that some of these are ancestors or parents of other elements in the list. Even worse, it could be possible that some elements appear in more than one set, being a candidate for different labels, possibly with different distances. This is likely to happen when there are elements which should be assigned different labels, but have a very small difference in visual styling.

To tackle these two issues, we first perform a cleaning step to remove ancestors from the list of candidates. We decorate each possible parent of all candidate elements with a specific marker and, after all elements of the list have been processed, remove marked elements from the list.

After the cleaning step, we select a single element from each of the candidate lists to be associated with the corresponding label. Note that, since an element can appear in more than one candidate list, it is actually possible that all the candidate lists contain the same elements, which means that we may not be able to propagate some of the labels if there are more labels than elements.

Our goal is to minimise the sum of the distances of the elements selected from the sets, while maximising the number of sets from which a candidate is picked. We call this problem *LABEL-ASSIGN*, and have found it easily convertible to the *minimum weighted maximum independent set* problem. We will provide more information on the rationale and conversion in Sect. 5. Based on previous research [22], we have implemented a greedy algorithm that returns an approximated solution.

The algorithm starts by building a list E of objects for all the nodes from the candidate lists associated with their label and distance. It first sorts them in ascending order according to the following function, as proposed by Sakai et al. [22]: $f(e) = W(e)/[O(e) + 1]$ where $W(e)$ is the distance of the element e and $O(e)$ is its number of occurrences in the candidate lists S_i .

It then builds a map of solutions that associates a label to an element from the candidates, iterating the following: The first element in the list E is selected as part of the solution, with its label as key and the node as a value, and is removed from E . At this stage, all the elements that belonged to the same set, as well as those from other sets that are competing for the same label, are removed from E together with all the other occurrences of the selected element. The current element is also removed from all the other candidate lists in which it appears. Now the element list E is again sorted and the loop repeated until there are no more candidates.

At the end of this stage, we have a map that associates an annotation label to one DOM element of the current record's subtree. We also know which

distances each element in the map had. We discard the record if the average distance between all the selected elements and the corresponding ones in the original example is above a threshold, or if we have matched a number of labels which is smaller than the 66 % of the total amount of labels in the original example. However, the user can modify these values via the control panel. The optionality parameter which has the default setting of 66 % allows for records with optional fields. If records have optional fields, it could cause the algorithm to mistakenly discard fully matched records because these records lack optional attributes that the user annotated in the example. In general, the optionality parameter allows the system to be tuned to deal with the variability that may be present in records due to optional fields as well as controlling how strict the system should be about the possibility of false positives since it forces the algorithm to only accept records which have a higher number of assigned labels.

4. Local Label Propagation. The next step is to replicate the labels to all of the elements within a matched data record with a similar meaning, for example all authors of a publication. Consider our example where, so far, only one of the anchor tags in both the second and third branch of the tree in Fig. 2 would have been selected as an author. We want to propagate this label to the other elements that also represent authors within the respective subtrees. To do so, we take all the visible descendants d of the record node with the same tag name (for example `div`) and all of the node's visible children that do not contain any d . Using agglomerative hierarchical clustering, we group these descendants of the record node according to a *distance function*. As a stopping criterion for the hierarchical clustering, we check the distance between the clusters against a threshold called the *grouping factor*. This is the second threshold bar that can be configured by the user in the control panel. The clustering distance function takes into consideration various factors, which can be both visual and structural cues. The structural cues currently considered are: the tag equality, the *style distance ratio* mentioned in the previous stage, and two discontinuity scores, namely the structural tag discontinuity and the field discontinuity score. The former increases as the elements are further away (separated by elements of different tags), while the latter increases as the elements between those being compared received a different label from the previous step. While the tag equality is trivial to understand and the style distance ratio has already been presented, the two discontinuity scores accommodate arbitrary scenarios in subtrees, and therefore deserve some more explanation. All subtree elements are projected into a linear vector in the order in which they appear in the document, starting from the root of the record. In this linear vector, all elements will be compared in their contiguity for each individual contiguity function. Sequences of different types of elements between two compared elements over this line will increase their structural tag discontinuity, while sequences of elements with different *assigned* labels will increase the field discontinuity. The reason why we need the distinction of assigned labels is that many nodes in each record subtree may not be selected for the cross-record propagation step, and therefore their label may be undefined. These nodes do not contribute to discontinuity in the context of record fields.

Once the two discontinuities have been summed up, we divide them by the maximum depth of the record subtree to mitigate the effect of the projection on a linear vector. As a result, the elements which are considered to be similar in structural and visual terms, or which happen to be presented in a contiguous fashion, will be put into the same cluster. Majority voting is used to decide which label should be applied to each cluster and it is then applied to all elements of the cluster.

5 Implementation Challenges

Implementing the DeepDesign tool using only web technologies introduced a number of challenges arising from both the technical limitations of the technologies and the complexity of the problems to be solved in the different steps of the algorithm. For example, browsers still do not offer a way to retrieve which rules from the stylesheets apply to an element. Although there are libraries to do so, they require expensive processing and cannot be used for complex computations.

Also, improving the runtime of the algorithm through parallelisation is very hard since the JavaScript Web Workers do not allow parallel access to the DOM, which is required in our case at every step.

Another challenge is the cross-record propagation step described in Sect. 4, where our tool needs to solve the *LABEL-ASSIGN* problem. We used an algorithm for *minimum-weighted maximum independent set* (*mWMIS*). This means having to convert instances of *LABEL-ASSIGN* into instances of *mWMIS* by building a graph G in such a way that, for every element in every candidate set, we create a node in the graph. If an element appears in more than one candidate set, we still add a different node to the graph. Every node corresponding to an element in a candidate set is connected to every other node from the same candidate set, meaning that they are in a clique. The clique also includes all elements from other candidate sets with the same label. Moreover, if an element appears in more than one candidate set, all the other nodes corresponding to that element need to be connected. Finally, the weights to the elements are the distances with which their set has been labelled.

Now let S be a solution for the *LABEL-ASSIGN* problem where its elements maximise the number of candidate sets covered and are those of minimum weight. Given how the graph has been constructed and the constraint that an element can only be picked once from any set and any label, S will be an independent set, and the nodes in S are those that make it of minimum weight. Conversely, let S now be a maximum independent set of minimum weight. Then we can pick the corresponding elements in the candidate sets, having guaranteed that at most one element per set is selected, and the element is not selected in more than one set. Since S had minimum weight, and weights are the same as in the instance of *LABEL-ASSIGN*, we will also have the minimum-weighted solution in this case. Unfortunately, for the *minimum weighted maximum independent set*, we cannot have an approximated algorithm within a ratio independently from the weights, unless $P = NP$ [23]. Therefore we have chosen a greedy algorithm for deriving

the *minimum-weighted maximum independent set* proposed in previous research that tries to approximate within a ratio that is dependent on the weights [22]. Finally, the cross-propagation algorithm was designed to find the solution with the smallest distance from the annotated elements, while maximising the number of labels that can be assigned in order to minimise the chance of having false negatives. This can however increase the occurrence of errors in the case where there is a high chance that some of the fields will be missing in the records to be detected. If the visual or structural differences between these fields according to $\text{dist}(x, y)$ is very low, it could happen that the LABEL-ASSIGN prefers to assign a label to the wrong element rather than keeping it unassigned. This can especially occur with pure text nodes which, having less styling information than usual elements, can more easily be marked as a candidate for the wrong label.

6 Evaluation

We have performed a preliminary evaluation of DeepDesign. To make sure that there was no bias in the websites and content used in the evaluation, we decided to base it on the top ten websites in the Alexa ranking⁴. Note that since Google occupied both positions 1 and 10 (with its Indian version), we selected the eleventh in the ranking as a replacement for the tenth. For each selected website, we identified the most important content to retrieve independent of whether it was statically or dynamically loaded. The pages used in this dataset contained a total of 212 records. We performed multiple labelling operations on different pages of each website, trying to capture different data elements that the website could offer in different HTML structures. For the sake of space we provide all the details on the labelling operations and the dataset in an archive.⁵

We selected four evaluation metrics: the *time* elapsed to perform the automatic matching expressed in seconds, the number of misplaced labels (labelling errors), the *precision* and *recall* over the dataset, and the *precision* and *recall* where partial matches are counted as positives. By partial matches, we refer to data records which were extracted by DeepDesign, but with some repeating fields not completely propagated. The tests were run on an i7-4770 with 16GB of RAM using Chrome 45. The use of the matching tool involves two main factors: the tolerance factor and the grouping factor. The former acts as a threshold for the pq-gram approximation algorithm, while the latter acts as threshold for the hierarchical clustering, considering two clusters ‘far enough’ when their distance is above the grouping factor. This means that the higher these two thresholds are, the more expensive the computation will be. For example, setting a tolerance factor of 1.0 implies considering any element in the DOM tree as a possible candidate record since 1.0 is the highest possible distance provided by the implementation of the pq-gram approximation⁶. Thus, the only way these can be discarded is if the cross-record propagation fails to propagate

⁴ <http://www.alexa.com/topsites> - 15.10.2015.

⁵ <http://dev.globis.ethz.ch/deepdesign/DDpreliminaryeval.zip>.

⁶ <https://github.com/hoonto/jqgram> - 15.10.2015.

Website	Pos.	T (sec)	TF	GF	TR	MR	PMR	LE	FP	FN	TM	Prec.	Prec.(PM)	Recall	Recall (P.M.)	TMR	DOM
Google	1	0.46	0.8	0.5	9	9	0	0	0	0	9	1	1	1	1	9	946
Facebook	2	4.2	0.95	0.5	8	5	0	4	0	3	5	1	1	0.625	0.625	5	3839
Youtube#1	3	1.6	0.75	0.5	15	15	0	0	0	0	15	1	1	1	1	15	2349
Youtube#2	3	6.4	0.8	0.1	27	27	0	0	51	0	78	0.34615	0.346154	1	1	27	5278
Baidu	4	1.6	0.9	0.65	9	7	2	1	0	10	10	0.7	0.9	0.778	1	9	747
Yahoo	5	1	0.65	0.5	20	19	1	1	0	0	20	0.95	1	0.95	1	20	1790
Amazon	6	5.7	1	0.2	16	7	9	8	0	0	16	0.4375	1	0.438	1	16	4076
Wikipedia	7	0.1	1	0.5	19	19	0	0	0	0	19	1	1	1	1	19	970
qq	8	1.3	0.95	2.5	22	17	2	0	1	19	0.89474	1	1	0.85	0.95	19	2435
Twitter	9	3.52	0.9	0.15	19	19	0	0	0	0	19	1	1	1	1	19	4201
Taobao	11	4.7	0.55	0.5	48	48	0	0	0	0	48	1	1	1	1	48	3062

Pos. - Position in the alexa ranking PMR - Partially Matched Records Prec. - Precision TM - Total Matches
T - Time elapsed (sec) P.M. - including Partial Matches FP - False Positives DOM - Size of DOM
TF - Tolerance factor TMR - Total Matched Records FN - False Negatives
GF - Grouping factor
TR - Total Records
MR - Matched Records
LE - Labeling errors

Dataset 1 (Alexa)		Dataset 2	
Precision	0.85	DeepDesign	RoadRunner
Precision PM	0.93	Precision	0.92
Recall	0.88	Recall	0.98
Recall PM	0.96		0.6

Fig. 4. Results of the evaluations. At the top we see detailed results from the websites from Alexa (Dataset 1). In the bottom-right corner, aggregated results are shown from both datasets.

labels to their descendants. Setting the grouping factor value too high instead forces the algorithm to go through more clustering procedures, implying more calls to the clustering distance function. Moreover, the DOM structure of each site will have an impact on the overall processing time. In Fig. 4, we report the size of the DOM of each web page used in our tests.

The evaluation for each site involved the following steps:

1. Identification of the data records in the page.
2. Labelling the fields of an example record.
3. Activation of data record matching.
4. Check against the number of examples: If every data record that we expected to match was not successfully matched, we would raise the tolerance or the grouping factors to repeat the matching.

We show the data collected during the evaluation in Fig. 4. For YouTube, we showcase two different relevant results, which involved different matching processes. The two sessions on YouTube were performed to extract video details from the subscription feed (#1) and from the “Advertised videos” (#2).

The median time for execution was 1.6 s. For some pages, the processing took considerably longer: YouTube #2 (6.4 s), Facebook (4.2 s), Taobao (4.7 s) and Amazon (5.7 s). However, it is important to note that the count of the elements in the DOM and the tolerance factor for these matchings were rather high. In the case of Facebook, this can probably be related to the high variability between different posts which can include link sharings, videos or status updates.

As true positives for our tool, we only considered those matches whose label assignment was handled correctly. If any element that should have received a label was assigned a different one or none, we considered them as partially

matched. On the other hand, any element that should not have been assigned a label but was assigned one is considered a labelling error if the rest of the record was correctly matched. The average precision without taking into account any partial matches was 0.85 (std. dev. 0.23), while it increases to 0.93 (std. dev. 0.18) if we consider partial matches. This is a good result, with one outlier in YouTube #2, where the tool matched 51 data records which were considered false positives, even though these were not visible on the screen. However, since these were indeed matched by the tool, even if they were not visible, they have to be taken into account. The average recall without taking partial matches into account is 0.88 (std. dev. 0.18), while the average with partial matches is slightly higher at 0.96 (std. dev. 0.11). The worst results were seen on YouTube #2 (51 false positives out of 27 total records to match). In all of the experiments, there were a total of 15 labelling errors, which include labels placed on elements that should not have been considered, giving 0.07 labelling errors per correctly matched data record. The fact that the tolerance factor needed to be high in the majority of the captures ($TF > 0.75$) might mean that the current implementation would benefit from an improvement in the subtree matching phase, with an algorithm that could potentially better discriminate between different subtrees. However, due to the constraints in the label propagation step explained in Sect. 4, the overall algorithm is still robust and offers good precision most of the time, especially when considering partial matches. Partial matches occur in four cases: Yahoo, Baidu, Amazon, and qq.com. Having partial matches can mean that the cross-record propagation step has failed to identify the best candidate element to migrate its label, or that the clustering phase has failed to group all of the similar fields together, for example all the authors within a publication.

We ran a second evaluation to compare DeepDesign against a tool from previous work. Although quite an old system, we chose RoadRunner [9] in our comparative evaluation since, to the best of our knowledge, it is the only tool from previous research that is still available for download. For the comparison, we decided to collect a set of pages that mixed different design practices so there was not an unfair bias towards DeepDesign which was designed to cater for modern designs. For this reason, we collected a set of 10 pages from 10 different websites which cover quite different design structures. Some of them have a table-based layout and some involve structures where the records do not have their own LCA. This dataset included a total of 223 records. It has to be noted that RoadRunner works at the page level so, in order to be able to feed the 10 pages to RoadRunner, we had to manually split each page to make them contain exactly one record per page. Another task that was handled manually (and therefore can be prone to error) was the configuration of RoadRunner, which involved an XML tool that allows many parameters to be specified to fine tune the extraction process. RoadRunner does not perform a record detection step since the records are already split across pages. We therefore compare only the quality of the alignment process, which in our tool is handled by the cross-record propagation and local-label propagation steps. Therefore, the granularity of this second evaluation is not at the record level, but at the field level. After

manually establishing the ground truth, we recorded the number of correctly separated fields by the two tools, together with the false positives and false negatives. As shown in Fig. 4, RoadRunner achieved 67 % precision and 60 % recall, with DeepDesign achieving better performance with 92 % precision and 98 % recall. However, RoadRunner was much faster, always completing the extraction process in less than a second, whereas DeepDesign has a median time of 6.3 s and a very high average running time of 14.2 s due to three pages of the dataset where the runtime reached approximately 14, 17 and 23 s, respectively. In two of these pages, records did not have an LCA and so required the additional processing for this case described in Sect. 4. Note that the times reported only include the execution of the algorithm and not the time for labelling. In the last case, each record had a unique LCA, but our hypothesis is that, since the records had very long text descriptions, the analysis of these descriptions for the cross-propagation step required a much longer time. It is important to note that while these are both systems for data extraction, the tools were designed to be used in different ways and hence the difference in runtimes was expected. RoadRunner is a system that runs as a batch algorithm with a list of input files and it visualises the results only at the end. DeepDesign extracts data while browsing, incrementally showing the matched results while the process is running by using the rendering engine of the browser to dynamically update the page.

7 Conclusion

We have presented DeepDesign, a tool for extracting data from web pages while browsing. Users guide the extraction process by performing a minimal labelling of fields within an example record and can fine tune the process as it runs based on an incremental, real-time visualisation of results. The evaluations performed are very encouraging and we plan to integrate it in a system that supports the development of websites based on digital mockups in order to carry out studies with developers and end-users. We have also started working on support for the detection of complex schema structures that involve relationships or aggregations over the data. Additionally, we are considering ad-hoc formulations of the tree-edit distance problem that can allow for more variability in the subtrees of data records. Finally, we plan to improve the support for HTML text nodes.

References

1. Zheng, S., Song, R., Wen, J., Giles, C.L.: Efficient record-level wrapper induction. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management. ACM (2009)
2. Liu, W., Meng, X., Meng, W.: Vision-based web data records extraction. In: Proceedings 9th International Workshop on the Web and Databases (2006)
3. Manabe, T., Tajima, K.: Extracting logical hierarchical structure of HTML documents based on headings. *Proc. VLDB Endowment* **8**(12), 1606–1617 (2015)
4. Pembe, F., Canan, F., Güngör, T.: A tree learning approach to web document sectional hierarchy extraction. In: Proceedings of the 2nd International Conference on Agents and Artificial Intelligence (2010)

5. Geel, M., Church, T., Norrie, M.C.: Sift: an end-user tool for gathering web content on the go. In: Proceedings of the 2012 ACM Symposium on Document Engineering, pp. 181–190. ACM (2012)
6. Murolo, A., Norrie, M.C.: Deriving custom post types from digital mockups. In: Cimiano, P., Frasincar, F., Houben, G.-J., Schwabe, D. (eds.) ICWE 2015. LNCS, vol. 9114, pp. 71–80. Springer, Heidelberg (2015)
7. Chang, C., Kayed, M., Girgis, M.R., Shaalan, K.F.: A survey of web information extraction systems. *IEEE Trans. Knowl. Data Eng.* **18**(10), 1411–1428 (2006)
8. Adelberg, B.: NoDoSE a tool for semi-automatically extracting structured and semistructured data from text documents. In: Proceedings of the 9th ACM SIGMOD International Conference on Management of Data (SIGMOD). ACM (1998)
9. Crescenzi, V., Mecca, G., Merialdo, P.: Roadrunner: towards automatic data extraction from large web sites. In: Proceedings of the 27th International Conference on Very Large Data Bases (VLDB). Morgan Kaufmann (2001)
10. Chang, C., Lui, S.: IEPAD: information extraction based on pattern discovery. In: Proceedings of the 10th International Conference on World Wide Web (WWW). ACM (2001)
11. Wang, J., Lochovsky, F.H.: Data extraction and label assignment for web databases. In: Proceedings of the 12th International Conference on World Wide Web (WWW). ACM (2003)
12. Zhai, Y., Liu, B.: Structured data extraction from the web based on partial tree alignment. *IEEE Trans. Knowl. Data Eng.* **18**(12), 1614–1628 (2006)
13. Lu, Y., He, H., Zhao, H., Meng, W., Yu, C.: Annotating search results from web databases. *IEEE Trans. Knowl. Data Eng.* **25**(3), 514–527 (2013)
14. Zhao, H., Meng, W., Wu, Z., Raghavan, V., Yu, C.: Fully automatic wrapper generation for search engines. In: Proceedings of the 14th International Conference on World Wide Web. ACM (2005)
15. Hong, J.L., Siew, E., Egerton, S.: ViWER-Data extraction for search engine results pages using visual cue and dom tree. In: Proceedings of the 1st International Conference on Information Retrieval & Knowledge Management (CAMP). IEEE (2010)
16. Liu, W., Meng, X., Meng, W.: Vide: a vision-based approach for deep web data extraction. *IEEE Trans. Knowl. Data Eng.* **22**(3), 447–460 (2010)
17. Laender, A.H., Ribeiro-Neto, B., da Silva, A.S.: DEByE - data extraction by example. *Data Knowl. Eng.* **40**(2), 121–154 (2002)
18. Chang, C., Kuo, S.: OLERA: semisupervised web-data extraction with visual support. *IEEE Intell. Syst.* **19**(6), 56–64 (2004)
19. Hogue, A., Karger, D.: Thresher: automating the unwrapping of semantic content from the world wide web. In: Proceedings of the 14th International Conference on World Wide Web. ACM (2005)
20. Levenshtein, V.: Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.* **10**, 707–710 (1966)
21. Augsten, N., Böhlen, M., Gamper, J.: Approximate matching of hierarchical data using Pq-Grams. In: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), VLDB Endowment (2005)
22. Sakai, S., Togasaki, M., Yamazaki, K.: A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Appl. Math.* **126**(2), 313–322 (2003)
23. Demange, M.: A note on the approximation of a minimum-weight maximal independent set. *Comput. Optim. Appl.* **14**(1), 157–169 (1999)