BER$_y$L: A System for Web Block Classification

Andrey Kravchenko(✉)

Department of Computer Science, University of Oxford, Oxford, UK
andrey.kravchenko@cs.ox.ac.uk

Abstract. Web blocks such as navigation menus, advertisements, headers, and footers are key components of Web pages that define not only the appearance, but also the way humans interact with different parts of the page. For machines, however, classifying and interacting with these blocks is a surprisingly hard task. Yet, Web block classification has varied applications in the fields of wrapper induction, assistance to visually impaired people, Web adaptation, Web page topic clustering, and Web search. Our system for Web block classification, BER$_y$L, performs automated classification of Web blocks through a combination of machine learning and declarative, model-driven feature extraction based on Datalog rules. BER$_y$L uses refined feature sets for the classification of individual blocks to achieve accurate classification for all the block types we have observed so far. The high accuracy is achieved through these carefully selected features, some even tuned to the specific block type. At the same time, BER$_y$L avoids a high cost of feature engineering through a model-driven rather than programmatic approach to extracting features. Not only does this reduce the time for feature engineering, the model-driven, declarative approach also allows for semi-automatic optimisation of the feature extraction system. We perform evaluation to validate these claims on a selected range of Web blocks.

1 Introduction

When a human looks at a Web page, he or she sees a meaningful and well-structured document. However, such interpretation is not accessible for the computer as it only "sees" the technical layers of the Web page [19] represented merely by the source code (e.g. HTML, CSS, and JavaScript files) and the rendered models (e.g. the DOM tree, CSSOM with computed attributes, and executed JavaScript). Whilst it is probably infeasible for a machine to replicate the human's perception and derive the associated mental model, it would be highly useful for it to understand the logical structure and functional role of various elements of the Web page for a wide range of different applications through

This work was supported by the ESPRC programme grant EP/M025268/1 "VADA: Value Added Data Systems – Principles and Architecture".

M. L. Gavrilova and C. J. K. Tan (Eds.): Trans. on Comput. Sci. XXXIII,
LNCS 10990, pp. 61–78, 2018.
https://doi.org/10.1007/978-3-662-58039-4_4

the analysis of the layout, as well as visual and textual features. Web search is an especially important potential application, since semantic understanding of a Web page allows the restriction of link analysis to clusters of semantically coherent blocks. Hence, we aim to build a system which provides a structural and semantic understanding of Web pages.

This paper is concerned with the task of Web block classification. Informally speaking, a Web block is a logically consistent segment of a Web page layout, an area which can be identified as being visually separated from other parts of the Web page. A Web block carries a certain semantic meaning, such as title, main content, advertisement, login area, footer, and so on. It is through the semantic meaning of individual Web blocks that a human understands the overall meaning of a Web page. There are many blocks with a common semantic meaning (i.e. a layer of Web specific objects [19]) among different websites and domains (e.g. headers, navigation menus, logos, pagination elements, and maps) that share common Web patterns. Even within one block type, individual blocks can vary significantly in both their structural and visual representations. For example, consider the diversity of navigation menus illustrated in Fig. 1. This diversity of blocks makes the task of their accurate and fast detection challenging from a research and implementation perspective. In general, the difficulty of the block classification problem lies not only in the complexity of individual classifiers, but also in the complexity of the entire system which needs to balance the individual accuracies of its constituent classifiers and its overall performance.

There are several important applications of Web block classification including automatic and semi-automatic wrapper induction [2,11,28,35], assisting visually impaired people with navigating the website's internal content [14], help in the task of mobile Web browsing [3,14,24,25,30,31], ad removal, Web page topic clustering [23], and the ubiquitous task of a Web search [5,6,27,29,34].

Our Web block classification system is known as BER$_y$L (**B**lock classification with **E**xtraction **R**ules and machine **L**earning). There are three main requirements that it must meet:

1. be able to cover a diverse range of blocks;
2. achieve acceptable precision and recall results for each individual block in the classification system, and maximise the overall performance of the system;
3. be adaptive to new block types and domains.

The task of Web block classification is technically challenging due to the diversity of blocks in terms of their internal structure (their representation in the DOM tree and the visual layout) and the split between domain-dependent and domain-independent blocks. Hence, from a technical perspective, it is important to have a global feature repository which can provide a framework for defining new features and block type classifiers through the instantiation of template relations. A BER$_y$L user will be able to extend it with new features and classifiers with ease by generating them from existing template relations, rather than writing them from scratch. This will make the whole hierarchy of features and classifiers leaner and the process of defining new block types and their respective classifiers more straightforward and less time-consuming. Ideally, we aim

to generate new block classifiers in a fully automated way such that, given a set of structurally and visually distinct Web blocks of the same type, the block classification system would be able to automatically identify the list of optimal features for describing that block, taking some of these features from the existing repository and generating new ones which did not exist in the repository beforehand. However, this approach would almost be infeasible in the case of BERyL since the diversity of block types that we want to classify is likely to cause the space of potential features to be extraordinarily large, if not infinite. Hence, we will need to limit the approach of the generation of new features and block type classifiers to a semi-automated approach.

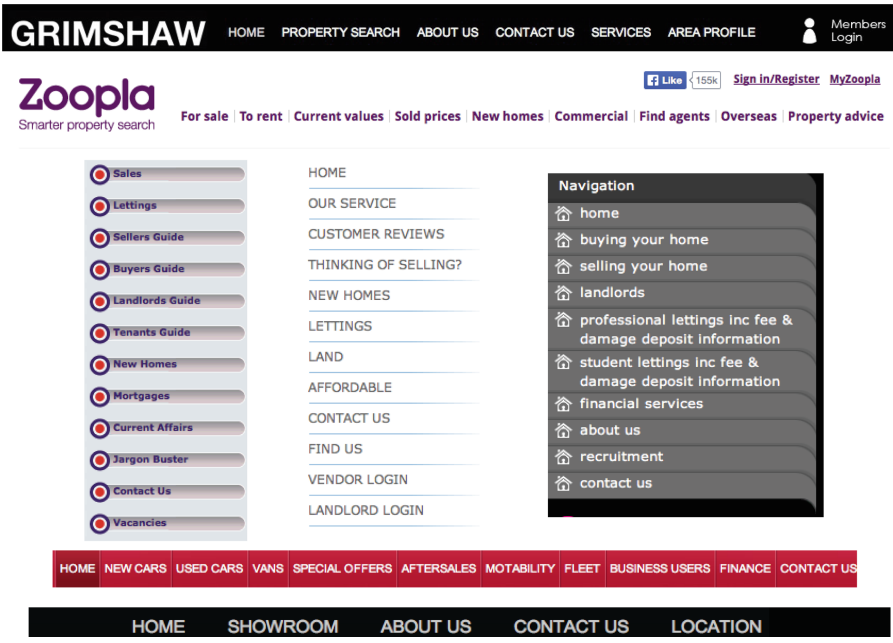

Fig. 1. The diversity of navigation menu types

The contributions of our approach in BERyL have two main aims: improving the quality of classification and minimising the effort of generating new classifiers (as explained in the above paragraph). Contributions 1–2 and 3–4 refer to the quality of classification and generation aspects respectively.

1. We employ domain-specific knowledge to enhance the accuracy and performance of Web block classifiers.
2. We provide template and global feature repositories which allow users of BERyL to easily add new features and block classifiers.
3. We encode domain-specific knowledge through a set of logical rules, e.g. that the navigation menu is always at the top or bottom of (and rarely to the side of) the main content area of the page.

4. The template and global feature repositories are implemented through baseline global features and template relations used to derive local block-specific features.

With respect to the employment of domain-specific knowledge for enhancing the performance of classifiers (contributions 1 and 3), our new approach to feature extraction allows us to easily integrate domain-specific pre- and post-classification filters which will ensure that the classifiers in question meet all the additional restrictions imposed by the domain in which they are applied.

We have also implemented template and global feature repositories (contributions 2 and 4) which are crucial to the automation and large-scale evaluation of the BER$_y$L system. The template repository is implemented through component-driven approach to feature extraction that is covered in detail in Sect. 3. Our system also supports baseline global features which can be shared between different classifiers. The template repository and baseline global features allow us to easily extend the system with new classifiers without reducing the accuracy of the classifiers that are already present in the system or significantly reducing the system's performance.

2 Related Work

There have been several papers published on this topic in the past ten years including [4,7,8,12,15–17,20–22,24,26,32,33,35,36]. Broadly speaking, Web block classification methods can be split into those based on machine learning (ML) techniques and those based on other approaches (i.e. the rule-based and heuristics). Also, the features used for the analysis of Web blocks in ML-driven and other algorithms can be subdivided into structural (based on the structure of a DOM tree or another intermediary structure representing a Web page), visual (based on the Web page's rendering), and lexical (based on the analysis of the page's free text).

Most of the approaches taken in this research have attempted to classify a relatively small set of domain-independent blocks with a limited number of features, whereas we aim to develop a unified approach to classify both domain-dependent and domain-independent blocks. Furthermore, none of the papers with which we are familiar discussed the extensibility of their approaches to new block types and features.

Finally, although most of these machine learning-based approaches require significantly more training than BER$_y$L, they usually reach a precision level below 70% [12,15,17,20,24]. The highest value of F_1 that any of these machine learning-based approaches reach (apart from [10] that reaches the F_1 of 99% but for a single very specific block type of "next" links and [18] that reaches the overall classification rate of 94% but for basic Web form elements within the transportation domain and requires a labour-intensive labelling and complex training procedure) is around 85% [24]. An evaluation of our BER$_y$L system (Sect. 4) has shown that it can achieve much higher levels of precision and recall. This can partly be explained by the fact that they have attempted to classify different

blocks with the same set of features, whereas we attempt to employ individual feature sets for different types of blocks.

Although most of the current block classification approaches are based on machine learning methods, to our knowledge there is no universal approach to the task of block classification.

3 Approach

The problem of Web block classification is, given a Web page P with associated annotated DOM T_P, to find a mapping from the set of sub-trees $T'_1, \ldots, T'_n$ of T_P to sets of labels, such that each sub-tree is labelled with, at most, one label per block type. The novelty of our approach is that we try to tailor feature sets for different Web blocks by application of declarative programming to feature extraction. We propose a component-driven approach combined with the use of Datalog-based BERyL language and powered by the use of block-specific knowledge. This, together with the repository of global features and relation templates, helps to create tailored features for different blocks and domains.

Component-Based Approach to Feature Extraction. We give a description of how our approach to feature extraction works in practice by introducing the concept of a **Component Model (CM)-driven approach to feature extraction**. Components are fragments of code that can be combined to allow for modular definition of complex features that we want to extract. Components can have different types (e.g. we distinguish between rule-based components that are mapped to Datalog programs and procedural components that are mapped to fragments of Java code). Components can also have parameters attached to them, which can either be atomic parameters, e.g. font size, sequential parameters, e.g. the tags of the DOM tree we want to consider for analysis, or higher-level parameters that can link to other components within the definition of a given component. Parameters are assigned values through *instantiations* $\mathfrak{C}[p \mapsto v]$, and for higher-level parameters we allow *references* as values, i.e. expressions of the shape $@n$ where n is the unique name of the referenced component.

Definition 1 (Component type). A *component type* $\mathfrak{C}$ is a triple $\langle I, O, U \rangle$ of two relation schemas I and O, the respective schema of the input and output relations for a component, and a set of formal parameters U.

Definition 2. A *formal component parameter* is a mapping of a unique name p, the parameter name, to a parameter type $\tau_p \in \mathscr{P}$. $\mathscr{P}$ is the set of permissible types and defined recursively as follows: Let $\mathscr{P}_A$ be the set of *atomic* types such as *Integer* or *String*. Then a permissible type is either an atomic type from $\mathscr{P}_A$, a sequence type $[\tau_i]$ where τ_i is itself another type from $\mathscr{P}$, or a component type.

$$\tau_p = \begin{cases} \alpha & \text{with } \alpha \in \mathscr{P}_A \\ [\tau_i] & \text{with } \tau_i \in \mathscr{P} \\ \mathfrak{C} & \text{with } \mathfrak{C} \text{ a component type} \end{cases}$$

For each type τ, we denote with $\mathsf{domain}(\tau)$ the set of permissible values for τ.

We call a parameter of atomic type an atomic parameter, of sequence type a sequence parameter, and of component type a component parameter. For an atomic type τ, $\mathsf{domain}(\tau)$ is the corresponding set of values. For a sequence type, it is the set of sequences over the values of the constituent type. For a component type, it is the set of (references to) ground components of type $\mathfrak{C}$. We call a component *generic* if it has at least one formal parameter, and *higher-order* if there is at least one component type parameter in U.

Let $\mathscr{N}$ be the set of component names. Then $\mathfrak{C}[p \mapsto v]$ is an *instantiation* for component type $\mathfrak{C}$ that binds the formal parameter with name p in $\mathfrak{C}$ to the value v, where $v \in \mathsf{domain}(p)$. We call an instantiation ground if all parameters in $\mathfrak{C}$ are bound to an actual value. For component parameters, we allow *references* as values, i.e. expressions of the shape $@n$ where n is the unique name of the referenced component.

Definition 3. Let $\mathscr{K}$ be the set of primitive components. A *(component) composition specification* describes how to combine components into more complex ones. A composition specification for a component type $\mathfrak{C}$ is then a triple $C = (n, \mathfrak{C}', \mathsf{E})$ where n is a unique name, $\mathfrak{C}'$ a ground instantiation for component type $\mathfrak{C} = (I, O, U)$, and E an expression that describes how to compose this component from other, more primitive ones. Specifically, a composition expression is an expression of the form:

$$\mathsf{E} = \begin{cases} k & \text{where } k \in \mathscr{K} \\ @n & \text{where } n \text{ is a reference to an already} \\ & \text{defined component} \\ \mathsf{E}_1 \lhd \ldots \lhd \mathsf{E}_\mathrm{r} & \text{where } \mathsf{E}_\mathrm{i} \text{ are composition expressions} \\ \mathsf{E}_1 \parallel \ldots \parallel \mathsf{E}_\mathrm{r} & \text{where } \mathsf{E}_\mathrm{i} \text{ are composition expressions} \\ \mathsf{E}_1 \oplus \ldots \oplus \mathsf{E}_\mathrm{r} & \text{where } \mathsf{E}_\mathrm{i} \text{ are composition expressions} \end{cases}$$

We call $\lhd$ isolated *sequential* composition, $\parallel$ *parallel*, but isolated composition, and $\oplus$ sequential, but *possibly dependent* composition.

We write $\textsc{type}(C) = \mathfrak{C}$, $\textsc{input}(C) = I$, and $\textsc{output}(C) = O$. Component expressions form trees (with possibly shared branches) and thus no cycles between components are possible. We refer to the name n assigned to a composition expression e as $\textsc{name}(e)$.

We refer to the components that are referenced in a composition expression as sub-components $C_1, \ldots, C_r$ and r the arity of C. We call the set of *defined sub-components* in E_C the set $\textsc{subs}(C)$ of all components defined in E_C or one of its sub-expressions, i.e. $\textsc{subs}(C) = \bigcup_{1 \le i \le r} \textsc{subs}(C_i) \cup \{(n_i : C_i) : 1 \le i \le r\}$.

A composition expression E_C is *valid*, if

1. the schemata of the sub-components are compatible: $\textsc{input}(C) = \textsc{input}(n_1)$, $\textsc{output}(n_{i-1}) = \textsc{input}(n_i)$ for all $1 < i \le r$, and $\textsc{output}(n_r) = \textsc{output}(C)$.

2. for each composition parameter $(p, \tau) \in \text{TYPE}(C)$ that is instantiated to component reference $@v$, there is a sub-component $v : C_v \in \text{SUBS}(C)$ such that $\text{TYPE}(C) = \tau$.

Given a component (and corresponding composition expression) the semantic of that component is then quite straightforward: in our framework, components are implemented either as declarative rules (rule-based components, written and evaluated as Datalog rules) or Java classes (procedural components). In both cases, implementations may query parameters of the surrounding component. For atomic parameters, the query returns the corresponding value, for sequence parameters it returns a suitable representation of a sequence in Datalog or Java, and for component parameters it returns an interface that allows the implementation to access the results of the other component.

Definition 4. Let $C = (n, \mathfrak{C}')$ be a component with corresponding composition expression $n : \mathsf{E}_C$ and U the set of parameters for $\text{TYPE}(C)$. Then, we denote with $[\![\mathsf{E}_C]\!](I)$ the output for C under E if executed on input I. The output is a set of pairs $n : O$ where n is a component name and O an instance of a component output schema. As such, we write $[\![\mathsf{E}_C]\!](I)[n]$ to access the output instance for the component with name n.

$$[\![n : \mathsf{E}_C]\!](I) = \{n : [\![\mathsf{E}_C]\!](I \cup U)[\text{NAME}(\mathsf{E}_C)]\}$$

$$[\![\mathsf{E}_C(I)]\!] = \begin{cases} \{n_1 : k(I)\} & \text{if } \mathsf{E} = k \in \mathscr{K} \\ \bigcup_{i \le r} [\![C_i]\!](I) & \text{if } \mathsf{E} = C_1 \parallel \ldots \parallel C_r \\ \bigcup_{i \le r} [\![C_r]\!]([\![C_{r-1}]\!](\ldots [\![C_1]\!](I) \ldots)[\text{NAME}(C_{r-1})]) & \text{if } \mathsf{E} = C_1 \lhd \ldots \lhd C_r \\ \bigcup_{i \le r} [\![C_r]\!]([\![C_{r-1}]\!](\ldots [\![C_1]\!](I) \ldots) \cup I) & \text{if } \mathsf{E} = C_1 \oplus \ldots \oplus C_r \end{cases}$$

We can now give a definition of a *component model*, which is a crucial part of our BERyL system, since each of its constituent classifiers corresponds to a unique component model.

Definition 5 (Component model). A component model is a set $\mathbb{M}$ of executable components such that for each component $C \in \mathbb{M}$ with a higher-order parameter p that references a component C', then $C' \in \mathbb{M}$ ("no dangling references").

Example 1. We give an example of a fragment of the real component model we use for the Next Link classifier (Fig. 2). Let us consider one of the most important features of the Next Link classifier that checks whether the direct left numeric visual sibling of a given numeric node is a link or not (if it is not a link that is a strong indicator that the numeric link in question is a numeric next link). We define this feature from the global template relation *two nodes with unary relation properties connected by a binary relation* that denotes two nodes connected to each other by a binary relation that also defines a unary property of the second node. Each of the nodes also holds one unary relation property of their own. That is a very generic template that can be used in

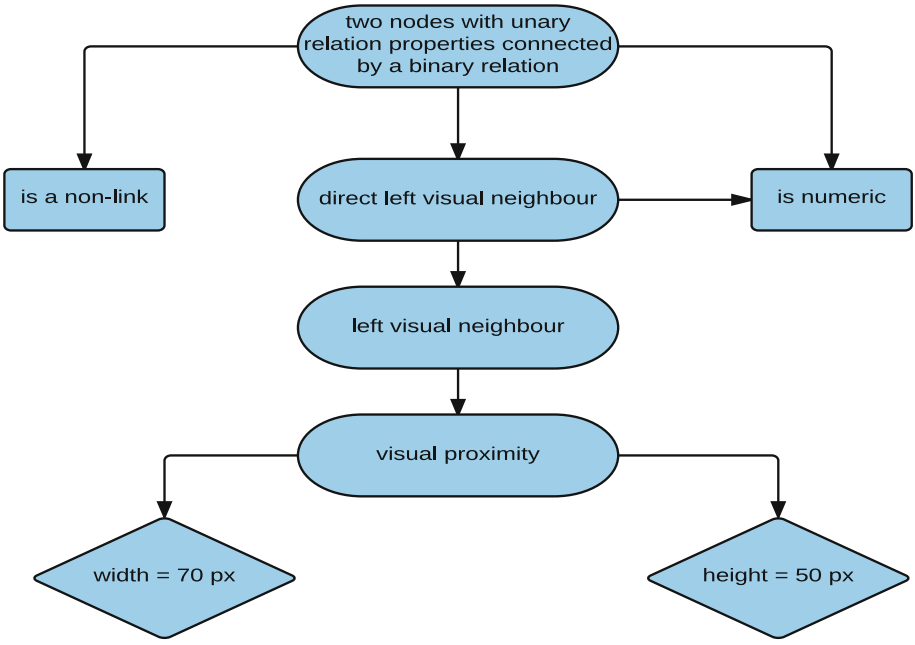

Fig. 2. An example of a component model for a single feature

many cases for different classifiers. In our case, the binary relation is the *direct left visual neighbour* relation with the property of the second node (the one to the visual left of the node in question) defined as *is numeric*. Note that the *direct left visual neighbour* binary relation, in turn, utilises the *visual proximity* binary relation that defines any visual proximity within the given boundaries regardless of direction. We set the width and height parameters for the visual proximity boundaries to 70 and 50 pixels respectively. The respective unary relation properties of the two nodes in question are *is numeric* for the first node and *is a non-link* for the second node.

The BER$_y$L *Language.* We provide a tool to the users of our system that allows them to define powerful BER$_y$L component models in an easy and intuitively understandable, declarative way. This BER$_y$L language is a version of Datalog with some extensions and is one of the key modules of our framework.

The BER$_y$L language is a dialect of Datalog that includes safe and stratified negation, aggregation, arithmetic, and comparison operators, denoted as $\mathsf{Datalog}^{\neg,\mathsf{Agg},\mathsf{ALU}}$. The usual restrictions apply, i.e. all variables in a negation, arithmetic operation, aggregation, or comparison must also appear in some other (positive) context. Further, there may be no dependency cycles over such operators. In the context of the BER$_y$L system, this language is used in a certain way:

Table 1. BER$_y$L's input facts

Structural:	
`dom::element(N,T,PStart,Start,End)`	DOM element node $N \in \mathcal{N}$ has tag $T{=}\tau(N)$, and parent node P, such that $(P, N) \in$ CHILD, with start label PStart, and start and end labels Start and End.[1]
`dom::attribute(A,N,T,V)`	DOM attribute node A of node N with $(N,A) \in$ ATTR has tag $T{=}\tau(A)$ and value $V{=}\sigma(A)$.
`dom::clickable(N)`	N is a clickable target (a link or has an onclick handler, i.e. $(\tau(N){=}a) \vee (\exists(N,A) \in \text{ATTR} \wedge \tau(A){=}onclick)$.
`dom::content(N,v,O,L)`	Node N corresponds to textual content v with $(N,N_i) \in \text{CHILD}^+ \wedge \neg\exists(N_i,N_j) \in \text{CHILD} \wedge (\sigma(N_1)\text{++}\ldots\text{++}\sigma(N_n){=}v)$[2], nodes N_i are sorted in the ascending order of their Start labels, node N starts at document offset O and has length L.
Visual:	
`css::box(N,Left,Top,Right,Bottom)`	bounding box of a DOM node N, such that $\exists(N,C_1),(N,C_2),(N,C_3),(N,C_4) \in \text{CSS}:$ $\tau(C_1){=}left\_coord \wedge \tau(C_2){=}top\_coord \wedge$ $\tau(C_3){=}right\_coord \wedge \tau(C_4){=}bottom\_coord \wedge$ $\sigma(C_1){=}Left \wedge \sigma(C_2){=}Top \wedge$ $\sigma(C_3){=}Right \wedge \sigma(C_4){=}Bottom$.
`css::page(Left,Top,Right,Bottom)`	bounding box of a DOM node N with $\tau(N){=}html$.
`css::resolution(1800,800)`	the average screen resolution.[3]
`css::font_family(N,Family)`	N is rendered with a font from the given family $(\exists(N,C) \in \text{CSS} \wedge \tau(C){=}font\_family \wedge \sigma(C){=}Family)$.
`css::font_size(N,Size)`	N is rendered with a font of the given size $(\exists(N,C) \in \text{CSS} \wedge \tau(C){=}font\_size \wedge \sigma(C){=}Size)$.
Content:	
`gate::annotation(N,A,v)`	the text of node *N* contains an instance of entity type *A* and the normalised representation of that instance is *v*.
Classification:	
`cls::classification(N,`*BT*`,Label)`	a classification $(N,Label) \in$ CLASS where $BT \in \mathbb{BT}$ is a block type with $Label \in BT$. For instance, the pagination classifier corresponds to the plm (pagination link model) namespace, e.g. `cls::classification(e_200,pagination, numeric_next_link)` and `cls::classification(e_300,pagination,non_num_next_link)`. Different classifiers can produce output on the same node, e.g. `cls::classification(e_100,header,header)` and `cls::classification(e_100,navigation, nav_menu)`.

[a]Start and end labels of a node correspond to a pre-order traversal of the DOM tree with a single incremental counter that assigns the start label the first time the node has been explored, and the end label when all the node's descendants have been explored.
[b]++ is a concatenation operator.
[c]According to a study by http://www.w3schools.com/browsers/browsers_display.asp

1. there is a number of input facts for representing the annotated DOM (see Table 1), as well as the output of previous components, such as the classification facts in Table 1;
2. each program carries a distinguished set of output predicates and only entailed facts for those predicates are ever returned.

The BER$_y$L language also uses a number of syntactic extensions to ease the development of complex rule programs. The most important ones are *namespaces*, *parameters*, and the explicit distinction of *output predicates*. All other intensional predicates are temporary ("helper") predicates only. For the purpose of brevity, we omit the precise definitions of these extensions.

Specifically, there are two classes of components and component types, which are used heavily in BER$_y$L language programs: components representing (1) relations and (2) features. These components operate on the universe $\mathscr{U}$ of DOM nodes and atomic values appearing in the DOM of a page. We typically use relations to either (a) distinguish sets of nodes, (b) relate sets of nodes to each other, or (c) attach additional information to nodes.

The Standard Library of the BER$_y$L *Language.* The BER$_y$L language also provides a set of predefined components that a specific block type may import, which are defined through the input facts from Table 1 and are written in the `relation` namespace. We call these predefined components **standard relations**. Figure 3 shows some of the standard relations in BER$_y$L: these range from structural relations between nodes (similar to XPath relations) over visual relations (such as proximity) to information about the rendering context (such as the dimensions of the first and last screen).

Example 2. Let us reconsider Example 1. The graph representation of the component model for this feature is given in Fig. 2. For the purpose of conciseness in our notation we abbreviate the main component and its constituent subcomponents in the following way: (a) is numeric as C_1, (b) is a non-link (or non-clickable) as C_2, (c) visual proximity as C_3, (d) left visual neighbour as C_4, (e) direct left visual neighbour as C_5, and (f) two nodes with unary relation properties connected by a binary relation as C_6. C_1–C_2 are unary relation components, whilst C_3–C_6 are binary relation components. We give the rules for these components in Fig. 4.

C_1 and C_2 are primitive components with no parameters attached to them and therefore have trivial empty instantiations and identity-binding expressions:

```
1 𝔈1 [] ; C1
2 𝔈2 [] ; C2
```

The rules of C_3 and C_4 correspond to standard binary relations `relation::visual_proximity` and `relation::left_visual_neighbour` as defined in Fig. 3. C_3 has two parameters dH and dV that represent the possible positions of the top-left coordinate of CSS boxes corresponding to node considered to be in the visual proximity of the CSS box of the given node. The instantiations and composition expressions of these two components are the following:

```
relation::preceding(Id,X,Y) ⇐
  dom::element(X,_,_,_,End),
  dom::element(Y,_,_,Start,_), End < Start,
  param::instantiation_id(Id).
relation::descendant(Id,X,Y) ⇐
  dom::element(X,_,_,StartX,EndX),
  dom::element(Y,_,_,StartY,EndY), StartY < StartX,
  EndX < EndY,
  param::instantiation_id(Id).
relation::leaf_descendant(Id,X,Y) ⇐ dom::element(X,_,_,_,_),
  relation::descendant(CId1,X,Y),
  ¬relation::descendant(CId2,Z,X),
  param::instantiation_id(Id).
relation::visual_proximity(Id,X,Y) ⇐
  css::box(X,LeftX,TopX,_,_),
  css::box(Y,LeftY,TopY,_,_),
  TopY-DVert ≤ TopX ≤ TopY+DVert,
  LeftY-DHor ≤ LeftX ≤ LeftY+DHor,
  param::dH(PId1,DHor), param::dV(PId2,DVert),
  param::instantiation_id(Id).
relation::left_visual_neighbour(Id,X,Y) ⇐
  relation::visual_proximity(CId,X,Y),
  css::box(X,_,_,RightX,_), css::box(Y,LeftY,_,_,_),
  RightX ≤ LeftY, param::instantiation_id(Id).
relation::first_screen(Id,Left,Top,Right,Bottom) ⇐
  css::page(Left,Top,_,_), css::resolution(H,V),
  Right = Left+H, Bottom = Top+V, param::instantiation_id(Id).
relation::last_screen(Id,Left,Top,Right,Bottom) ⇐
  css::page(_,_,Right,Bottom), css::resolution(H,V),
  Left = Right-H, Top = Bottom-V, param::instantiation_id(Id).
```

Fig. 3. The standard library of the BERyL language

$\mathfrak{C}_3$[dH↦70, dV↦50]; C_3
$\mathfrak{C}_4$[]; $C_3 \lhd \mathfrak{C}_4$

We now give the instantiation and the binding expression of binary relation component C_5 that specifies whether one node is a direct left visual neighbour of another node and has a single unary relation component parameter:

$\mathfrak{C}_5$[sibling_pred↦ C_1]; $((C_3 \lhd C_4) \parallel C_1) \lhd C_5$

Finally, we give the instantiation and binding expression of binary relation component C_6 according to our example:

$\mathfrak{C}_6$[binary_pred↦ C_5, node_pred↦ C_1, sibling_pred↦ C_2];
$((((C_2 \parallel C_3) \oplus C_4) \oplus C_5) \parallel C_2) \lhd C_6$

Note that it seems more obvious to define the composition expression for C_5 as $((C_1 \parallel C_3) \lhd C_4) \lhd C_5$, but in that case we would have to recompute

```
C1: relation::numeric(Id1,X) ⇐ gate::annotation(X,"NUMBER",_),
      param::instantiation_id(Id1).
C2: relation::non_clickable(Id2,X) ⇐ dom::element(X,_,_,_,_),
      ¬(dom::clickable(X)), param::instantiation_id(Id2).
C3: relation::visual_ proximity(Id3,X,Y)
C4: relation::left_visual_neighbour(Id4,X,Y)
C5: relation::direct_left_visual_neighbour(Id5,X,Y) ⇐
      relation::left_visual_neighbour(CId1,X,Y),
      ¬relation::indirect_left_visual_neighbour(CId2,X,Y),
      param::sibling_pred(CId3,Y),
      param::instantiation_id(Id5).
    relation::indirect_left_visual_neighbour(CId2,X,Y) ⇐
      relation::left_visual_neighbour(CId4,X,Y),
      relation::left_visual_neighbour(CId5,X,Z),
      css::box(Y,LeftY,_,RightY,_), css::box(Z,LeftZ,_,RightZ,_),
      RightY ≤ LeftZ, RightX ≤ LeftY,
      param::instantiation_id(CId2).
C6: relation::binary_unary(Id6,X,Y) ⇐
      param::binary_pred(CId1,X,Y),
      param::node_pred(CId2,X),
      param::sibling_pred(CId3,Y),
      param::instantiation_id(Id6).
```

Fig. 4. Rules corresponding to components C_1–C_6

C_1 at the time of evaluation of the final component C_6, and the composition expression for C_6 would have been $((((C_1 \parallel C_3) \triangleleft C_4) \triangleleft C_5) \parallel (C_1 \parallel C_2)) \triangleleft C_6$, as the semantics of the $\triangleleft$ operator would have restricted visibility of C_2 at the time of the computation of C_6, and we would have encountered an additional computational overhead.

Declarative Approach to Feature Extraction. In BER$_y$L, we use a declarative approach to feature extraction wherever possible, in particular through the BER$_y$L language described above, since that (**1**) allows us to combine it with relations and global features which provide a succinct representation of the current feature set. This, in turn, allows us to simplify the definition of new features through the employment of existing global features or relation instantiations and to learn new features by automatically finding the right combination of parameters for relation instantiations. (**2**) It is also much easier to learn Datalog [1,9] predicates automatically than to learn procedural language programs, which is likely to come in useful in the large-scale block classification phase of BER$_y$L when we will have to infer new features automatically. In other cases which require the use of efficient libraries and data structures or intense numerical computation (e.g. features acquired from image processing), we employ a procedural approach for feature extraction implemented through Java.

We use Datalog as the declarative language of choice, since it is fast and widely used [9,13]. Also, with a Datalog-based approach to feature extraction we

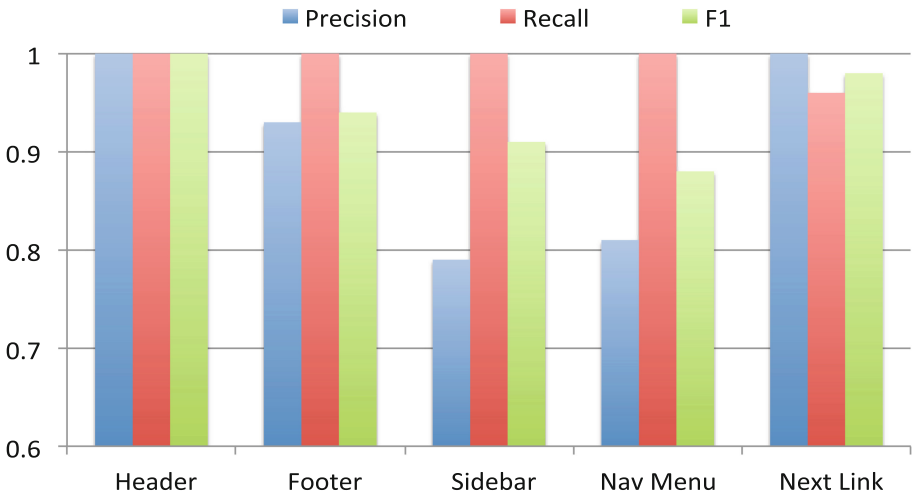

Fig. 5. BERyL's accuracy results on the five classifiers

can easily extend our extraction rules to run on databases, which can come in useful if we have to process large sets of training or evaluation data.

Web Block Classification. It does not seem feasible to solve the block classification problem through a set of logic-based rules, as it is often the case that there are many potential features which can be used to characterise a specific block type. However, only a few play a major role in uniquely distinguishing this block type from all others. Some of these features are continuous (e.g. the block's width and height), and it can be difficult for a human to manually specify accurate threshold boundaries. Hence, for BERyL, we decided to use a machine learning (ML) approach to Web block classification.

Comparative Analysis of Machine Learning Techniques. The current version of the BERyL Web block classification system uses the C4.5 Decision Tree as the classification model, as it allows us to check how the features are used in rules that guide the overall classification, which is not the case with other ML classification models such as SVMs. Also, the rules generated by the C4.5 Decision Tree can be easily translated into Datalog rules, which can become useful if we decide to run the ML classification stage of BERyL in Datalog, same as the feature extraction stage.

4 Evaluation

In Fig. 5 we present the evaluation results for five domain-independent block classifiers (headers, footers, sidebars, navigation menus, and next links) obtained on 500 randomly-selected pages from four different domains (Real Estate, Used

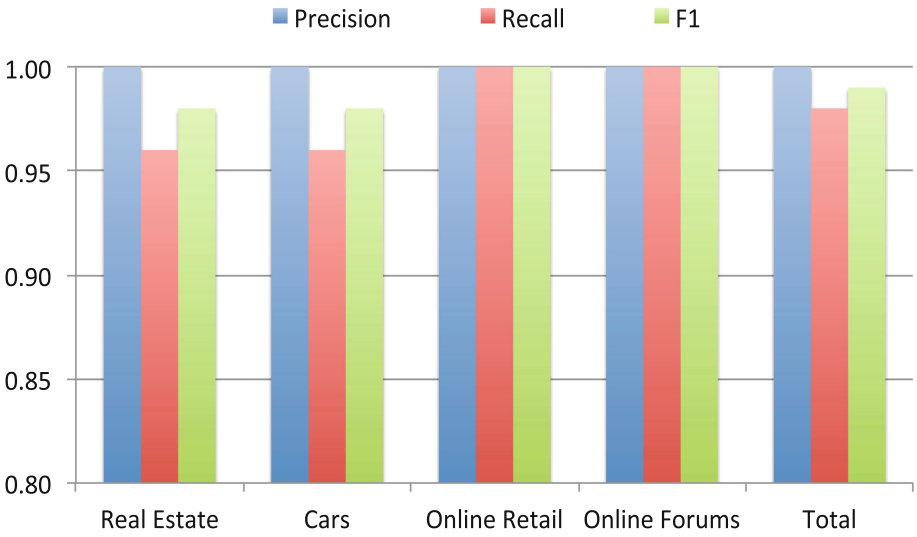

Fig. 6. Precision and recall of the pagination link model

Cars, Online Retail, Blogs and Forums). For each block type and domain, the pages have been selected randomly from a listings page (such as yell.com) or from a Google search. The latter favours popular websites, but that should not affect the results presented here. For all these classifiers we have achieved good precision, recall, and F_1 scores. All the classifiers have precision and recall scores above 80% apart from the Sidebar classifier, which has a precision score of just below 80%. This can be explained by the fact that a sidebar usually does not have obvious clues, such as the NEXT annotation for non-numeric next links, and therefore it is harder to distinguish True Positives from False Positives and False Negatives. Note that the Next Link and Header classifiers achieve a perfect precision of 100%. This can be explained by the fact that we use a highly tailored feature set that includes features specific to the Next Link block, and that headers are very distinct from other blocks, and our fairly simple set of six features used for the classification of this block is sufficient to achieve a perfect separation between header and non-header DOM nodes.

Accuracy of Pagination Link Classification. We put a special emphasis on the Pagination Link classifier, due to its importance to the DIADEM system, and hence have performed a more detailed evaluation for this block type than for the other three. We present detailed evaluation results for the Pagination Link classifier in Fig. 6, which illustrates that in all four domains our approach achieves 100% precision, and recall is never below 96%. This high accuracy means that our approach can be used to crawl or otherwise navigate paginated websites with a very low risk of missing information or retrieving unrelated Web pages. Numeric pagination links are generally harder to classify than non-numeric ones, due to their greater variety and the larger set of candidates. Though precision

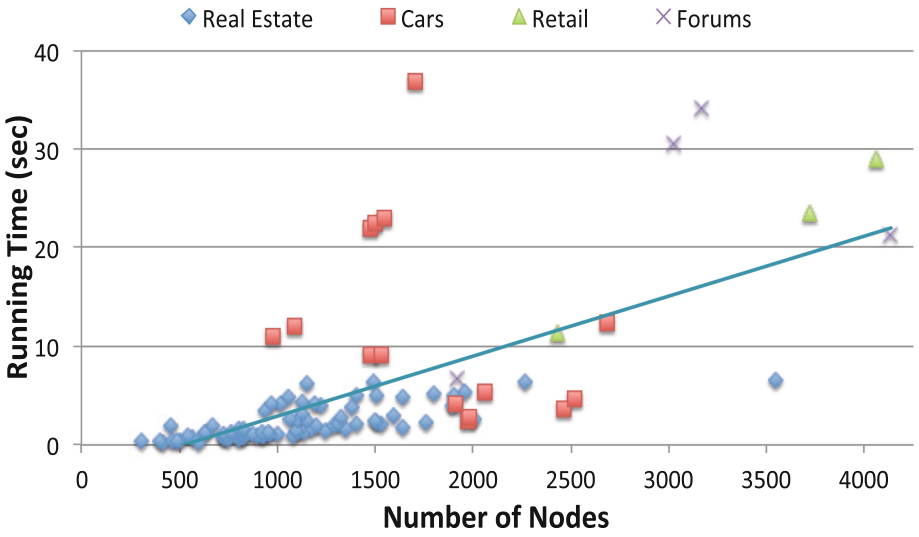

Fig. 7. Performance of the pagination link model

is 100% for both cases, recall is on average slightly lower for numeric pagination links (98% vs. 99%) and in some domains quite notable (e.g. Real Estate with 96% vs. 99%).

Performance Results. The speed of feature extraction is crucial for the scalability of our approach to allow the crawling of entire websites. As discussed above, the use of visual features by itself imposes a certain penalty, as a page needs to be rendered for those features to be computed. We present the performance results of BERyL in Fig. 7, which shows, for the example of the Pagination Link classifier, that the performance is highly correlated to page size, with most reasonably sized pages being processed in well below 10 seconds (including page fetch and rendering). It is interesting to observe that the domains for which we use Google to generate the corpus, and for which the corpus is thus biased towards popular websites, seem to require more time than the Real Estate domain, for which the corpus is randomly picked from yell.com.

Comparison to Other Approaches. We compare the precision, recall, and F_1 results achieved by the BERyL system with the results of other systems that were covered in Sect. 2. Some of the papers analysed there do not present the accuracy results, so we omit them from this comparison, and for others we compare to the approaches, which cover the block types also covered by the BERyL system, and if there is no overlap between the block types covered, we compare on the average accuracy results for all classifiers in the system. As a lot of block types covered by our system are not covered by other approaches, we compare the precision, recall, and F_1 metrics over two block types where there is an overlap (navigation menus and pagination bars), and the average precision, recall, and F_1 metrics for

entire classification systems. If there is no data for a specific metric or classifier for the method we are benchmarking against, we indicate this by N/A in the relevant cell of Table 2. In all cases BER$_y$L achieves higher precision and recall results for individual classifiers, as well as average precision and recall results for all classifiers. Our intuition is that the better results achieved by our system are due to the fact that we use distinct feature vectors for each of the block types, and each of these feature vectors is highly tailored to the block type it is being extracted on, whilst other approaches attempt to use a single coarse-grained feature vector for all block types their systems classify.

Table 2. Precision, recall, and F_1 of the BER$_y$L system compared to other systems

	Pagination bars			Navigation menus			System average		
	Precision	Recall	F_1	Precision	Recall	F_1	Precision	Recall	F_1
BER$_y$L	1.00	1.00	1.00	0.88	1.00	0.94	0.97	1.00	0.98
[15]	0.42	0.96	0.58	0.98	0.37	0.54	0.73	0.65	0.69
[24]	N/A	N/A	N/A	N/A	N/A	0.88	N/A	N/A	0.75
[20]	N/A	N/A	0.82	N/A	N/A	0.82	N/A	N/A	0.52
[12]	N/A	N/A	N/A	N/A	N/A	N/A	0.77	0.71	0.74
[17]	N/A	N/A	N/A	N/A	N/A	N/A	0.75	0.66	0.70

5 Conclusion

We propose a Web block classification system, BER$_y$L, which utilises global feature and template repositories for providing substantial improvement in the manual effort of defining new features and improving the performance of feature extraction.

We aim to pursue multiple avenues for future research, in particular (**1**) further exploration of how domain knowledge can improve block classification and the differentiation between domain-independent and domain-dependent classifiers, (**2**) exploration of holistic approach to Web block classification, which imposes constraints between different block types, e.g. that a footer cannot be above a header, (**3**) exploration of the α-accuracy problem, which allows the system to find an optimal balance between the accuracy and performance expectations set by the user, and (**4**) automatic learning of features for individual block classifiers.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley Longman Publishing Co. Inc., Boston (1995)

2. Baumgartner, R., Flesca, S., Gottlob, G.: Visual web information extraction with Lixto. In: VLDB (2001)
3. Baluja, S.: Browsing on small screens: recasting web-page segmentation into an efficient machine learning framework. In: WWW 2006 (2006)
4. Burget, R., Rudolfova, I.: Web page element classification based on visual features. In: 2009 First Asia Conference on Intelligent Information and Database Systems (2009)
5. Cai, D., Yu, S., Wen, J., Ma, W.: Block-based web search. In: SIGIR 2004, 25–29 July 2004 (2004)
6. Cai, D., He, X., Wen, J., Ma, W.: Block-level link analysis. In: SIGIR 2004, 25–29 July 2004 (2004)
7. Cao, Y., Niu, Z., Dai, L., Zhao, Y.: Extraction of informative blocks from web pages. In: ALPIT 2008 (2008)
8. Chen, J., Zhou, B., Shi, J., Zhang, H., Fengwu, Q.: Function-based object model towards website adaptation. In: WWW 2010, 1–5 May 2010 (2010)
9. de Moor, O., Gottlob, G., Furche, T., Sellers, A. (eds.): Datalog 2.0 2010. LNCS, vol. 6702. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24206-9
10. Furche, T., Grasso, G., Kravchenko, A., Schallhart, C.: Turn the page: automated traversal of paginated websites. In: Brambilla, M., Tokuda, T., Tolksdorf, R. (eds.) ICWE 2012. LNCS, vol. 7387, pp. 332–346. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31753-8_27
11. Furche, T., et al.: DIADEM: domain-centric, intelligent, automated data extraction methodology. In: WWW 2012 (2012)
12. Goel, A., Michelson, M., Knoblock, C.A.: Harvesting maps on the web. Int. J. Doc. Anal. Recognit. **14**(4), 349 (2011)
13. Gottlob, G., Orsi, G., Pieris, A., Šimkus, M.: Datalog and its extensions for semantic web databases. In: Eiter, T., Krennwallner, T. (eds.) Reasoning Web 2012. LNCS, vol. 7487, pp. 54–77. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33158-9_2
14. Gupta, S., Kaiser, G., Neistadt, D., Grimm, P.: DOM-based content extraction of HTML documents. In: WWW 2003, 20–24 May 2003 (2003)
15. Kang, J., Choi, J.: Block classification of a web page by using a combination of multiple classifiers. In: Fourth International Conference on Networked Computing and Advanced Information Management, 2–4 September 2008 (2008)
16. Kang, J., Choi, J.: Recognising informative web page blocks using visual segmentation for efficient information extraction. J. Univ. Comput. Sci. **14**(11), 1893 (2008)
17. Keller, M., Hartenstein, H.: GRABEX: a graph-based method for web site block classification and its application on mining breadcrumb trails. In: 2013 IEEE/WIC/ACM International Conferences on Web Intelligence (WI) and Intelligent Agent Technology (IAT) (2013)
18. Kordomatis, I., Herzog, C., Fayzrakhmanov, R.R., Krüpl-Sypien, B., Holzinger, W., Baumgartner, R.: Web object identification for web automation and meta-search. In: WIMS 2013 (2012)
19. Krüpl-Sypien, B., Fayzrakhmanov, R.R., Holzinger, W., Panzenböck, M., Baumgartner, R.: A versatile model for web page representation, information extraction and content re-packaging. In: DocEng 2011, 19–22 September 2011 (2011)
20. Lee, C.H., Kan, M., Lai, S.: Stylistic and lexical co-training for web block classification. In: WIDM 2004, 12–13 November 2004 (2004)
21. Li, C., Dong, J., Chen, J.: Extraction of informative blocks from web pages based on VIPS. J. Comput. Inf. Syst. **6**(1), 271 (2010)

22. Liu, W., Meng, X.: VIDE: a vision-based approach for deep web data extraction. IEEE Trans. Knowl. Data Engineering **22**(3), 447 (2010)
23. Luo, P., Lin, F., Xiong, Y., Zhao, Y., Shi, Z.: Towards combining web classification and web information extraction: a case study. In: KDD 2009, 28 June–1 July (2009)
24. Maekawa, T., Hara, T., Nishio, S.: Image classification for mobile web browsing. In: WWW 2006, 23–26 May (2006)
25. Romero, R., Berger, A.: Automatic partitioning of web pages using clustering. In: Brewster, S., Dunlop, M. (eds.) Mobile HCI 2004. LNCS, vol. 3160, pp. 388–393. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28637-0_43
26. Song, R., Liu, H., Wen, J., Ma, W.: Learning block importance models for web pages. In: WWW 2004, 17–22 May (2004)
27. Vadrevu, S., Velipasaoglu, E.: Identifying primary content from web page and its application to web search ranking. In: WWW 2011 (2011)
28. Wang, J., et al.: Can we learn a template-independent wrapper for news article extraction from a single training site? In: KDD 2009, 28 June–1 July (2009)
29. Wu, C., Zeng, G., Xu, G.: A web page segmentation algorithm for extracting product information. In: Proceedings of the 2006 IEEE International Conference on Information Acquisition, 20–23 August 2006 (2006)
30. Xiang, P., Yang, X., Shi, Y.: Effective page segmentation combining pattern analysis and visual separators for browsing on small screens. In: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence (2006)
31. Xiang, P., Yang, X., Shi, Y.: Web page segmentation based on gestalt theory. In: 2007 IEEE International Conference on Multimedia and Expo (2007)
32. Yang, X., Shi, Y.: Learning web block functions using roles of images. In: Third International Conference on Pervasive Computing and Applications, 6–8 October 2008 (2008)
33. Yi, L., Liu, B., Li, X.: Eliminating noisy information in web pages for data mining. In: SIGKDD 2003, 24–27 August 2003 (2003)
34. Yu, S., Cai, D., Wen, J., Ma, W.: Improving pseudo-relevance feedback in web information retrieval using web page segmentation. In: WWW 2003, 20–24 May 2003 (2003)
35. Zheng, S., Song, R., Wen, J., Giles, C.L.: Efficient record-level wrapper induction. In: CIKM 2009, 2–6 November 2009 (2009)
36. Zhu, J., Nie, Z., Wen, J., Zhang, B., Ma, W.: Simultaneous record detection and attribute labeling in web data extraction. In: KDD 2006, 20–23 August 2006 (2006)