Automatic Extraction of Web Data Records Containing User-Generated Content*

Xinying Song†, Jing Liu†, Yunbo Cao‡, Chin-Yew Lin‡, and Hsiao-Wuen Hon‡
† Harbin Institute of Technology, Harbin 150001, P.R.China
‡ Microsoft Research Asia, Beijing 100190, P.R.China
xysong@mtlab.hit.edu.cn, jliu@ir.hit.edu.cn, {yunbo.cao, cyl, hon}@microsoft.com

ABSTRACT

In this paper, we are concerned with the problem of automatically extracting web data records that contain user-generated content (UGC). In previous work, web data records are usually assumed to be well-formed with a limited amount of UGC, and thus can be extracted by testing repetitive structure similarity. However, when a web data record includes a large portion of free-format UGC, the similarity test between records may fail, which in turn results in lower performance. In our work, we find that certain domain constraints (e.g., *post-date*) can be used to design better similarity measures capable of circumventing the influence of UGC. In addition, we also use anchor points provided by the domain constraints to improve the extraction process, which ends in an algorithm called MiBAT (Mining data records Based on Anchor Trees). We conduct extensive experiments on a dataset consisting of forum thread pages which are collected from 307 sites that cover 219 different forum software packages. Our approach achieves a precision of 98.9% and a recall of 97.3% with respect to post record extraction. On page level, it perfectly handles 91.7% of pages without extracting any wrong posts or missing any golden posts. We also apply our approach to comment extraction and achieve good results as well.

Categories and Subject Descriptors

H.3.m [**Information Storage and Retrieval**]: Miscellaneous - Data Extraction; Web

General Terms

Algorithms, Performance, Experimentation

Keywords

User-generated content, information extraction, structured data

*This work was done when Xinying Song and Jing Liu were visiting students at Microsoft Research Asia.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.
Copyright 2010 ACM 978-1-4503-0099-5/10/10 ...$10.00.

1. INTRODUCTION

Web 2.0, web applications that encourage user participation, is a well known concept nowadays and is becoming more and more popular. Along with its popularity, enormous valuable knowledge and information, which we call user-generated content (UGC), has been accumulated over years and still keeps growing. Extracting this valuable web data in an automatic and scalable manner can benefit a lot of applications like question answering [22], blog or review mining [10], and expert search on web communities.

Typically, web pages generated by Web 2.0 applications contain a large amount of UGC, such as forum posts, blogs, reviews, comments, etc. According to Wikipedia, UGC refers to "various kinds of media content, publicly available, that are produced by end-users", and thus has high diversity in both content and format. In this paper we focus on tackling the complexity of extracting web data records containing UGC. Hereafter, for the ease of presentation, we will use as our primary example the application of extracting posts from web forums (as shown in Fig. 1) although our approach can be applied to other types of applications as well.

Figure 1: A typical web forum thread page, showing two posts and one embedded advertisement bar

Web data extraction has been a hot research topic [4] in recent years. Recent work mainly follows two categories of approaches: semi-automatic and fully automatic. Semi-automatic approaches require manually labeled data for ei-

ther learning extraction rules [11], inducing wrappers based on a tree-structured template [6, 8, 25, 26], or training supervised statistical models on a specific domain [20, 27]. Due to the laborious nature of labeling, such semi-automatic approaches are not scalable for web scale data extraction.

In contrast, fully automatic approaches do not require any labeled data. Such approaches mainly study two subcategories of problems: (1) extracting a list of data objects (records) from a single page and (2) learning a template from multiple pages of the same type [2, 7]. The problem we study in this paper falls into the first sub-category. One of the representative approaches is MDR (Mining Data Records in Web pages) [12, 13] (including its extension work [14, 17, 23]). On the basis of MDR we are to develop our own approach. MDR identifies a list of records by conducting a similarity test against a pre-defined threshold for two sub-trees in the DOM tree of a web page. Such a method is referred to as the similarity-based approach [15], because the underlying assumption is that data records belonging to the same list usually have similar DOM tree structures.

Web data records containing UGC usually consist of two parts: well-formatted structured data (e.g., author, publication date, etc.), referred to as the template part, and free-format unstructured UGC. Due to the existence of UGC, the values of similarity between data records may vary greatly, which makes it less practical to set a good and robust similarity threshold and thus results in failure of the similarity-based approach. Fig. 2 shows the tree alignment for the two posts in Fig. 1. We can see that the two records look dissimilar due to the existence of the large portion of UGC.

Intuitively the problem can be solved if we are able to differentiate the structured template from unstructured UGC on DOM trees and use the template part to perform the similarity test. However, it is not easy to make such differentiation in an accurate and robust way.

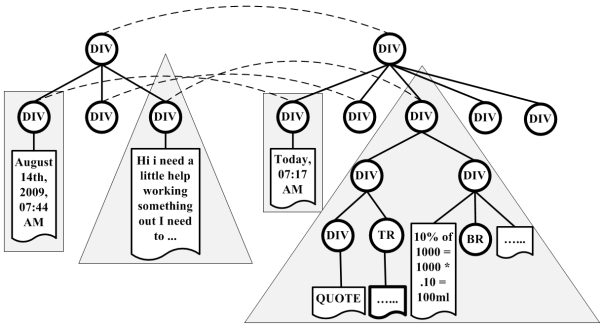

Figure 2: Tree match of two posts (gray triangles denote UGC while gray rectangles denote *post-date*)

Inspired by domain dependent work [20, 27], we find that some domain dependent constraints help detect the appropriate part of the tree for the similarity test. For example, for extracting posts from web forums, a good and intuitive constraint will be the *post-date* (publication date of a post) because it is a part of the structured data of posts occurring in every post and also can be easily identified (Fig. 2). Motivated by this intuition, we propose two similarity measures to solve the difficulty caused by UGC. Note that, in addition to forums, almost all types of web data records containing UGC have *post-date*, such as blogs, user comments (e.g. Twitter, Flickr, YouTube, Digg) or reviews (e.g. Amazon), etc. Therefore, our proposal is not restricted to forum sites.

Domain constraints also provide strong anchor point information for data record detection. For example, each forum post must contain exactly one sub-tree containing *post-date*. We proposed a novel data record extraction algorithm inspired by this intuition.

In summary, in this paper we aim to solve the problem of extracting from a single page a list of web data records that contain UGC in a fully automatic way. Previous work in this topic usually focuses on data objects containing no UGC, for example product lists [13, 23], search engine results [5, 17, 24] or DBLP literature reference records [15]. None of them explicitly claim to take care of the UGC part. Yang et al. [20] work on forum data extraction but in a semi-automatic way. Our contributions are as follows:

- We formulate similarity measures and propose to incorporate domain constraints to help design good similarity measures, on the basis of which an MDR-like similarity-based approach can overcome the similarity test issue caused by UGC (Sec. 4).
- We propose a novel mining algorithm called MiBAT which makes use of domain constraints to acquire anchor point information. Compared to MDR, MiBAT can not only extract non-consecutive data records, but also overcomes MDR's greedy deficiency [15] (Sec. 5).
- We develop a dataset collected from 307 forum sites formatted in 219 different forum software packages, on which our method achieves a satisfactory result of 98.9% in precision and 97.3% in recall (Sec. 6.1). To the best of our knowledge, this is the most comprehensive evaluation on forum post extraction.

2. RELATED WORK

Web data extraction has been an extensively studied research topic in recent years, resulting in a rich variety of approaches. We discuss highly relevant work here and refer the readers to a survey [4] for further study.

Early work on automatically extracting data records from a single page employs a set of heuristic rules to identify data record boundaries, including [9] and OMINI [3]. Later work is based on repetitive pattern mining from HTML tag sequences, such as IEPAD [5] and Dela [19]. Recent work is based on similar sub-tree mining on the DOM tree of the web page, represented by MDR [13]. It is reported in [13] that MDR outperforms both OMINI and IEPAD.

Due to its simplicity and effectiveness, MDR has attracted wide research interests and been extended in many studies. One improvement direction is incorporating visual layout information [17, 23, 24]. However, visual features usually require proper rendering with additional resources (such as CSS files), thus not being always available and generally helpful. Our work in this paper is purely based on the DOM tree structure without incorporating any visual features. We will show by experimental results that such a pure tag-tree based approach achieves satisfactory performance as well.

Web data can be a relation of k-tuple (where each record has k attributes), or a complex object with a hierarchical structure like nested lists [4]. The former is called *flat* and the latter *nested*. In this paper we mainly focus on flat data, for web data records containing UGC are usually displayed

in a flat fashion. Nested objects can also be naturally handled by an extention of using a post-order traversal along the DOM tree as shown in [14].

Though working with different motivations, Miao et al. [15] also address similar issues of MDR, i.e. the similarity test issue and the greedy manner (Sec. 3.1), but in a different way. They transform the input page from a DOM tree into tag path occurrence patterns (called visual signals) and identify the set of tag paths that represent a list of objects by applying spectral clustering. However, visual signals are easily affected by not only the number of data records present on the page but also the noisy tree nodes sharing the same tag paths. Therefore it is hard to guarantee the performance of the clustering on the visual signal set.

Our key idea is employing domain constraints to give strong clues to extract data records accurately. The proposed *post-date* is just one kind of domain constraint, which we will explain in Sec. 4.2. Interestingly we have noticed some related work that utilizes this idea either implicitly or in different scenarios. Embley et al. [9] utilize an ontology heuristic that one or more fields appear once and only once in each record; this resembles the concept of domain constraints in our work. However, their motivation is to get an estimation of the number of records on the page, which is combined with other heuristics like identifiable "separator" tags to discover the record boundary. In contrast, in our work we leverage domain constraints to inspect the structure of the template part of records, based on which the similarity test can be conducted and records can be determined. Zheng et al. [25] detect record boundary by the annotation evidence available from labeled data, however, such labeling evidence is not generally available in a large scale.

Zhu et al. [27] and Yang et al. [20], though employing different statistical models, both resort to the same idea: integrating data record extraction, attribute extraction and labeling into one phase, so that the data record extraction can benefit from the availability of the semantics of attribute labeling. This can be seen as utilizing more domain constraints in our view. However, their work follows essentially a semi-automatic approach. To fulfill the need of the statistical model, their work requires heavy domain-specific knowledge, including various pre-defined (attribute) labeling spaces [27], manually-crafted rich features for identifying attribute content and capturing the sophisticated relationships between attributes and records [20]. The performance will be highly dependent on the quality of the labeled data as well as the richness of the feature set designed. In the case of updating the attribute sets or applying such models from one domain (e.g. forums) to other types of data (e.g. blogs and reviews), we probably would have to re-design the feature set and re-train the model. Therefore we refer to such work as highly domain dependent. In our case, as will be shown in Sec. 6, by incorporating only a small amount of general domain constraints in a fully automatic fashion, our work is accurate across a wide variety of domains/applications.

To summarize the relation with previous work, our approach (1) utilizes the domain constraints not incorporated by previous fully-automatic approaches [13], which results in a more robust performance; (2) resorts to only a bit of general domain knowledge compared to the highly domain dependent approach [20, 27] (that relies on rich domain specific knowledge and features), which makes it applicable to varieties of domains/applications.

3. MDR AND OUR PROPOSAL

In this section, we introduce our proposal for extracting web data records containing UGC by reviewing MDR [13] and discussing its limitations.

3.1 MDR and its Limitations

To extract data records, MDR is based on two basic observations [13]:

1. A group of similar objects, which forms a data region, is usually presented in a *contiguous* region and formatted using *similar* HTML tags.
2. Every record in a data region is formed by *the same number of adjacent child sub-trees* under *the same parent* node.

Based on these assumptions, MDR employs a fairly straightforward greedy approach to identify data records, by enumerating the start offset and length of every combination of sub-trees under each parent and checking if the pair of two adjacent sub-tree combinations is similar in HTML tags or structure against a predefined threshold.

The main issue that we find regarding MDR is the similarity measure, which judges whether two sub-trees are similar or not. In the original work [13], MDR compares only the tag strings of the roots of two trees. We refer to this kind of similarity measure as **Top Level (TL)**. We can expect that, due to the lack of inspection on lower level structure, this similarity measure would bring errors. In Liu's later work [12], they mention the use of the tree similarity by comparing two trees, as illustrated in Fig. 3. We refer to it as **Tree Similarity (TS)**. However, as discussed in Sec. 1, two data records of the same type may still be quite dissimilar when they contain a large portion of UGC. Sec. 4.1 will formally define similarity measures.

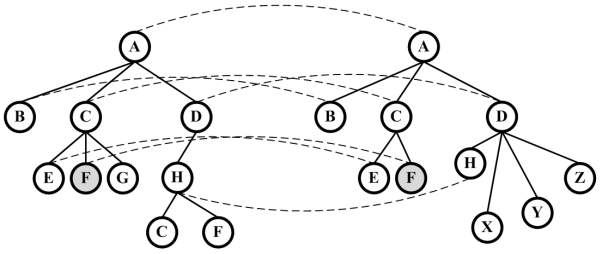

Figure 3: Alignment of two trees

Another issue, as denoted in [15], is that MDR works in a greedy manner: any mistakes made in earlier steps, by noise, will affect the mining procedure and thus corrupt the final result. In addition, MDR cannot identify non-consecutive data records.

3.2 Our Proposal

Our main proposal for enhancing MDR is to refine the similarity test between two (data) records to alleviate the effects caused by the irregular UGC part. More specifically, we propose better similarity measures by focusing the calculation of similarity on appropriate tree fragments, after conducting the tree matching procedure on the two trees. The key challenge is how to select the appropriate tree fragments that the similarity calculation should focus on.

As discussed in Sec. 1, ideally the best way is to differentiate the regular template part from the irregular UGC

part and compute the similarity on the template part to serve as the similarity measure. We refer to this measure as **Template Tree (TT)**. For example in Fig. 3, we treat all sub-trees beneath Node D as UGC, and use the remaining trees to measure similarity.

However, it is not easy to differentiate UGC from the template due to the flexible and complicated usage of HTML tags. We have to resort to finding an approximate tree fragment that works as close to the template part as possible. One simple way to do the approximation might be computing similarity using the top N levels of nodes only. We refer to it as **Top N Level (TnL)**. However, TnL may still work poorly due to a lack of deeper structural information.

Inspired by domain dependent work [20, 27], we find that domain constraints help detect the appropriate tree fragments for the similarity test. For example, for the application of extracting posts from web forums, a good constraint will be the *post-date* (publication date of a post) because it is a part of structured data occurring in every post and also can be easily identified (Fig. 2). We will discuss the domain constraints in more detail and show how to use those to design better similarity measures in Sec. 4.2.

Domain constraints also provide strong anchor point information for data record detection. For example, for extracting forum posts, each post (data record) must contain exactly one sub-tree containing *post-date*. Inspired by this intuition, we propose a novel algorithm called MiBAT (Mining data records Based on Anchor Trees) that detects the position and boundary of each record by utilizing such anchor points. The proposed MiBAT does not have the greedy deficiency of MDR and it can also identify and extract non-consecutive data records in a natural and uniform manner. We will introduce more details about MiBAT in Sec. 5.

4. SIMILARITY MEASURES

In this section we will formulate similarity measures and propose measures guided by domain constraints.

4.1 Formulation

The structure of a web page can be described by the DOM tree [1]. A tree is an ordered pair $T = (V, E)$ comprising a set V of nodes together with a set E of edges. Hereafter, for ease of presentation, we denote the node sets of two trees T_1 and T_2 to be V_1 and V_2 respectively.

Two trees can be compared and matched by finding an optimal *mapping* between them, based on which the similarity score can be computed. The concept of tree mapping [18] is formally defined as follows:

Definition 1. A *mapping* M from tree T_1 to T_2 is a set of ordered pairs of nodes (u, v), $u \in V_1, v \in V_2$, satisfying the following conditions that for all $(u_1, v_1), (u_2, v_2) \in M$:

- $u_1 = u_2$ iff $v_1 = v_2$,
- u_1 is on the left of u_2 iff v_1 is on the left of v_2, and
- u_1 is an ancestor of u_2 iff v_1 is an ancestor of v_2.

In our work we adopt a restricted version of tree mapping called *top-down* mapping [16] defined below, which has been successfully applied to many web related applications [8]:

Definition 2. A mapping M from tree T_1 to T_2 is *top-down* if it satisfies the condition: for all non-root nodes $u \in V_1$, $v \in V_2$, if $(u, v) \in M$, then $(parent(u), parent(v)) \in M$, where $parent(v)$ is the parent of v.

The tree matching shown in Fig. 3 is actually a top-down mapping. As in [12, 23], we will use a top-down mapping algorithm given by [21] in $O(n^2)$ time.

Generally, the similarity of two trees is computed as [12]:

Definition 3. Given the tree mapping result M, the similarity score of two trees T_1 and T_2 is computed as

$$TreeSim(T_1, T_2) = \frac{|M|}{(|V_1| + |V_2|)/2} \quad (1)$$

where $|M|$ is the number of match pairs in M, which also equals to the number of matched nodes in T_1 or in T_2.

We can see that Equ. (1) takes all tree nodes into account. As discussed before, given two trees and the mapping result, multiple similarity scores, referred to as *similarity measures*, can be calculated based on different sub-sets of tree nodes. We refer to sub-sets of tree nodes as *tree fragments*[1].

Definition 4. A *tree fragment selection function* is mapping $f : \mathcal{V} \rightarrow \mathcal{V}$, which maps the node set V of T to a sub-set of nodes $f(V)$, i.e. $f(V) \subseteq V$.

Definition 5. Given two trees T_1, T_2 and their mapping result M, a *similarity measure* associated with a tree fragment selection function f is computed as

$$TreeSim_f(T_1, T_2) = \frac{|M \cap (f(V_1) \times f(V_2))|}{(|f(V_1)| + |f(V_2)|)/2} \quad (2)$$

where $f(V_1) \times f(V_2) = \{(u, v) | u \in f(V_1), v \in f(V_2)\}$, which filters out irrelevant match pairs from M with respect to the relevant tree fragments $f(V_1)$ and $f(V_2)$.

Each similarity measure is uniquely determined by the associated tree fragment selection function f. Particularly, the four similarity measures discussed in Sec. 3 are formally defined as:

Top Level (TL) is by $f_{\mathrm{TL}}(V) = \{root(T)\}$, where $root(T)$ is the root of T.

Tree Similarity (TS) is by $f_{\mathrm{TS}}(V) = V$.

Template Tree (TT) is by $f_{\mathrm{TT}}(V) = template(T)$, where $template(T)$ is the node set of the template part of T.

Top N Level (TnL) is by $f_{\mathrm{TnL}}(V) = \{v | v \in V, depth(v) \leq n\}$ where $depth(v)$ is the depth of v. Note that TL is a special case of TnL when $n = 1$.

4.2 Similarity Measures by Domain Constraints

We next show how to use domain constraints to derive similarity measures. Particularly, for the ease of presentation, we will use *post-date* (publication date of a post) as the example constraint although other constraints can be utilized in a similar manner.

A good domain constraint like *post-date* usually has two properties: (1) being easily identified (e.g., a method based on regular expression matching is sufficient in our experiments) and (2) always occurring as key structured data in every data record even across different types of media (e.g., *post-date* can be found in reviews and blogs as well).

[1]Here a tree fragment is actually a set of nodes, instead of a sub-tree with sets of nodes and edges. We can see that, given the tree mapping result, a similarity score can be computed using the sets of nodes only, as shown in Equ. (1).

We refer to the lowest tree node (usually a leaf node) that contains *post-date* as the *pivot*. Our intuition is that the pivot usually belongs to the structured part of a data record, based on which we can sketch (at least part of) the template part for the similarity test, and thus obtain approximation similarity measures that work close to the ideal TT.

The simplest sketch of the template will be the pivot itself, which corresponds to judging two sub-trees of records as similar if they match at the pivot. We define the corresponding similarity measure as follows:

Pivot Match (PM) is by $f_{\mathrm{PM}}(V) = \{p\}$ where p is the pivot of T.

One may find that all ancestors of a pivot, which forms a tree path from root to the pivot, should also belong to the template part and thus be matched. Actually this has already been captured in PM, due to the top-down tree mapping procedure in use which requires that parent nodes must be matched before child nodes (Sec. 4.1).

Next we try to further improve this similarity measure by enlarging the involved tree fragment. Our intuition is that, since the pivot usually belongs to the template part of the record, its sibling nodes are also likely to belong to the template part. Therefore we have the following measure:

Pivot and Siblings (PS) is by $f_{\mathrm{PS}}(V) = \{v | v \in V, parent(v) = parent(p)\}$ where p is the pivot of T, $parent(v)$ is the parent of v.

Taking the two trees T_1 and T_2 in Fig. 3 as an example, if F is the pivot, then $f_{\mathrm{PS}}(V_1) = \{E, F, G\}$, $f_{\mathrm{PS}}(V_2) = \{E, F\}$.

It is very likely that, compared to PM, PS explores a large portion of the template, and thus has more discriminative power. We will show in Sec. 6.1 that PS achieves significantly better results than PM.

We conclude this section by summarizing in Fig. 4 the generalization relationship between similarity measures discussed so far. Edge linkage means that the more general measure (the higher) can turn into the more special one (the lower) by enlarging the tree fragments in similarity computation. We can see that TT is the ideal similarity measure, but in general it is hard to obtain. Compared to the ideal measure, TS is strict while other measures are lenient.

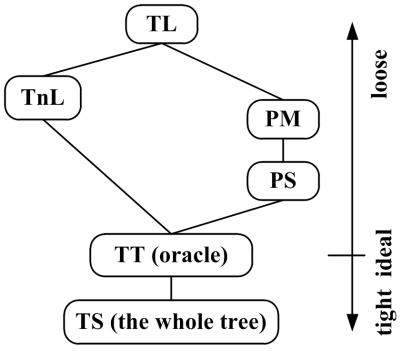

Figure 4: Relationships between similarity measures

5. MINING BASED ON ANCHOR TREES

We have introduced two domain-constraint guided similarity measures, i.e. PM and PS. In this section, we propose a data record mining algorithm using either PM or PS.

Our intuition is very simple: each record consists of one or several sub-trees, only one of which contains the pivot (Sec. 4.2). We call such sub-trees that contain pivots as *anchor* trees, since they provide anchor point information about where data records are located. We simply look for possible records around those anchor trees.

For example, in Fig. 5, having identified those anchor trees (triangle nodes in the figure), we can find and extract each data record composed by a set of adjacent sibling sub-trees around every anchor tree (gray nodes in the figure).

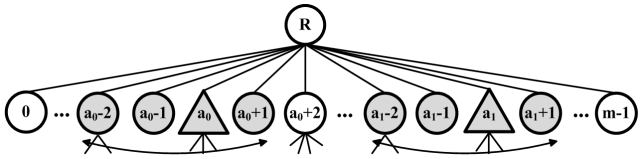

Figure 5: Mining data records based on anchor trees (triangle nodes represent anchor trees while every four consecutive gray nodes shows a data record)

5.1 Algorithm Overview

In our context, we slightly extend the basic assumptions made by MDR [13] and assume that data records have the following assumptions:

Same parent A list of data records are formed by child sub-trees under the same parent node.

Same length Each data record consists of the same number (maybe more than one) of adjacent child sub-trees. We refer to this number as the record length.

Non-contiguity The data record list does not have to be consecutive. There may be irrelevant nodes embedded in the middle (e.g. the advertisement bar in Fig. 1).

Similarity Data records must be structurally similar with each other to some extent. More specifically, by this assumption any two records must satisfy: (1) all pairs of corresponding sub-trees have the same HTML tag at root, i.e. the two sub-tree lists have the same tag sequence at the top level; (2) one pair of corresponding sub-trees, i.e. the anchor trees, must be judged as similar with respect to the domain-constraint guided similarity measure in use.

The first two assumptions are due to MDR. MDR also assumes that data record lists are consecutive, which is relaxed in our work by the third assumption. The forth assumption is due to our motivation and proposal in previous sections.

One may suspect that these assumptions may be too strong. However, based on our observations, they indeed reasonably capture the truth. For example consider the **same length** assumption. As pointed out by MDR, this assumption only requires that records have the same number of nodes at the top level; it allows records to have fairly diverse structure at lower levels. Web data records generated from a common template/schema usually share a similar structure but may differ in specific data values or fields. Data fields are usually shown at lower levels while the structure skeleton is primarily determined by upper level nodes. For this reason it is very likely that data records, though perhaps having missing data values in deeper structure, still have the same

number of nodes at the top level. Therefore, this assumption not only enables the mining procedure but also captures a wide variety of regularly structured web data objects.

Given these assumptions, our algorithm can be stated as follows: along a traversal on the DOM tree, for each parent node we (1) find the anchor trees and then (2) determine the record boundary, i.e. start offset and length, and extract data records around each anchor tree. In Fig. 5, having identified the anchor trees (denoted by gray triangle nodes), we know that data records start from the position -2 relative to each anchor tree and have a length of 4.

Alg. 1 shows the overall algorithm of MiBAT. Note that under a parent node there may be multiple sets of data objects, each corresponding to a different set of anchor trees. MiBAT will find all sets of anchor trees (Line 4), and process each for record extraction (Lines 5∼7). Line 6 determines the record boundary and returns data records.

Algorithm 1 Mining based on anchor trees

MiBAT(T)
1: $\Omega \leftarrow \{\}$
2: **for** parent tree node p in T
3: $t_1 \dots t_n \leftarrow$ the child sub-tree list of p
4: $\Delta \leftarrow$ FindAnchorTrees($t_1 \dots t_n$)
5: **for** anchor tree list $(a_1 \dots a_m)$ in Δ
6: $R \leftarrow$ DetermineBoundary($t_1 \dots t_n, a_1 \dots a_m$)
7: $\Omega \leftarrow \Omega \cup \{R\}$ ▷ a list of data records found
8: **return** Ω ▷ return all record lists

In the next two sub-sections we will discuss how to find anchor trees from a child sub-tree list of a parent and how to determine record boundary, respectively.

5.2 Finding Anchor Trees

Anchor trees are a set of sibling sub-trees under the same parent, no matter if they are consecutive or non-consecutive, as long as they all satisfy the domain-constraint guided similarity measure, i.e. matching at or around the pivot.

As discussed in Sec. 4.2, the text of a pivot can be easily identified by a pivot format classifier. For example, a date-time format classifier can be employed in the case of using *post-date* as the domain constraint. However, not all the nodes containing text in pivot format are real pivots. For example, in forum posts, UGC may also contain strings in date format. We refer to the nodes containing text in pivot format as candidate pivots. From the definition we know that candidate pivots are real pivots only if they can match between all data records. In the example shown in Fig. 6, given gray triangles being anchor trees, we can see that only Node A is the real pivot.

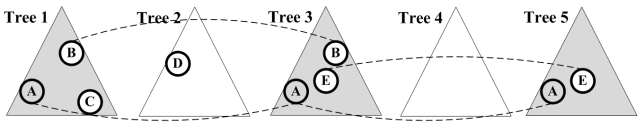

Figure 6: Finding anchor trees (each triangle denotes a tree; each circle denotes a candidate pivot; the gray node is the real pivot; gray triangles are anchor trees)

We employ the algorithm shown in Alg. 2 to find anchor trees and also to obtain the real pivot set on-the-fly as a byproduct. The basic idea is very simple: once we obtain candidate pivots, we can use them to identify new anchor trees (Lines 9∼11); once a new anchor tree is added, we use it to update the candidate pivot set (Line 12). The *covered*[i] ($i = 1 \dots n$) is used to ensure that a tree belongs to at most one anchor tree set. It also helps avoid returning redundant sub-sets of the anchor trees. Function DomainCompare() compares two trees using one of the domain-guided similarity measures, i.e. either PM or PS. It also returns the candidate pivots matched in the tree matching.

Algorithm 2 Finding anchor trees

FindAnchorTrees($t_1 \dots t_n$)
1: $\Delta \leftarrow \{\}$
2: *covered*[i] $\leftarrow 0$ for $i = 1 \dots n$
3: **for** $i \leftarrow 1$ **to** n
4: **if** *covered*[i] $= 1$ **then continue**
5: $a_1 \leftarrow i$, $m \leftarrow 1$ ▷ anchor tree list with counter of m
6: $CPSet \leftarrow$ candidate pivots in t_i ▷ by classifier
7: **for** $j \leftarrow i + 1$ **to** n
8: **if** *covered*[j] $= 1$ **then continue**
9: $matchedCP \leftarrow$ DomainCompare($t_i, t_j, CPSet$)
10: **if** $matchedCP \neq \emptyset$ ▷ similarity test succeeds
11: $m \leftarrow m + 1$, $a_m \leftarrow j$ ▷ expand the list
12: $CPSet \leftarrow CPSet \cap matchedCP$ ▷ update
13: *covered*[j] $\leftarrow 1$
14: **if** $m \geq 2$ ▷ $m = 1$ means t_i is not an anchor tree
15: $\Delta \leftarrow \Delta \cup \{(a_1 \dots a_m)\}$
16: **return** Δ ▷ return all anchor tree lists

DomainCompare($t_i, t_j, CPSet$)
1: $M \leftarrow$ TreeMatching(t_i, t_j)
2: $matchedCP \leftarrow \{\}$
3: **for** u in $CPSet$ ▷ check each in $CPSet$
4: **if exists** candidate pivot v in t_j **that**
DomainSimilarity(M, t_i, u, t_j, v) $> \tau$
▷ PM or PS, using u as t_i's pivot, v as t_j's pivot
5: $matchedCP \leftarrow matchedCP \cup \{u\}$
6: **return** $matchedCP$

Using Fig. 6 as an example, we start from Tree 1 and the candidate pivot set (denoted as $CPSet$) being $\{A, B, C\}$. Tree 2 is not an anchor tree because it cannot match Tree 1 at any candidate pivot, thus not being similar by domain constraints. Tree 3 is identified as an anchor tree and $CPSet$ is updated to be $\{A, B\}$. Tree 4 is skipped since it does not contain any candidate pivot. Lastly, Tree 5 is added to the anchor tree list and $CPSet$ is finally updated to be $\{A\}$. In such a procedure we successfully find anchor trees 1, 3 and 5, and the real pivot node A.

Note that since Tree 2 also contains candidate pivots, it may belong to another anchor tree list and thus a new set of data records, together with other trees. Alg. 2 will continue this procedure from Tree 2 (Lines 3∼5), but would not find any matches and will stop.

5.3 Determining Record Boundary

When anchor trees have been identified, we are ready to determine the boundary of records, i.e. (1) the start offset of a record relative to each anchor tree, and (2) the record length. Once the record boundary is obtained, it will be straightforward to extract data records.

In this section we first use the three cases in Fig. 7 as an example to show the basic idea, then present the general algorithm and its pseudo-code in Alg. 3.

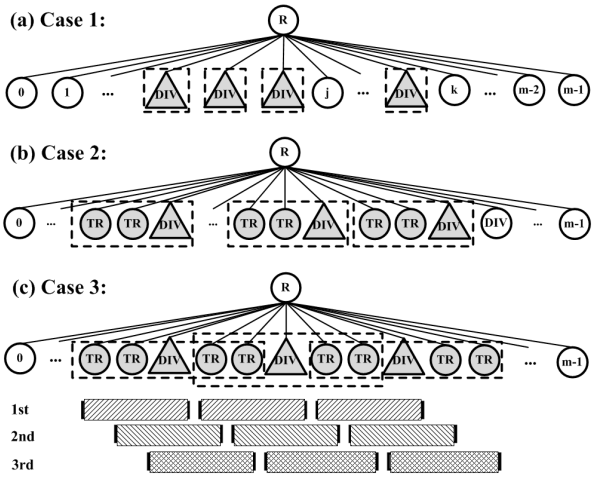

Figure 7: Boundary determination (triangles denote anchor trees; dashed boxes denote expansions)

Case 1. Two or more anchor trees are adjacent. In this case, it is trivial to see that every single anchor tree forms a data record, as shown in Fig. 7(a).

If it is not Case 1, then the minimal distance between two anchor trees is greater than 1, and each data record may consist of multiple adjacent sub-trees around the anchor tree, instead of the single anchor tree only. The intuition here is that: *each data record should consist of as many consistent sub-trees around each anchor tree as possible, as long as the* ***similarity*** *assumption in Sec. 5.1 is satisfied.* Take Fig. 7(b) as an example. Compared to treating each data record as consisting of a single anchor tree (the triangular `DIV`) only, it would make more sense to treat a record as consisting of a sub-tree triple of `TR TR DIV` (in dashed boxes), because all such triples naturally form a data record set in the figure. Therefore record boundary should result in the largest sections of adjacent sub-trees that consistently form a list of data records around all anchor trees.

Starting from the anchor tree, we try to expand the data record in two directions before (1) we encounter the left or right boundary of the child list or another anchor tree (Line 2 and 9 in Alg. 3), or (2) the newly expanded tree violates the **similarity** assumption in Sec. 5.1 (Line 4 and 10 in Alg. 3). We say that all valid expanded sub-trees around each anchor tree form an *expansion* (illustrated in dashed boxes in Fig. 7). Then we face either Case 2 or Case 3 below.

Case 2. The length of each expansion is less than or equal to the minimal distance between two anchor trees. Take Fig. 7(b) as an example, where the expansion is `TR TR DIV`. In such a case, no two expansion regions around different anchor trees overlap with each other and it is natural that the sub-trees within each expansion form a data record.

Case 3. If the length of each expansion is greater than the minimal distance between two anchor trees, there must be two expansion regions overlapping on a few sub-trees. Take Fig. 7(c) as an example, where the expansion around each anchor tree contains exactly five sub-trees of `TR TR DIV TR TR` and two consecutive expansion regions overlap on two sub-trees of `TR TR`. In this case, the largest record length will be determined by the minimal distance of two anchor trees, i.e. 3 in Fig. 7(c), and there will be ambiguity about the start offset of the data record. For example in Fig. 7(c) there are three possible start offsets, i.e. -2, -1 and 0 respectively. In this case, we just need to find the start offset leading to the maximum similarity among all possible choices. Note that setting a similarity threshold would not resolve the ambiguity.

All three cases can be integrated into one procedure, as illustrated in Alg. 3. The input is the child sub-tree list $t_1 \dots t_n$ and indices of anchor trees $a_1 \dots a_m$. As discussed above, we first obtain the minimal distance between two anchor trees (*anchorGap*, Line 1), and the expansion (Lines 2∼14). For Case 1 the length of expansion *expanLen* is 1. Then the record length k will be the smaller one between *anchorGap* and *expanLen* (Line 16). Lines 17∼20 enumerate all possible start offsets and select the best record list that has the largest similarity score. Given the length of k, for each start offset of x, the record list is $R^{(x)} = R_1^{(x)} \dots R_m^{(x)}$(Line 19), where $R_i^{(x)} = t_{a_i+x} \dots t_{a_i+x+k-1}$ $(1 \le i \le m)$ is the sub-tree list of the ith record (Line 18). The similarity score of $R^{(x)}$ is defined as

$$Score(R^{(x)}) = \sum_{1<i\le m} \sum_{0\le j<k} TreeSim(t_{a_i+x+j}, t_{a_{i-1}+x+j}) \tag{3}$$

that is, the sum of the tree similarity scores of the corresponding sub-trees between every two consecutive data records.

Note that for Cases 1 and 2, where the record length is equal to the expansion length, i.e. $k = expanLen$, there will be only one offset candidate $x = 0$ (Line 17), which is consistent with what we have discussed above. Therefore all three cases are integrated into Alg. 3.

5.4 Main Region Selection

Both MDR and MiBAT output every data record list identified. Every data record list is called a *data region* [13]. In our scenario we have to determine which region contains the list of concerned records, i.e. forum posts. We refer to it as the *main region*.

Ideally the main region can be identified by building a classifier using rich domain features (e.g. size or position of the region). In our work, we instead use two simple heuristics:

1. The list of posts must have *post-date* in each record due to the domain constraints in the context.
2. The list of posts should occupy a majority of the page and each record should be somewhat similar to each other. These two factors can be combined into one score as the sum of matched node number in the tree matching between every two consecutive records.

Therefore, to select the main region we first filter out irrelevant data regions by Heuristic 1 and then select one that has the largest score defined by Heuristic 2. We found in experiments that this selection procedure is highly effective.

Note that Heuristic 1 is built-in by MiBAT, due to the similarity test by domain constraints. But for MDR, we have to explicitly apply Heuristic 1 by filtering out regions that do not contain a matched node containing text in date-time format.

Algorithm 3 Determining record boundary

DETERMINEBOUNDARY($t_1 \dots t_n, a_1 \dots a_m$)

```
anchorGap ← min_{1<i≤m}(a_i − a_{i−1})
left ← 0                                   ▷ left boundary of expansion
for k ← 1 to min{anchorGap, a_1} − 1
    if exists 1 ≤ i, j ≤ m that DIFFTAG(t_{a_i−k}, t_{a_j−k})
        break
    else
        left ← left − 1
right ← 0                                  ▷ right boundary of expansion
for k ← 1 to min{anchorGap − 1, n − a_m}
    if exists 1 ≤ i, j ≤ m that DIFFTAG(t_{a_i+k}, t_{a_j+k})
        break
    else
        right ← right + 1
expanLen ← right − left + 1                ▷ length of expansion
R* = [ ]                                   ▷ intialize the result
k ← min{anchorGap, expanLen}               ▷ length of record
for x ← k − expanLen to 0                  ▷ enumerate start offset
    R_i^(x) ← t_{a_i+x} ... t_{a_i+x+k−1} for i = 1 ... m
    R^(x) ← R_1^(x) ... R_m^(x)            ▷ records of the current offset
    R* = arg max{Score(R*), Score(R^(x))}  ▷ Equ. (3)
return R*                                  ▷ return the best record list
```

6. EXPERIMENTS

In this section, we evaluate our approach empirically. Particularly, we (1) test the scalability of our approach by applying it to forum pages from various types of forum sites (Sec. 6.1) and review/blog pages (Sec. 6.2) and (2) demonstrate its effectiveness by comparing it with one state-of-the-art semi-automatic approach [20] (Sec. 6.3).

6.1 Forum Posts Extraction

6.1.1 Dataset

In this experiment, we are to evaluate the performance of our method when applying it to forum post extraction. There are many forum software types on the web for forum content generation and management. To verify the scalability of our approach, we require that the dataset covers as many software types (and therefore forum sites) as possible.

In order to construct the dataset, we first built a list of forum software by checking a few well-known web sites that list, review, or compare forum software packages, i.e. Big Boards, Forum Matrix, Forum Software Reviews, Hot Scripts, and Wikipedia. Then, for each software package within the list, we tried to find at least one sample site by checking the official site of the software or issuing queries like "forums powered by XYZ" to search engines like Bing and Google. For some software packages that have multiple versions, e.g. vBulletin, we kept one sample site for each version but treated them as belonging to the same software package, due to a lack of clear measurements of the structural difference between multiple versions. We also added a few forum sites powered by customized software. Finally, we obtained a dataset consisting of 307 forum sites that cover 171 known forum software packages and 48 customized types.

The forum software list shows a great variety, covering most popular software packages (like vBulletin, phpBB, Invision, etc) and various programming languages (like PHP, ASP.NET, etc). For the 307 forum sites, we didn't restrict their topics. Thus, they cover many categories of topics (including travel, computer, science and mathematics, photography and also general topics).

From each forum site that we collected, we randomly crawled a few thread pages. After removing those non-thread pages and thread pages containing zero or only one post, we kept 1,200 thread pages (452 pages were discarded during the process). After manually labeling the dataset (by identifying the golden posts as demonstrated in Fig. 1), we used 200 of them as the development set and the remaining 1,000 pages as the test set. We are also careful to make sure that the test set contains at least one and at most six thread pages for each forum site. In the test set, the total number of posts is 11,139. 64.8% of pages contain no more than 10 posts while 91% of pages contain no more than 20 posts. 12.3% of pages contain only 2 posts while 0.4% of pages contain more than 100 posts.

6.1.2 Baseline Methods

As our approach is fully-automatic (without the use of labeled data), we implemented MDR [12, 13] as one of the baseline methods. Besides, as discussed in Sec. 3.1, MDR can only extract a consecutive record list. It may miss those posts from the non-consecutive points, either to the end, thus missing one or two records, or in the middle, thus resulting in two separated record lists. As a workaround, we proposed a two-pass (2Pass) method to help MDR extract as many data records as possible from a non-consecutive data region. It is quite straightforward: in the first pass we employ the basic MDR to explore a set of consecutive records, then in the second pass we check those non-consecutive siblings and put them back to the existing region if they also meet the comparison criterion, regardless of whether they are adjacent to the existing records or not. We used the 2Pass method as another baseline and denoted it as MDR2Pass. To make the baselines stronger, we also extended both baselines with all types of similarity measures.

We utilized the development set to tune the similarity threshold for all similarity measures. For TnL, we use $n = 3$ and denote it as T3L.

6.1.3 Evaluation Measures

We evaluate the result on both post level and page level. On post level we use the standard precision and recall as evaluation metrics. On page level, we examine how many pages a method can handle perfectly without extracting any wrong, or missing any golden, posts. We also examine how many pages a method extracts/misses zero or ≤ 2 wrong/golden posts.

A post record is regarded as correctly extracted if it contains exactly the same set of DOM tree nodes with the golden record after removing blank tag nodes, e.g. `<hr />`.

6.1.4 Results

Tables 1 and 2 give the results on both post and page level. We refer to MiBAT using Pivot and Siblings (PS) as the best method. We conducted the significant test (sign test) and the result is that the best method outperforms other methods significantly (p-value=0.01).

For Table 1, we have the following observations:

- Both proposed similarity measures, i.e. PM and PS,

are effective, since they improve MDR and MDR2Pass significantly (precision and recall: from around 70% to over 85%).

- MiBAT is very effective using both proposed similarity measures, being able to achieve more than 96% in both precision and recall.
- To compare the similarity measures, PS outperforms PM in both precision and recall. The performance increase is due to more accurate constraints, and thus more accurate anchor tree identification.

Table 1: Post extraction (post level, prec./rec.)

	MDR	MDR2Pass	MiBAT
TL	55.8% / 70.0%	47.0% / 71.1%	-/-
TS	80.8% / 73.0%	60.2% / 80.5%	-/-
T3L	76.1% / 77.1%	55.2% / 79.8%	-/-
PM	90.0% / 85.4%	90.4% / 87.5%	97.5% / 96.2%
PS	90.4% / 86.2%	91.2% / 88.2%	98.9% / 97.3%

On page level, as shown in Table 2, besides similar observations as those with post level, we discuss new results in three aspects:

- Percentage of pages that are processed perfectly, i.e. neither missing any golden posts nor extracting any wrong posts. We can see that by using stronger similarity measures and better mining algorithms, we increase this percentage from around 40% to over 90%.
- Precision related percentage. Our best method won't extract any wrong posts in 98.5% of pages, and it won't extract more than two wrong posts in 99% of pages.
- Recall related percentage. Our best method won't miss any golden posts in 91.7% of pages, and it won't miss more than two golden posts in 96.8% of pages.

Table 2: Post extraction (page level)

Model	Perfect	Extract wrong posts		Miss golden posts	
		0	≤ 2	0	≤ 2
MDR					
TL	37.1%	60.7%	80.1%	58.0%	75.9%
TS	46.0%	85.7%	93.2%	46.8%	77.6%
T3L	53.9%	77.6%	90.3%	61.9%	82.8%
PM	66.5%	84.4%	90.9%	66.8%	86.6%
PS	66.5%	84.8%	91.5%	66.8%	87.1%
MDR2Pass					
TL	39.4%	50.5%	71.4%	66.9%	76.7%
TS	54.0%	85.0%	92.7%	55.0%	87.3%
T3L	57.7%	70.8%	85.0%	70.8%	86.0%
PM	75.1%	84.7%	90.9%	75.4%	88.8%
PS	74.9%	85.2%	91.6%	75.2%	89.2%
MiBAT					
PM	90.7%	97.5%	98.0%	90.7%	96.0%
PS	91.7%	98.5%	99.0%	91.7%	96.8%

6.1.5 *Error analysis*

Since the best method perfectly processed 91.7% of pages without errors, we checked all remaining 83 pages and examined the reasons. We found that two major errors are due to (1) forums having the first post under a different parent (22 pages, 26.5%) or in a different structure (28 pages, 33.7%) from the remainings posts, and (2) pivot format classifier error (10 pages, 12%). The former is beyond the scope of our method while the latter can be alleviated by using a more accurate date-time classifier. In addition, we also notice that 6 page errors (7.2%) are due to main region selection error, which can be solved using more domain features.

6.2 Blog and Review Comments Extraction

In this section, we are to test the performance of our method and two baselines when applied to other domains, i.e. extracting comments from blogs and reviews pages.

We constructed two datasets. One consists of randomly sampled blog pages that potentially contain comments from the index repository of Bing. The other consists of randomly crawled pages from a list of manually collected 15 well-known UGC sites, which contain user reviews and comments on products, news, books and pictures, e.g., Amazon, Flickr, Epions.com. After manually labeling each page and discarding those with zero or only one comment, the final datasets consist of 221 pages from 163 blog sites and 246 pages from the 15 UGC sites.

In this experiment, we didn't tune the threshold, but used that obtained in Sec. 6.1 directly. However, we introduce one more heuristic in the main region selection: we filter out those data regions in which the HTML text is a purely date-time string. This heuristic is very useful when processing blog pages since there is usually a long list of links pointing to older blog archives with text in date-time format.

Table 3 shows the experimental results, where B1 refers to baseline MDR2Pass+TS, B2 refers to MDR2Pass+PS and M is our method MiBAT+PS. For blog comments, our method significantly outperforms the baselines. For review comments, though having similar a recall to baselines (near 81%), our method achieves a higher precision (94.1%).

Note that the result in Table 3 is not as good as that in Table 1 and Table 2, especially for the recall of review comment extraction (81.8%). One main reason is that in 29 pages (out of 246) of the dataset of review pages, not all comment records are located under the same parents, which voilates the **same parent** assumption (Sec. 5.1). Due to space limitations, we leave the investigation of this issue to the journal version of this paper.

Table 3: Comment extraction

	Precision	Recall	Perfect	Extract ≤ 2 wrong	Miss ≤ 2 golden
Blog comments					
B1	52.5%	76.6%	45.7%	74.2%	73.8%
B2	58.5%	79.9%	65.2%	77.8%	81.9%
M	95.8%	91.1%	78.3%	96.4%	89.6%
Review comments					
B1	89.3%	80.0%	63.7%	85.8%	79.6%
B2	91.8%	81.4%	72.2%	84.1%	81.2%
M	94.1%	81.8%	73.9%	87.3%	82.4%

6.3 Comparison with a Semi-Automatic Method

In this section, we are to show the effectiveness of our approach by comparing it with one state-of-the-art semi-automatic approach (Yang et al., [20]).

We use the same dataset as in Yang et al. Among the 20 forum sites in this dataset, there turn out to be 4 known forum software packages and 5 customized types (since 12 out of 20 forum sites use vBulletin). Our dataset stated in Sec. 6.1 contains all the 4 known forum software packages and overlaps on 8 forum sites with their dataset. In addition, we do not exclude those single-post pages from their test corpus, although handling single-post pages is not within the scope of this paper.

Yang et al.'s work extracts posts from web forum sites by employing a supervised statistical model equipped with both page-level and site-level knowledge as features. However, our method relies only on the single post page and does not

require any labeled data. Again in this experiment we did not tune the threshold parameter, but used the parameter obtained in Sec. 6.1 directly. Pivot and Siblings (PS) is used as the similarity measure.

Table 4 compares the results on post level. The first three rows show the performance of Yang et al.'s method incorporating features of different levels, i.e. single page features (denoted as SP), site-level features (SL) and multiple page features (MP). The next two rows report MiBAT's result, corresponding to taking or not taking single-post pages into account in evaluation respectively.

Table 4: Comparison with Yang et al. (post level)

Model		Precision	Recall
Yang et al. [20]	SP	93.2%	65.6%
	SP+SL	93.7%	69.5%
	SP+SL+MP	99.6%	94.1%
MiBAT	≥ 2 posts	99.3%	99.2%
	all pages	95.6%	98.1%
MiBAT+MP	all pages	99.7%	98.5%

We have the following observations, which confirm that our method is very effective:

- Given the same condition, i.e. extracting from a single page only, our model (Row 2 in our method) significantly outperforms Yang et al.'s work (Row 1 in their method) with respect to both precision and recall.
- Even given the optimal condition for each method, our method still achieves a competitive performance with theirs (Row 2 in our method v.s. Row 3 in their method). The relatively lower precision in our method is due to those single-post pages, for which MiBAT returns an irrelevant record list as the main region for it cannot extract single posts.

So far we have focused on the settings that MiBAT works in the online mode, that is, it takes one single page as input, without any additional site-level information. However, it is fairly straightforward to generate a tree-structured template (wrapper) from the extracted post records, as demonstrated by existing research [23], and this template can be used to extract records from pages of the same type on the site. In this way, single-post pages can be also handled naturally.

Therefore, we conducted a simple experiment to test MiBAT's performance when it works in the batch mode. For each site, we use randomly sampled 20 thread pages to learn the template, by (1) employing MiBAT to process each page, (2) selecting the page for which the extracted data region has the largest score defined by Heuristic 2 in Sec. 5.4 and (3) generating the template from this page as in [23]. No human efforts are involved in the process. The last row (MiBAT+MP) in Table 4 shows the performance of this method, which achieves competitive precision and much better recall compared with Yang et al.

7. CONCLUSIONS AND FUTURE WORK

In this paper we studied the problem of automatically extracting from a single page a list of web data records that contain UGC. We proposed to utilize domain constraints to design better similarity measures as well as to acquire anchor point information for data record extraction, resulting in our novel mining algorithm MiBAT. Extensive experiments prove the effectiveness of our work.

Future work follows two paths: (1) continue to improve our approach by relaxing the assumptions and applying to more domains, and (2) move to extract data attributes [23].

8. ACKNOWLEDGEMENT

We would like to thank Jiang-Ming Yang for sharing the dataset in the referred paper, and Matt Callcut for his proofreading of this paper.

9. REFERENCES

[1] http://www.w3.org/dom/.
[2] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *SIGMOD*, 2003.
[3] D. Buttler, L. Liu, and C. Pu. A fully automated object extraction system for the world wide web. In *ICDCS*, 2001.
[4] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan. A survey of web information extraction systems. *IEEE Trans. on Knowl. and Data Eng.*, 18(10):1411–1428, 2006.
[5] C.-H. Chang and S.-C. Lui. IEPAD: information extraction based on pattern discovery. In *WWW*, 2001.
[6] S.-L. Chuang and J. Y.-j. Hsu. Tree-structured template generation for web pages. In *WI*, 2004.
[7] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, 2001.
[8] D. de Castro Reis, P. B. Golgher, A. S. da Silva, and A. H. F. Laender. Automatic web news extraction using tree edit distance. In *WWW*, 2004.
[9] D. W. Embley, Y. Jiang, and Y. k. Ng. Record-boundary discovery in web documents. In *SIGMOD*, 1999.
[10] M. Hu and B. Liu. Mining and summarizing customer reviews. In *SIGKDD*, 2004.
[11] A. H. F. Laender, B. A. Ribeiro-neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Record*, 31:84–93, 2002.
[12] B. Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Springer Series, 346–367, 2007.
[13] B. Liu, R. Grossman, and Y. Zhai. Mining data records in web pages. In *SIGKDD*, 2003.
[14] B. Liu and Y. Zhai. NET - a system for extracting web data from flat and nested data records. In *WISE*, 2005.
[15] G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, and L. E. Moser. Extracting data records from the web using tag path clustering. In *WWW*, 2009.
[16] S. M. Selkow. The tree-to-tree editing problem. *Inf. Process. Lett.*, 6(6):184–186, 1977.
[17] K. Simon and G. Lausen. ViPER: augmenting automatic information extraction with visual perceptions. In *CIKM*, 2005.
[18] K.-C. Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.
[19] J. Wang and F. H. Lochovsky. Data extraction and label assignment for web databases. In *WWW*, 2003.
[20] J.-M. Yang, R. Cai, Y. Wang, J. Zhu, L. Zhang, and W.-Y. Ma. Incorporating site-level knowledge to extract structured data from web forums. In *WWW*, 2009.
[21] W. Yang. Identifying syntactic differences between two programs. *Softw. Pract. Exper.*, 21(7):739–755, 1991.
[22] W.-Y. Yang, Y. Cao, and C.-Y. Lin. A structural support vector method for extracting contexts and answers of questions from online forums. In *EMNLP*, 2009.
[23] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *WWW*, 2005.
[24] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. T. Yu. Fully automatic wrapper generation for search engines. In *WWW*, 2005.
[25] S. Zheng, R. Song, J.-R. Wen, and C. L. Giles. Efficient record-level wrapper induction. In *CIKM*, 2009.
[26] S. Zheng, R. Song, J.-R. Wen, and D. Wu. Joint optimization of wrapper generation and template detection. In *SIGKDD*, 2007.
[27] J. Zhu, Z. Nie, J.-R. Wen, B. Zhang, and W.-Y. Ma. Simultaneous record detection and attribute labeling in web data extraction. In *SIGKDD*, 2006.