Automatic Extraction of Top-k Lists from the Web

Zhixian Zhang*1, Zheng Wang*2, Haixun Wang+3, Kenny Q. Zhu*4

*Shanghai Jiao Tong University
Shanghai, China
1zhangzhixian1989@gmail.com 2wangzhdemail@gmail.com 4kzhu@cs.sjtu.edu.cn

+Microsoft Research Asia
Beijing, China
3Haixun.Wang@microsoft.com

Abstract— **List data is an important source of structured data on the web. This paper is concerned with "top-k list" pages, which are web pages that describe a list of k instances of a particular topic or concept. Examples include "the 10 tallest persons in the world" and "the 50 hits of 2010 you don't want to miss". We present an efficient algorithm that extracts the target lists with high accuracy even when the input pages contain other non-target lists of the same size or errors. The extraction of such lists can help enrich existing knowledge bases about general concepts, or act as a proprocessing step to produce facts for a fact answering engine.**

I. Introduction

The world wide web is by far the largest source of information today. Much of that information contains structured data such as tables and lists which are very valuable for knowledge discoverage and data mining. This structured data is valuable not only because of the relational values it contains, but also because it is relatively easier to unlock information from data with some regular patterns than free text which makes up most of the web content. However, when encoded in HTML, structured data becomes *semi-structured*. And because HTML is designed for rendering in a browser, different HTML code segments can give the same visual effect at least to the human eye. As a result, HTML coding is much less stringent than XML, and inconsistencies and errors are abundant in HTML documents. All these pose significant challenges in the extraction of structured data from the web [1].

In this demo, we focus on list data in web pages. In particular, we are interested in extracting from a kind of web pages which present a list of k instances of a topic or a concept. Examples of such topic include "20 Most Influential Scientists Alive Today", "Ten Hollywood Classics You Shouldn't Miss", "50 Tallest Persons in the World". Figure 1.(a) shows a snapshot of one such web page [2].

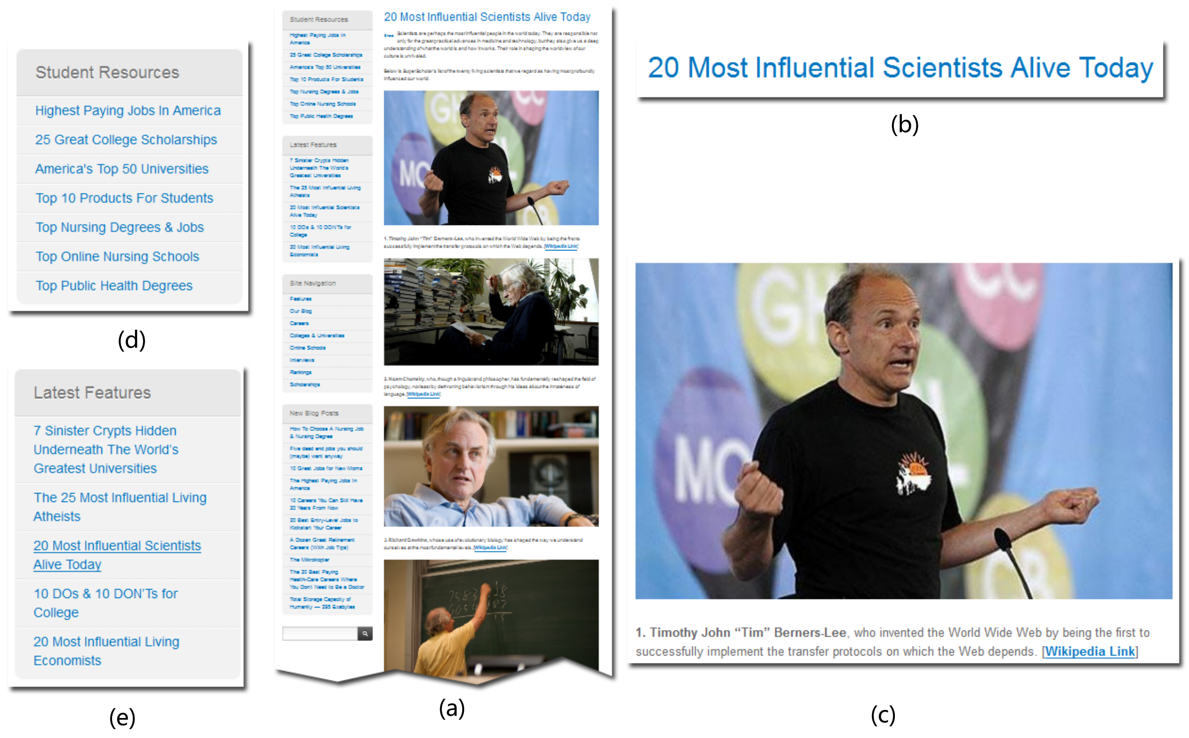
Informally, our problem is: given a web page with a title that contains a integer number k (Figure 1.(b)), check whether the page contains a list of k items as its *main content*, and if it does, extract these k items. We call such lists *top-k lists* and pages that contain the entirety of a top-k list *top-k pages*. There are also lists that span multiple pages but are connected by hyperlinked "Next" button, but these are not considered in this work. A typical scenario we consider, like the one in Figure 1, is that each list item is more than an instance of the topic in the title, but instead contain additional information such as a textual description and images, like in Figure 1.(c). Our objective is to extract the actual instance whenever possible, but in the worst case, produce list items that at least *contain* the wanted instances.

The work described in this work is an important step in our bigger effort of building an effective fact answer engine [3]. With such an engine, we can answer queries such as "Who are the 10 tallest persons in the world", or "What are 50 best-selling books in 2010" directly, instead of referring the users to a set of ranked pages like all search engines do today. Because of the special relation between the page title and the main list contained in the page, the semantics of the list items are more specific and hence it's much easier for us to integrate the extracted lists into a general knowledge base that empowers the fact answer engine.

There were many previous attempts to extract lists or tables from the web. None of them targets the top-k list extraction that is studied in this work. In fact, most of the methods are based on either very specific list-related tags [4], [5] such as `<ul>`, `<li>` and `<table>` or the similarity between DOM trees [6], [7] and completely ignore the visual aspect of HTML documents. These approaches are likely to be brittle because of the dynamic and inconsistent nature of web pages. More recently several groups have attempted to leverage visual information in HTML in information extraction. Most notably, Ventex [8] and HyLiEn [9] are designed to correlate the rendered visual model or features with the corresponding DOM structure and achieved remarkable improvements in performance. However, these techniques indiscriminatingly extract *all* elements of *all* lists or tables from a web page, therefore the objective is different from that of this work which is to extract *one* specific list from a page while purging all other lists (e.g. (d) and (e) in Figure 1) as noise. The latter poses different challenges such as distinguishing ambiguous list boundaries and identifying noise and unwanted lists.

The main approach of this work goes along the line of analyzing similar tag paths in the DOM tree with the help of visual features to identify the main list in a page. The key contributions of this demo are:

- We defined a novel top-k list extract problem which is useful in knowledge discovery and fact answering;
- We designed an unsupervised general-purpose algorithm along with a number of key optimizations that is capable



Student Resources
Highest Paying Jobs In America
25 Great College Scholarships
America's Top 50 Universities
Top 10 Products For Students
Top Nursing Degrees & Jobs
Top Online Nursing Schools
Top Public Health Degrees

Latest Features
7 Sinister Crypts Hidden Underneath The World's Greatest Universities
The 25 Most Influential Living Atheists
20 Most Influential Scientists Alive Today
10 DOs & 10 DON'Ts for College
20 Most Influential Living Economists

20 Most Influential Scientists Alive Today

1. Timothy John "Tim" Berners-Lee, who invented the World Wide Web by being the first to successfully implement the transfer protocols on which the Web depends. [Wikipedia Link]

Fig. 1. Snapshot of a typical *top-k page* and its page segments

of extracting top-k lists from any web pages (in Section II);

- Our evaluation shows that our algorithm scales with the data size and achieves significantly better accuracy than competing methods (in Section III).

II. Our Approach

We first make some preliminary definitions. An HTML *Document Object Model* (DOM) is a tree structure whose nodes are HTML elements identifiable by HTML tags and associated attributes. A *tag path* is a path from the root to an arbitrary node in the DOM tree. A top-k list item, or *list item* in short, is a segment of HTML page which represents a unique item in the top-k list of a top-k page. A list item can contain multiple *item components* such as a title (which is often the name of an entity), some descriptive text or an image.

Our basic algorithm runs in four steps. First, we compute the tag path for every node in the DOM tree of the input page. Second, we group nodes with an identical tag path into one *equivalence class*, and we select those equivalence classes which have exactly k members as our candidate classes. In effect, an equivalence class represents a list of item components. Third, for each of these candidate classes, we employ an *GrowUp* operation to merge some of the equivalence classes together, which essentially form a number of candidate lists. Now the item components that belong to the same list item are grouped together. Finally, we rank the candidate list by their importance to the page, and return the top ranking list as the result. If there doesn't exist any candidate list, then this input page is not a top-k page.

The basic algorithm solves most but not all top-k list extraction problems. To improve the result quality and the performance, we further introduce four optimization heuristics.

1) **Visual Area.** In the basic algorithm, the relative importance of a candidate list to the whole page is calculated as number of text characters in the list against the total number of text characters. This doesn't work when the list contains large non-text areas such as white space or images, or the list uses large fonts. As an enhancement, we try to estimate the total *visual area* of the candidate list versus the total area of the page. This can done by the calculating the combination of image sizes, font sizes and potential white spaces.
2) **Interleaving Lists.** For aesthetic reasons, a top-l list may have an "interleaving pattern", where list items have alternate visual styles such as background colors or fonts. In the basic algorithm, this kind of list may be treated as two lists of size $k/2$ and removed mistakingly. We include a special heuristic to detect such interleaving patterns and reconstruct the whole list from two smaller candidate lists.
3) $k+1$ **Problem.** Some top-k pages have a top-k list with an additional header or footer that looks almost exactly the same and has the same tag path as the real list items. The basic algorithm cannot distinguish between the real items and this "fake" item. There is a similar $k-1$ problem where the first or the last item of the real

top-k list has a slightly different style and is excluded from the equivalence class. We pay special attention to equivalence classes of size $k+1$ and $k-1$ and try to identify these boundary cases by comparing the subtree of the items or analyzing the text content.

4) **Item Title Extraction.** From our observation of many top-k pages from different domain, we generalized a number of rules for identifying the title entities in a top-k list. For example, titles are often placed before other item components, and often occupy a single line in the rendered display with stylized fonts.

III. Experimental Results

A. Experimental Setup

In order to evaluate our extraction system automatically, we first design the benchmark dataset. Test cases in benchmark are collected from the whole Internet. Our goal is to extract list items from the whole Internet and not to extract list items from particular websites.

In collection phrase, we use a paradigm-superlative Number Object- which indicates the web page contains a list of N items in the same topic. We first google an instance of the paradigm such as Best 10 Players, then we collect the search result using a heuristics. By using different instances of the paradigm, we collect 100 web pages from different websites and domains.

After collection, we label the cached web pages. For each web page, we label the first item of data record list which we want to extract.

Finally we generate two benchmarks $B1$ and $B2$ which have 92 pages and 96 pages respectively. And there are no overlapping between them.

In order to compare with MDR[6] system, we generate another benchmark $B3$ from the testbed[10] which is used in Gengxin Miao's paper[7]. The testbed is designed for information extraction from the deep Web, which has 253 Web pages from 51 web sites randomly with search forms. We randomly choose 142 pages from 29 web sites to form $B3$. Since our system needs to know the list item number k of the page, we manually assign the list number k for each page in the benchmark. Also we label the first item of data record list of each page.

Our experiments were carried out on a Core (TM) 2 computer with a 2GHz Duo CPU and 3GB of RAM.

B. Effect of Optimizations

Given a page p for evaluation, we have the extracted list items S_l. If one of the items in S_l contains our manually added label, then we consider this page are extracted correctly.

Here we use precision and recall as the evaluation metrics for our system's accuracy test. The precision(P) and recall(R) are defined straightforwardly as follows:

$$P = \frac{|\text{correctly extracted pages}|}{|\text{pages with returned list items}|} \quad (1)$$

$$R = \frac{|\text{correctly extracted pages}|}{|\text{number of testcases in benchmark}|} \quad (2)$$

TABLE I

Average Accuracy Comparison

Version	P(%)	R(%)
Basic	87.3	59.5
Optimized	**90.3**	**70.9**

TABLE II

Average Execution Time of Different Stages

Version	Total	I/O	Parsing	Algorithm	Optimization
Basic(ms)	402.9	141.5	115.8	145.6	0
Optimized(ms)	374.7	141.6	116.2	**116.4**	**0.5**

We compare the performance of two versions of our system. the basic and the optimized. The result of the experiment is shown in Figure 2 and the average accuracy statistics are shown in Table I.

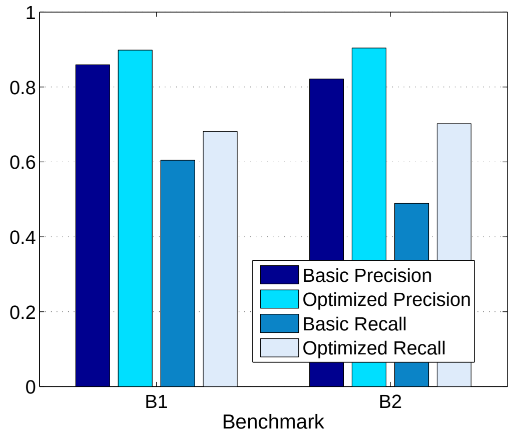


Fig. 2. Precision and Recall for Each Benchmark

The result indicates that the optimizations can remarkably improve the performance, especially of the recall score. Thus, the accuracy our final system is satisfying and sufficient for solving our problem.

We also record the average running time for each step to test efficiency, which is listed in Table II. The data shows the high efficiency of our system, the algorithm and optimization only take up about one third of total running time. Besides, the optimized system has a shorter running time than the basic one, which indicates those optimizations won't harm the efficiency of the whole system.

Also, we conduct a scaling test of file size, which is shown in Figure 3. The test set are 120 pages with proper size from $B1$ and $B2$. The result shows a certain positive correlation between file size and running time.

C. Peer Comparison

When comparing with MDR, we would like to introduce the evaluation metrics used in Miao's work[7]. The meaning of precision and recall is a little different which is defined as P^* and R^*. The ground truth are all the list items of a Web site, true positives are the list items correctly extracted

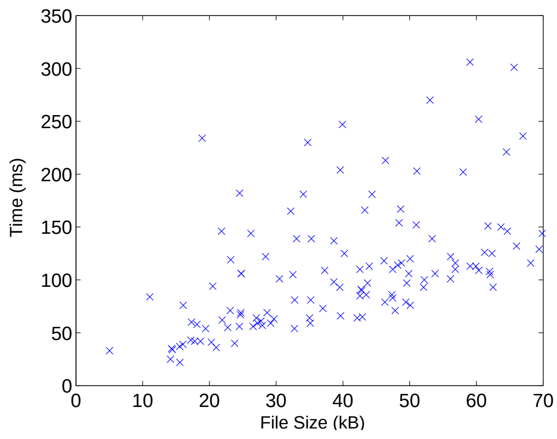


Fig. 3. Scaling Test of File Size

TABLE III
PEER COMPARISON

Version	P^* (%)	R^* (%)	Avg Time
Optimized	**90.4**	**73.2**	144.4 ms
MDR	54.4	56.4	1-2 s

and false positives are the list items that incorrectly included in the same list with true positives. For the case that both false and true positives equal to 0, we define P^* to be zero.

$$P^* = \frac{|\text{true positives}|}{|\text{true positives}| + |\text{false positives}|} \tag{3}$$

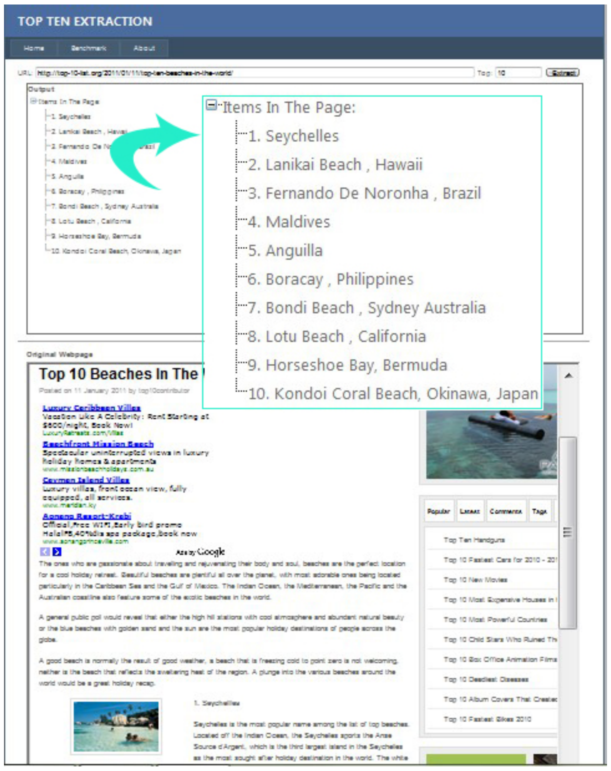
$$R^* = \frac{|\text{true possitive}|}{|\text{ground truth}|} \tag{4}$$

The result of the comparison is listed in Table III, which shows that our system has an obviously higher precision and recall than MDR system in B3. Furthermore, efficiency of the Optimized System is better since the average time is much shorter than MDR. The experimental result of MDR is very close to that in Miao's work[7] ($P^* = 59.8\%$ and $R^* = 61.8\%$), which infers the validity of the experiment.

We also tries to test MDR with $B1$ and $B2$. However, for more than half of the pages we test, it takes more than 1 minute to give result. Even if we omit those pages, the precision P and recall R are still very low($< 15\%$), which can not form any sensible comparison. As a result, MDR system is not suitable for solving our problem.

IV. DEMONSTRATION SETUP

At the heart of our demonstration is a web-based Top-k list extraction user interface. A screenshot of this interface is shown in Figure 4. On this site, user can pick any of the pre-installed benchmark data set, and execute our extraction algorithm on the set. User can examine the original page and the extracted output side-by-side. They can also tune the behavior of the system by enabling or disabling various optimization and check the outcome. The demo page also provides option to run other algorithms such as Web tables, MDR or Ventex on the same data set. In addition to pre-installed data, user can optionally type in any URL and a number k. The system will retrieve the page in real time and attempt to extract the top-k list from it, or declare failure if it doesn't exist.



Items In The Page:
1. Seychelles
2. Lanikai Beach , Hawaii
3. Fernando De Noronha , Brazil
4. Maldives
5. Anguilla
6. Boracay , Philippines
7. Bondi Beach , Sydney Australia
8. Lotu Beach , California
9. Horseshoe Bay, Bermuda
10. Kondoi Coral Beach, Okinawa, Japan

Top 10 Beaches In The

Fig. 4. The Top-k Extraction Web GUI

REFERENCES

[1] T. Weninger, F. Fumarola, R. Barber, J. Han, and D. Malerba, "Unexpected results in automatic list extraction on the web," *SIGKDD Explorations*, vol. 12, no. 2, pp. 26–30, 2010.
[2] "20 most influential scientists alive today," http://www.superscholar.org/features/20-most-influential-scientists-alive-today/.
[3] X. Yin, W. Tan, and C. Liu, "Facto: a fact lookup engine based on web tables," in *WWW*, 2011, pp. 507–516.
[4] "Google sets," http://labs.google.com/sets.
[5] M. J. Cafarella, E. Wu, A. Halevy, Y. Zhang, and D. Z. Wang, "Webtables: Exploring the power of tables on the web," in *VLDB*, 2008.
[6] B. Liu, R. L. Grossman, and Y. Zhai, "Mining data records in web pages," in *KDD*, 2003, pp. 601–606.
[7] 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, pp. 981–990.
[8] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak, "Towards domain-independent information extraction from web tables," in *WWW*. ACM Press, 2007, pp. 71–80.
[9] F. Fumarola, T. Weninger, R. Barber, D. Malerba, and J. Han, "Extracting general lists from web documents: A hybrid approach," in *IEA/AIE (1)*, 2011, pp. 285–294.
[10] Y. Yamada, N. Craswell, T. Nakatoh, and S.Hirokawa, "Testbed for information extraction from deep web," in *WWW*, 2005.