Automatic Extraction of Clickable Structured Web Contents for Name Entity Queries

Xiaoxin Yin, Wenzhao Tan, Xiao Li, Yi-Chin Tu
Microsoft Research
One Microsoft Way
Redmond, WA 98052
{xyin, wentan, xiaol, yichint}@microsoft.com

ABSTRACT

Today the major web search engines answer queries by showing ten result snippets, which need to be inspected by users for identifying relevant results. In this paper we investigate how to extract structured information from the web, in order to directly answer queries by showing the contents being searched for. We treat users' search trails (i.e., post-search browsing behaviors) as implicit labels on the relevance between web contents and user queries. Based on such labels we use information extraction approach to build wrappers and extract structured information. An important observation is that many web sites contain pages for name entities of certain categories (e.g., AOL Music contains a page for each musician), and these pages have the same format. This makes it possible to build wrappers from a small amount of implicit labels, and use them to extract structured information from many web pages for different name entities. We propose STRUCLICK, a fully automated system for extracting structured information for queries containing name entities of certain categories. It can identify important web sites from web search logs, build wrappers from users' search trails, filter out bad wrappers built from random user clicks, and combine structured information from different web sites for each query. Comparing with existing approaches on information extraction, STRUCLICK can assign semantics to extracted data without any human labeling or supervision. We perform comprehensive experiments, which show STRUCLICK achieves high accuracy and good scalability.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *search process.*

General Terms

Algorithms, Measurement, Experimentation.

Keywords

Web search, Information extraction.

1. INTRODUCTION

Although web search engines have evolved much in the past decade, the paradigm of "ten result snippets" barely changes over time. After submitting a query, the user needs to read each snippet to decide whether the corresponding web page has the contents he is searching for, and clicks on the link to see the page.

If a search engine can provide "direct answers" for a significant portion of user queries, it can save a large amount of time spent by each user in reading snippets. Take the query {Britney Spears songs}[1] as an example. For this query Google does not show songs directly (except two results from video vertical). There are actually many web pages providing perfect contents for this query. For example, http://www.last.fm/music/Britney+Spears/ and http://www.rhapsody.com/britney-spears contain lists of songs by Britney Spears sorted by popularity, in rather structured layouts. If we can show a list of songs as results for this query, or convert the snippet of each search result into a list of songs, the user can directly see the information he is searching for, and click on some links to fulfill his need, e.g., listening to the songs.

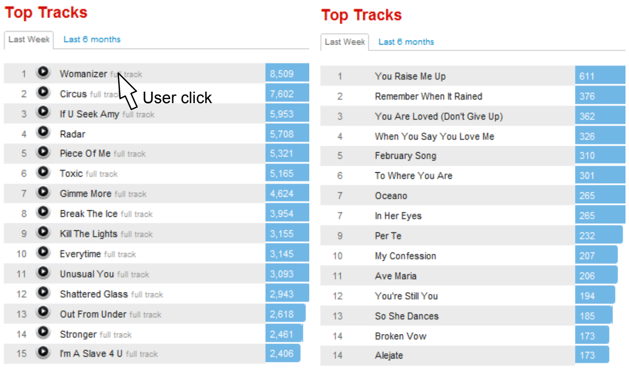
It would *not* be very difficult to provide such direct answers if a search engine could understand the semantics of web page contents. However, in lack of an effective approach for understanding web contents, the "ten result snippets" still dominate the search result pages. Some search engines show direct answers for a very small portion of queries, such as query {rav4} on Bing.com. However, these direct answers are usually based on backend relational databases containing well structured data, instead of information on the web.

Many approaches have been proposed for extracting structured information from the web. One popular category of approaches is wrapper induction [12][18], which builds a wrapper for web pages of a certain format based on manually labeled examples. Another popular category is automatic template generation [2][7][8][15][21], which converts an HTML page into a more structured format such as XML. Unfortunately neither of them can be directly used to supply structured data to search engines. Wrapper induction cannot scale up to the whole web because manual labeling is needed for each format of pages on each web site. Automatic template generation approaches can only convert all contents on web pages into structured format, but cannot provide semantics for the data to allow search on them.

In this paper we try to bridge the gap between web search queries and structured information on web pages. We propose an approach for finding and extracting structured information from the web that match with queries. Our approach is based on the search trails of users, i.e., a sequence of URLs a user clicks after submitting a query and clicking a search result. Because these post-search clicks are usually for fulfilling the original query intent, we use the contents being clicked (e.g., the clicked URLs and their anchor texts) as implicit labels from users, and use such labels to build wrappers and extract more data to answer queries. For example, a user may search for {Britney Spears songs}, click on a result URL http://www.last.fm/music/Britney+Spears/ (as shown in Figure 1(a)), and on that page click another URL http://www.last.fm/music/Britney+Spears/_/Womanizer, which links to a song "Womanizer". Then we can know the last clicked

WWW 2010, April 26-30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

[1] We use "{*x*}" to represent a web search query *x*.



Top Tracks

Last Week Last 6 months

1	Womanizer full track	8,509
2	Circus full track	7,602
3	If U Seek Amy full track	5,953
4	Radar	5,708
5	Piece Of Me full track	5,321
6	Toxic full track	5,165
7	Gimme More full track	4,624
8	Break The Ice full track	3,954
9	Kill The Lights full track	3,155
10	Everytime full track	3,145
11	Unusual You full track	3,093
12	Shattered Glass full track	2,943
13	Out From Under full track	2,618
14	Stronger full track	2,461
15	I'm A Slave 4 U full track	2,406

(a)

Top Tracks

Last Week Last 6 months

1	You Raise Me Up	611
2	Remember When It Rained	376
3	You Are Loved (Don't Give Up)	362
4	When You Say You Love Me	326
5	February Song	310
6	To Where You Are	301
7	Oceano	265
7	In Her Eyes	265
9	Per Te	232
10	My Confession	207
11	Ave Maria	206
12	You're Still You	194
13	So She Dances	185
14	Broken Vow	173
14	Alejate	173

(b)

Figure 1: (a) Part of music page of Britney Spears on www.last.fm, (b) That of Josh Groban

URL and its anchor text "Womanizer" are likely to be a piece of relevant answer for the original query. We can also extract other songs on the same page as pieces of answers for that query.

A web site containing structured web pages usually has pages in uniform format for many name entities of the same category. For example, www.last.fm has a page for each of many musicians, like the pages in Figure 1 (a) and (b). If we have a list of musicians, and have seen different queries like {[musician] songs} with clicks on URLs like http://www.last.fm/music/*/, we can infer that each such web page contains songs of a musician, which can be extracted to answer corresponding queries.

We present the STRUCLICK system in this paper. In general, it takes many categories of name entities (e.g., musicians, actors, cities, national parks), and finds web sites providing structured web pages for each category of name entities. Based on user search trails of queries containing name entities, it extracts structured information from the web pages, and uses them to answer user queries directly. STRUCLICK is a very powerful system as it can build a wrapper from a small number of user clicks, and apply it to all web pages of the same format to extract information. It is a fully automated system, as it does not require any manual labeling or supervision, and can generate structured information for different generic and popular search intents for a category of entities[2] (e.g., songs of musicians or attractions of cities).

To the best of our knowledge, this is the first study on extracting structured information using web search logs. Because intents of user queries are best captured through web search logs, we believe logs are a most necessary input for answering queries with structured data. In this first study we confine our scope within queries containing name entities, and contents on web pages that are clickable, i.e., associated with hyperlinks. The first constraint does not limit the significance of our work as it is reported 71% of queries contain name entities [11]. It will be our future work to remove the second constraint.

There are three major challenges for accomplishing the above task. The first challenge is how to identify sets of web pages with uniform format, when it is impossible to inspect the content of every page because of the huge data amount. We propose an approach for finding URLs with common patterns. According to our experiments, URLs with same pattern correspond to pages with uniform formats most of time. The second challenge is that the amount of user clicked contents is usually small, based on which we need to build HTML wrappers to extract large amount of structured information. An approach based on paths of HTML tags [16] is used, which can build wrappers and extract information efficiently. The third challenge is how to distinguish relevant data from irrelevant data. As shown by our experiments, users often click on URLs not relevant to their original queries, which leads to significant amount of noise in the extracted data. Moreover, there is no information for the relevance of vast majority of extracted items without user clicks. Based on the observation that items extracted by a wrapper are usually all relevant or all irrelevant, we propose a graph-regularization based approach to identify the relevant items and good wrappers.

We perform comprehensive experiments to study the accuracy and scalability of STRUCLICK, and use human judgments via Amazon Mechanical Turk [1] to validate the results. It is shown that STRUCLICK can extract a large amount of structured information from a small number of user clicks, filter out the significant amount of noise caused by noises in users' search trails, and finally produce highly relevant structured information (with accuracy 90%–99% for different categories of queries). It is also shown that STRUCLICK is highly scalable, which makes it an ideal system for information extraction from the web.

The rest of this paper is organized as follows. We discuss related work in Section 2. Section 3 describes the architecture and algorithms of STRUCLICK system. We present empirical study in Section 4, and conclude this study in Section 5.

2. RELATED WORK

Extracting structured information from web pages has been studied for more than a decade. Early work is focused on wrapper induction, which learns extraction rules from manually labeled examples [13]. Such systems include WIEN [12], Stalker [18]. These approaches are semi-automatic as they require labeled examples for each set of web pages of a certain format from a web site. Such labeling procedure is not scalable as there are a very large number of such web sites, with new sites emerging and existing sites changing formats from time to time.

In the last decade there are many studies on automatic extraction of structured information from web pages. IEPAD [7] and MDR [15] focus on extracting repeated patterns from a single web page. [16] utilizes "path of tags" to identify each type of objects in HTML DOM trees. The approaches in [2][8][21] create patterns or templates from many web pages of same format, in order to extract data from them. RoadRunner [8] uses a web page as the initial template, and keeps modifying the template when comparing it with more pages. EXALG [2] is based on the assumption that a set of tokens co-occurring with same frequency in different pages are likely to form a template. DEPTA [21] uses partial tree alignment on HTML DOM trees to extract data.

Although the above approaches can automatically extract structured data from web pages of the same format, they cannot provide any semantics to each data field being extracted, which means they simply organize the data in HTML pages into a structured format (e.g., XML). To get semantics of data, one has to label each data field for each format of pages, which is unscalable for web scale tasks. It is also difficult to select the web sites to extract data from, for both semi-automatic and automatic information extraction methods. In contrast, we combine the searching and post-searching browsing behaviors of users to identify the semantics of data fields, which enables extracting data suitable for answering queries.

[2] It is very easy to get entities in different categories from web sites like Wikipedia and freebase.

The problem of automatically annotating data on the web has been studied extensively for creating the Semantic Web [3]. SemTag [9] uses an existing knowledge base, and learns distributions of keywords to label a huge number of web pages. In [17] an approach is proposed to extract information with a given ontology by learning from user supplied examples. These approaches both require user provided training data, and are based on spatial locality of web page layout, i.e., semantic tags can be found from surrounding contents on HTML pages. These features may limit their accuracy because different web sites may have very different styles, and semantic tags may not exist in surrounding contents. Our approach is very different from them, as we use users' search trails for training, and build wrappers using information extraction approaches instead of relying on spatial layout of web pages.

Automated information extraction from web pages of arbitrary formats has also been well studied. In [5] Banko et al. study how to automatically extract information using linguistic parser from the web. In [6] Cafarella et al. extract information from tables and lists on the web to answer queries. Although these approaches can be applied to any web pages, they rely on linguistics or results from search engines to assign semantics to extracted data for answering queries, which limits their accuracy. [6] reports that a relevant table can be returned in top-2 results in 47% of cases. Our approach is very different from above approaches as we perform information extraction from web pages of uniform format. Based on users' search trails, and the consistent semantics of data extracted from uniformly formatted pages, we achieve very high accuracy (≥ 97% for top results).

Paşca [19] has done many studies on automatically finding entities of different categories, and important attributes for each category. These are very important inputs for our system, as our goal is to find important data items for each category of entities.

3. STRUCLICK SYSTEM

3.1 System Overview

In this section we provide an overview of the STRUCLICK system. Three inputs are needed for this system. The first input is a reasonably comprehensive set of HTML pages on the web, which can be retrieved from index of Bing.com. The second input is the search trails of users, i.e., the clicks made by users after querying a major search engine (Google, Yahoo!, or Bing), which can be found in the browsing logs of consenting users of Internet Explorer 8. The third input is name entities of different categories. The titles of articles within each category or list of Wikipedia or freebase can be used as a category of entities. We can also get such data from different web sites like IMDB or Amazon, or use automatic approach [19] to collect them.

We focus on web search queries containing name entities of each category (e.g., musicians), and possibly a word or phrase indicating generic and popular intent for that category of entities (e.g., songs of musicians). We call a word (or phrase) that co-appears with many entities of a category in user queries as an *intent word* for that category. Table 1 shows the intent words with most clicks in search trails for four categories[3]. Many web sites provide certain aspects of information for a category of entities, and our goal is to extract information from clickable contents of web pages, which can answer queries involving each category of entities and each popular intent word. Although we cannot find structured information for every generic intent of every category of entities, we can handle many of the important intents such as movies, songs, lyrics, concerts, coupons, hotels, restaurants, etc.

Table 1: Top intent words for four categories of entities

Actors	***Musicians***	***Cities***	***National parks***
pictures	lyrics	craiglist	lodging
movies	songs	times	map
songs	pictures	hotels	pictures
wallpaper	live	university	camping
thriller	2009	airport	hotels

As shown in Figure 2, the STRUCLICK system contains three major components: The URL Pattern Summarizer, the Information Extractor, and the Authority Analyzer. The URL Pattern Summarizer takes different categories of name entities as input, and finds queries consisted of an entity in some category and an intent word. Then it analyzes the clicked result URLs for these queries to find sets of URLs sharing the same pattern, which correspond to web pages of uniform format. For example, www.last.fm has a page for each musician with URLs like http://www.last.fm/music/*/, and such pages are often clicked for queries like {[musician] songs}.

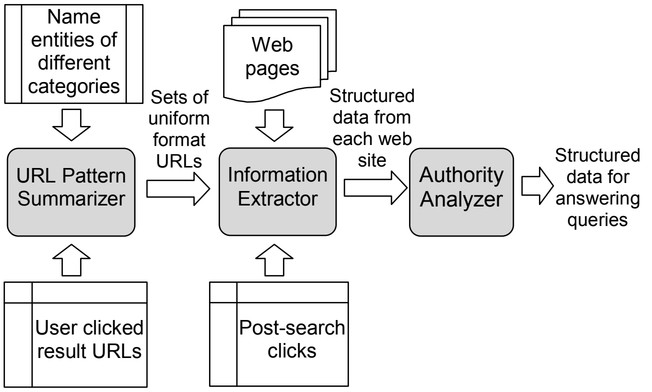

Name entities of different categories
Web pages
Sets of uniform format URLs
Structured data from each web site
URL Pattern Summarizer
Information Extractor
Authority Analyzer
Structured data for answering queries
User clicked result URLs
Post-search clicks

Figure 2: Overview of STRUCLICK system

The second component is Information Extractor, which takes each set of uniformly formatted web pages and analyzes the post-search clicks on them. It builds one or more wrappers for the entity names, clicked URLs and their anchor texts, and extracts such information from all web pages of the same format, no matter whether they have been clicked or not.

These extracted data usually contains much noise because users may click on links irrelevant to their original search intents. The Authority Analyzer takes data extracted from different web sites, and infers the relevance of data and authority of web sites using a graph-regularization approach, based on the observation that items extracted by same wrapper are usually all relevant or all irrelevant. Finally it merges all relevant data, and show to user when receiving a suitable query.

In general, STRUCLICK is a highly-automated system and relies on search and browsing logs to extract structured information for certain categories of entities. Comparing with existing systems for extracting data with semantics, STRUCLICK is almost free as it does not require any manual labeling or supervision.

3.2 URL Pattern Summarizer

Similar to most existing approaches, our information extractor can only be applied to web pages with uniform format. Therefore, the first step of STRUCLICK is to find sets of web pages of same format, from all result pages clicked by users for each category of entities and each intent word.

[3] We ignore words with redundant meanings and offensive words like "sex".

Because of the large number of pages involved, it is prohibitively expensive to compare the formats of these pages. On the other hand, we find pages of uniform format usually share a common URL pattern. For example, each page of musician on last.fm has URL like http://www.last.fm/music/*, and each page of songs of musician on Yahoo! music has URL like http://new.music.yahoo.com/*/tracks. Therefore, we try to find such URL patterns from the search result URLs clicked by users, which correspond to sets of uniform format pages most of time.

Definition 1 (**URL pattern**). A URL pattern contains a list of tokens, each being a string or a "*" (wildcard). A URL pattern matches with a URL if all strings in the pattern can be matched in the URL and each wildcard matches with a string without token separators ("/", ".", "&", "?", "="). □

When matching a URL with a pattern there are three outcomes: (1) Matched, (2) no match because they have different number of tokens or different token separators, and (3) compromised, i.e., the pattern needs to be generalized to match with the URL. Suppose pattern p_1 = http://www.imdb.com/name/nm0000*. For URL u_1 = http://www.imdb.com/name/nm2067953/, p_1 and u_1 are compromised to form pattern http://www.imdb.com/name/nm*. For URL u_2 = http://www.imdb.com/title/tt0051418/, p_1 and u_1 are compromised to generate pattern http://www.imdb.com/*/*. For URL http://www.imdb.com/video/imdb/vi3338469913/, p_1 cannot be matched with it.

Given all clicked result URLs, we hope to select a list of URL patterns, so that most URLs can match with at least one pattern. Each pattern should match many URLs, but should be as special as possible so that it does not match URLs of different formats.

First we divide all result URLs by their web domains as we do not study patterns applicable to multiple domains. For URLs from each domain, we start from an empty pattern set. We iterate through the URLs, and try to match each URL with every existing pattern. If a URL and a pattern are compromised with a new pattern generated, we include the new pattern into our pattern set. We also create a new pattern based on each URL, unless it can be matched or compromised with an existing pattern and there are already many patterns (>100).

A set of patterns are generated after iterating through all URLs in a domain, and we need to select a subset of good patterns. In general, we prefer patterns that are more specific (i.e., containing less wildcards and more characters) and cover more URLs. For each pattern p, let *coverage*(p) be the number of URLs matching with p, *wildcard*(p) be the number of wildcards in p, and *length*(p) be the number of non-wildcard characters in p (not including the web domain). The score of a pattern is defined as

$$s(p) = \left(\frac{1}{wildcard(p)+1} + \rho \cdot length(p)\right) \cdot \log_2 coverage(p), \quad (1)$$

where ρ is set to 0.03 in our system.

We select a subset of good patterns using a greedy algorithm, by selecting the pattern with highest score, removing all URLs matched with it, and selecting the next pattern. This procedure is stopped when less than 5% of all URLs are left.

It is shown by our experiments that each selected URL pattern usually matches with a large number of URLs of the same format. Therefore, in the following components we treat URLs matching with each pattern as a separate source of information.

3.3 Information Extractor

Information extractor takes the search trails of queries containing name entities, builds wrappers for the clicked links which are likely to be items of interests for the user, and extracts structured information from all web pages of the same format. We will explain these three steps in this section.

After a user submits a query, he usually clicks on a result URL, and on that page he may make some clicks. We call these clicks as *follow-up clicks*, and the clicked links are usually relevant to his original search intent. We treat each unique link from follow-up clicks as a relevant item for the original query. For example, if a user queries with {Britney Spears songs}, clicks on http://www.last.fm/music/Britney+Spears/ (Figure 1(a)), and clicks on http://www.last.fm/music/Britney+Spears/_/Womanizer (with anchor text "Womanizer"), then we consider "Womanizer" and the corresponding URL as a relevant item for {Britney Spears songs}.

For each URL pattern p found in the result URLs, let $U(p)$ be the set of all URLs matched with p. For each u in $U(p)$, we get $fc(u)$, the set of URLs clicked by follow-up clicks made on u, from the search trails of users. Our goal is to build wrappers that can extract URLs in $fc(u)$ and their anchors from the result URLs, and also extract other URLs and anchors of the same format from all URLs in $U(p)$.

Information extraction from web pages of uniform format has been extensively studied in the past decade, based on different approaches including regular expressions [8], HTML templates [2][8], and partial tree alignment [21]. A recent study [16] provides a simple and effective approach based on "tag-path", i.e., the tags on a path from the root to each node in the HTML DOM tree. Based on our observations, tag-paths are very effective in identifying a type of clicked links in a set of uniformly formatted web pages, because the layout of such links is usually unique on the pages. There is often class information on tags that distinguish different types of HTML elements, and we consider the class information specified for any tag that is closest to the leaf nodes. For example, on pages with URL pattern http://www.last.fm/music/*, each song URL has a tag-path of "<html><body><div><div><div><div><div><div> <div><table><tbody><tr><td class="subjectCell"><div><a>".

We adopt a tag-path based approach that is similar to [16] for building wrappers. When processing a URL pattern p, we build the HTML DOM tree for each page u in $U(p)$ using Html Agility Pack[4], and search for the clicked URLs on u in every element in the DOM tree. Whenever a clicked URL is found, we store the tag-path of that element as a candidate wrapper. After generating all candidate wrappers, we calculate the coverage of each of them, which is the percentage of URLs with follow-up clicks that can be extracted by this wrapper. All candidate wrappers with coverage lower than 5% are removed, and the remaining ones are used to extract data.

There are wrappers that extract apparently useless data and we remove them in this step. Some wrappers extract items containing navigational links (e.g., "Photos", "Videos") or function links (e.g., "sort by year"). We can usually remove them by calculating the *uniqueness* of the anchor texts and URLs extracted by a wrapper. The uniqueness of a set of anchor texts (or URLs) is defined as the number of unique anchor texts (or URLs) divided by the total number. Any wrapper with uniqueness less than 20% for either anchor texts or URLs will be removed. Although many irrelevant wrappers and items can be removed in this way, many of them remain in our dataset. For example, for musicians' songs we still have many wrappers extracting musicians' names, concerts, user comments, etc. Authority Analyzer will identify the relevant wrappers and combine data from them.

[4] http://www.codeplex.com/htmlagilitypack

Besides extracting the clicked items from web pages following each URL pattern, we also need to extract the entity name from each page, in order to know which entity these items belong to. This can be done using the same approach based on tag-paths, with some minor modifications. The first difference is that, since the entity name often appears with some extra text in HTML elements, we incorporate such text in our wrappers. For example, suppose we want build a candidate wrapper from the page of Britney Spears on AOL Music (http://music.aol.com/artist/britney-spears/1290171), and find "Britney Spears" appears in the page title which is "Britney Spears – AOL Music", we build a candidate wrapper of "<html><head><title>(*) – AOL Music", in which (*) is a wildcard and represents the string to be extracted. After generating all candidate wrappers, we need to select a single wrapper for extracting entity names, because each page should contain a single entity name. We say a wrapper is correct on a page if it extracts exactly one string that is the entity name. The wrapper that is correct on most pages is selected, which will be used to extract the entity name from each page.

3.4 Authority Analyzer

3.4.1 Overview

For each web search topic studied (e.g., musicians' songs), the Information Extractor creates a set of wrappers, each extracting a set of entities and a list of items for each entity from qualified web pages. Each item contains a URL and an anchor text, which can be considered as the name of the item. Because these wrappers are built from follow-up clicks, and some follow-up clicks are not relevant to the original search intent (e.g., users explore different types of information), there are usually a significant portion of wrappers extracting irrelevant items. For example, a user may search for songs of a musician, go to a result page, and then click on a link to a concert of the musician. We may build a wrapper for concerts of musicians from such follow-up clicks, which provides irrelevant items for the search topic.

It is a challenging task to select wrappers providing relevant items and remove the others. There are two important facts to take advantage of. The first one is that because each wrapper extracts items following a certain format, the items extracted by the same wrapper are usually all relevant or all irrelevant. For example, on musicians' pages on www.last.fm (with URLs like http://www.last.fm/music/*/), different wrappers extract the songs, albums, similar artists, events, user comments, or navigational links. But no wrapper extracts multiple types of items. The second fact is there are many popular and relevant items provided by many different web sites. For example, each web site listing Britney Spears' songs usually has the most popular songs like "Womanizer" and "Piece of me". If we can infer these are relevant items based on information from one web site, they can help us infer the relevance of items in many other web sites.

Some existing papers focus on the truth discovery problem [10][20], which studies how to find authoritative information sources, based on the assumption that correct information from different sources should be same or similar, while incorrect information should be different. However, this assumption does not hold in our problem, as different web sites often contain the same set of irrelevant items. For example, many different music web sites put albums, concerts, similar artists, etc. on the same page with songs of each musician. These items may be clicked by users for {[musician] songs} queries and then extracted by our wrappers. Our problem is also different as we only know the items clicked by follow-up clicks are more likely to be relevant, but know nothing about the relevance of most of the items which are never clicked. Therefore, we base our approach on the following basic principles:

(1) Two items extracted by the same wrapper are likely to be both relevant or both irrelevant.
(2) An item is likely to be relevant to a topic if it is clicked by a follow-up click of a query on that topic.

Principle (1) indicates that an item should have higher relevance if it is provided by wrappers that provide many relevant items. The reader may have concerns with principle (2) because some follow-up clicks are on irrelevant items. This is unlikely to cause problems for our algorithm, because we will optimize a function that combines all items provided by all wrappers, and a relatively small number of irrelevant clicks will not affect the accuracy. If a wrapper provides irrelevant items, even if a few of them are clicked, we can still infer this wrapper is not relevant based on principle (1).

3.4.2 Optimization

Based on above principles, we hope to assign a relevance score to each item so that items extracted by the same wrapper have similar scores, and items with more follow-up clicks have higher scores. We propose an approach based on graph regularization based learning [14][22][23], and we make significant modifications to it to fit our problem. The goal of graph regularization is to assign values to each node in a graph, so that neighbor nodes have similar values and the value of each node is similar to its pre-assigned value (which is usually a class label taking value of 0 or 1). This can fit into an optimization framework with an analytical solution [22].

One option for solving our problem is to use graph regularization by creating a graph according to our two principles. The graph contains a node for each item and an edge between each two nodes if the corresponding items are extracted by same wrapper. Each node has a label of 1 if the item is clicked and 0 otherwise. However, this method is problematic because an item not being clicked only means its relevance is unknown, instead of it being irrelevant. In fact our problem is more similar to one-class classification with a very small number of positive examples (usually <1%). Graph regularization [22][23] treats each node as equally important, and thus will assign zero or almost zero relevance to most items. Moreover, items receiving more clicks are obviously more popular for users, and we are also more confident about their relevance. Therefore, they should play more important roles in graph regularization.

We modify the approach in [22][23] by assigning different weights to different nodes in the optimization procedure. Very low weights are assigned to un-clicked items, and the weight of each clicked item is proportional to the number of clicks. We find an analytical solution to this problem, which can be computed efficiently. The details are as follows.

For a category of entities and an intent word (e.g., musicians' songs), suppose there are n wrappers w_1, …, w_n, and m items t_1, …,t_m. An item may be provided by multiple wrappers, because items are considered to be the same if they are for the same entity and share the same name. Each wrapper w provides a set of items $T(w)$, and we construct a $m \times n$ matrix W so that W_{ik} equals 1 if $t_i \in T(w_k)$ and 0 otherwise. Consider a graph G containing a node for each item. There is an edge $e_{ij} \in E(G)$ if any wrapper provides both t_i and t_j, and its weight $w(e_{ij})$ is the number of such wrappers. It can be easily proved that WW^T is the adjacency matrix of G, i.e., $w(e_{ij}) = (WW^T)_{ij}$.

We want to assign a relevance score f_i to each item t_i, so that (1) if t_i has high relevance, its neighbors in graph G should also have high relevance, and (2) if t_i receives follow-up click(s) from a query on the specific search topic, it should have high relevance. Let $\boldsymbol{f}$ be the vector ($f_1, \ldots, f_m$), and $\boldsymbol{y}$ be a vector so that y_i=1 if t_i receives follow-up click(s). A good function for optimization is provided in [22]:

$$Q(\boldsymbol{f}) = \frac{1}{2}\left(\sum_{e_{ij} \in E(G)} w(e_{ij}) \cdot \left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}}\right)^2 + \mu\|\boldsymbol{f} - \boldsymbol{y}\|^2\right) \quad (2)$$

, where $\mu > 0$, and d_i equals the sum of all elements in the i^{th} row of WW^T (i.e., total weight of all edges from the node of t_i).

$Q(\boldsymbol{f})$ contains two parts: $Q_1(\boldsymbol{f}) = \sum_{e_{ij} \in E(G)} w(e_{ij}) \cdot \left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}}\right)^2$ represents the coherence within the graph, and $Q_2(\boldsymbol{f}) = \|\boldsymbol{f} - \boldsymbol{y}\|^2$ represents the coherence with labeled examples, which are items receiving follow-up clicks in our case. It is proved in [22][23] that $Q(\boldsymbol{f})$ is minimized when

$$\boldsymbol{f}^* = (1-\alpha)(I - \alpha S)^{-1}\boldsymbol{y} \quad (3)$$

, where $\alpha = 1/(1+\mu)$, $S = D^{-1/2}WW^{\mathrm{T}}D^{-1/2}$, and D is a diagonal matrix with $D_{ii} = d_i$.

In [22][23] there is a class label on each example. While in our problem the relevance of an item is unknown if there is no follow-up click on it. This means there are only labels on some positive examples, but not on majority of them and the negative examples. Therefore, our problem is more similar to one-class classification, and $Q(\boldsymbol{f})$ cannot be used.

In general we should consider an unlabeled item to be positive if it is tightly related to positive items in the graph, and consider it to be negative if otherwise. This can be modeled by modifying the optimization function $Q(\boldsymbol{f})$. We keep the original $Q_1(\boldsymbol{f})$ and define

$$Q_2(\boldsymbol{f}) = \sum_{i=1}^{m} \lambda_i (f_i - y_i)^2 \quad (4)$$

, where λ_i is the weight of item t_i. Let $fc(t_i)$ be the number of follow-up clicks on t_i. We set $\lambda_i = 1$ if $fc(t_i) = 0$, and $\lambda_i = \gamma \cdot fc(t_i)$ if $fc(t_i) > 0$, where γ is a parameter that is much higher than 1. In this way it becomes much less important that items without follow-up clicks match with their "labels". Please note assigning different weights (λ_i) to different items is very different from assigning different labels (y_i), because f_i and y_i represent probability of an item being relevant and should be in range [0,1], and assigning very different y_i to different clicked items make it very difficult to minimize $Q(\boldsymbol{f})$.

Let Λ be a diagonal matrix that $\Lambda_{ii} = \lambda_i$. The function to be minimized becomes

$$Q(\boldsymbol{f}) = \frac{1}{2}\left(Q_1(\boldsymbol{f}) + \mu(\boldsymbol{f} - \boldsymbol{y})^{\mathrm{T}}\Lambda(\boldsymbol{f} - \boldsymbol{y})\right). \quad (5)$$

The following theorem tells us how $Q(\boldsymbol{f})$ can be minimized.

THEOREM 1. $Q(\boldsymbol{f})$ is minimized by

$$\boldsymbol{f}^* = \mu\Lambda'^{-1}\left(I - S\Lambda'^{-1}\right)^{-1}\Lambda\boldsymbol{y} \quad (6)$$

, where $\Lambda' = I + \mu\Lambda$.

PROOF. $Q(\boldsymbol{f})$ is minimized when $\frac{\partial Q}{\partial f} = 0$. It is shown in [22] that $\frac{\partial Q_1}{\partial f} = \boldsymbol{f} - S\boldsymbol{f}$. Thus we have

$$\left.\frac{\partial Q}{\partial \boldsymbol{f}}\right|_{f=f^*} = \boldsymbol{f}^* - S\boldsymbol{f}^* + \mu\Lambda\boldsymbol{f}^* - \mu\Lambda\boldsymbol{y} = 0$$

$$\Leftrightarrow (I + \mu\Lambda - S)\boldsymbol{f}^* = \mu\Lambda\boldsymbol{y}$$

With $\Lambda' = I + \mu\Lambda$, we have

$$\left(I - S\Lambda'^{-1}\right)\Lambda'\boldsymbol{f}^* = \mu\Lambda\boldsymbol{y}$$

$$\Leftrightarrow \Lambda'\boldsymbol{f}^* = \mu\left(I - S\Lambda'^{-1}\right)^{-1}\Lambda\boldsymbol{y}. \quad \square$$

3.4.3 Iterative computation procedure

Because the high dimensionality of S and the high cost of matrix inversion, it is impractical to directly compute $\boldsymbol{f}^*$ based on Equation (6). As shown in [22], if we set $\boldsymbol{f}_0 = \boldsymbol{y}$ and iteratively compute $\boldsymbol{f}_{k+1} = \alpha S\boldsymbol{f}_k + (1-\alpha)\boldsymbol{y}$ (where $\alpha = 1/(1+\mu)$), then $\lim_{k\to\infty}(\boldsymbol{f}_k) = \boldsymbol{f}^*$ as defined in Equation (3). We can convert Equation (6) into

$$\alpha\Lambda'\boldsymbol{f}^* = (1-\alpha)\left(I - \alpha\left(\tfrac{1}{\alpha}S\Lambda'^{-1}\right)\right)^{-1}\Lambda\boldsymbol{y} \quad (7)$$

Since Equation (7) is analogous to Equation (3), we can use a similar iterative procedure as shown below.

1. Let $\boldsymbol{x}_0 = \Lambda\boldsymbol{y}$.
2. Repeat:
3. $\boldsymbol{x}_{k+1} = \alpha\left(\tfrac{1}{\alpha}S\Lambda'^{-1}\right)\boldsymbol{x}_k + (1-\alpha)\Lambda\boldsymbol{y}$ (8)
4. Until $\boldsymbol{x}_k$ converges to $\boldsymbol{x}^*$

We can easily infer $\lim_{k\to\infty}(\boldsymbol{x}_k) = \alpha\Lambda'\boldsymbol{f}^*$ as in [22], and thus $\boldsymbol{f}^* = \frac{1}{\alpha}\Lambda'^{-1}\boldsymbol{x}^*$.

Please note $S = D^{-1/2}WW^{\mathrm{T}}D^{-1/2}$ is a $m \times m$ matrix, and it is very costly to compute S when m is large. Fortunately we can decompose step 3 into two steps to simplify computation as in [14]. Let $B = D^{-1/2}W$. Step 3 can be decomposed into

3(1). $\boldsymbol{z}_k = B^T\Lambda'^{-1}\boldsymbol{x}_k$ (9)

3(2). $\boldsymbol{x}_{k+1} = B\boldsymbol{z}_k + (1-\alpha)\Lambda\boldsymbol{y}$ (10)

It is much easier to compute $\boldsymbol{z}_k$, which represents the score of each wrapper in k^{th} step. The number of non-zero entries in B is equal to that in W (since D is a diagonal matrix), which is the total number of items provided by the wrappers. Therefore, each iteration can finish in linear time w.r.t. input size.

As shown in [22], the above procedure converges when the max eigen value of $\frac{1}{\alpha}S\Lambda'^{-1}$ is no greater than 1. This posts some requirements on weight λ_i, according to the following lemma.

LEMMA 1. The maximum eigen value of $\frac{1}{\alpha}S\Lambda'^{-1}$ is no greater than 1 if $\lambda_i \geq 1, i = 1, \ldots, m$.

PROOF. Matrix $\frac{1}{\alpha}S\Lambda'^{-1}$ is similar to $\frac{1}{\alpha}\Lambda'^{-1}S$ because $\Lambda'\left(\frac{1}{\alpha}\Lambda'^{-1}S\right)\Lambda'^{-1} = \frac{1}{\alpha}S\Lambda'^{-1}$. Since S is positive-definite, $\|S\boldsymbol{x}\| \leq \|\lambda_{\max}\boldsymbol{x}\|$, where $\lambda_{\max}$ is the maximum eigen value of S. (This can be easily proved through orthogonalization $S = Q^T\Lambda_S Q$.) It is already shown in [22] that any eigen value of S is no greater than 1, which means $\|S\boldsymbol{x}\| \leq \|\boldsymbol{x}\|$ for any $\boldsymbol{x}$. We can guarantee $\left\|\frac{1}{\alpha}\Lambda'^{-1}S\boldsymbol{x}\right\| \leq \|\boldsymbol{x}\|$ if no entry of $\frac{1}{\alpha}\Lambda'^{-1}$ is greater than 1. Since $\Lambda'^{-1}{}_{ii} = \frac{1}{1+\mu\lambda_i}$, this condition becomes $\frac{1}{\alpha}\frac{1}{1+\mu\lambda_i} \leq 1$, which means $\lambda_i \geq 1$. $\square$

3.4.4 Relevance of wrappers

In general, the relevance of each item can be computed using the above iterative procedure. After it converges, we have the final relevance of each item, from which we can infer the relevance of wrapper w_i as the average relevance of its items, i.e.,

$$rel(w_i) = \frac{\sum_{t_i \in T(w_i)} f_i}{|T(w_i)|}. \tag{11}$$

Because each URL pattern usually provides relevant items in a single format, we select the wrapper from each URL pattern with highest relevance, and ignore other wrappers from the same URL pattern. Because the pages from each URL pattern with significant number of user clicks usually contain some relevant information, the best wrapper from each URL pattern is usually relevant. We also remove wrappers with very relevance (< 0.001).

3.4.5 *Combining data from different web sites*

After selecting relevant wrappers and extracting data from different web sites, there is a final step of combining extracted data. This step is only needed when we want to generate a unified list of extracted items for each entity, which can be directly shown to users to answer their queries.

When combining all items for an entity *e*, we first get the list of items extracted by each wrapper for *e*. Then we order all items for each entity according to their popularities. An item appearing on multiple web domains for an entity is usually a popular item. Therefore, we simply use the number of web domains providing each item to rank the items. Whenever there is a tie, we use the sum of relevance of wrappers providing each item to resolve the ties. Please note we do not rely on relevance of wrappers to rank items, because relevance is very different from popularity.

Because different web domains often represent the same item in slightly different ways, we consider two item names to be the same if their *normalized forms* are the same. An item name is normalized by (1) removing contents in parentheses (e.g., year of a movie), (2) applying Porter's stemmer on each word, and (3) sorting the words alphabetically.

A list of items can be generated for each entity using the above procedure. In our experiments we will study whether such items are relevant to user queries.

4. EXPERIMENTS

We now present the experimental evaluation of our approaches. All experiments are run on a Windows server with dual 2.66GHz Intel quad-core CPU and 32GB main memory. All experiments are run with a single core.

4.1 Datasets

We collect users' search trails from the browsing logs of Internet Explorer 8 of consenting users from 2009/04/01 to 2009/08/31. For each web search query submitted to Google, Yahoo!, or Bing Search (or Live Search), we collect the user query, clicked URL on search result page, and the next clicked URL.

Since we will test the accuracy of the items found by STRUCLICK by human (via Amazon Mechanical Turk [1]), we need to use queries with clear intents. Each query should contain an entity of a known category, and another word or phrase indicating unambiguous intent. We choose four very different categories of queries: Movies of actors, songs of musicians, lodging of national parks, and tourism of cities. Table 2 shows the source of each type of entities (where "*" represents wildcard). For each type of entities and each corresponding intent word ("movies", "songs", "tourism", or "lodging"), we collect all queries consisted of an entity and an intent word (e.g., {Tom Hanks movies}), and find their search trails in our logs. Although we only test the accuracy of STRUCLICK on these categories of queries, there is no human labeling or intervention needed. STRUCLICK can be easily applied to queries containing all categories of entities and all important intent words for each category in fully automated ways. The processing of different categories of entities and different intent words are independent and can be done in parallel.

Table 2: Data sources for each class of entities

Class of entity	*Num. Entity*	*Wikipedia categories or Web source*
actors	19432	*_film_actors
musicians	21091	*_female_singers, *_male_singers, music_groups
cities	1000	www.tiptopglobe.com/biggest-cities-world
national parks	2337	*_national_parks, national_parks_*

When extracting data from web pages, we use the indexed pages of Bing Search on 2009/06/01 as a replicate of the Web[5].

4.2 URL Pattern Summarizer

We first show results of URL Pattern Summarizer. Table 3 shows the number of search result URLs with follow-up clicks, number of URL patterns found, and percentage of result URLs covered by any pattern. We can see majority of URLs are covered by some pattern, which shows it is possible to use information extracted from these patterns to satisfy majority of users' needs.

Table 3: Statistics of URL patterns

Category of queries	*#URLs*	*#Patterns*	*Coverage*
actor movies	70750	83	89.72%
musician songs	55057	153	83.76%
city tourism	3234	19	52.50%
national park lodging	2383	13	50.10%
Total	131424	268	85.46%

The following are the URL patterns covering most URLs for musician songs queries. It can be easily verified that each of them provides many URLs of the same format providing songs of different musicians.

http://music.aol.com/artist/*/*
http://www.songarea.com/music-codes/*.html
http://www.whosdatedwho.com/music/songs/*.htm
http://www.allthelyrics.com/lyrics/*
http://www.stlyrics.com/songs/*/*.html
http://www.mp3.com/artist/*/summary
http://new.music.yahoo.com/*/tracks
http://mog.com/music/*
http://www.mtv.com/music/artist/*/lyrics.jhtml
http://www.allbutforgottenoldies.net/*.html

The most important job of URL Pattern Summarizer is to find web pages with same format. Therefore, we test how likely a pair of URLs satisfying a URL pattern are of the same format. We randomly select 20 pairs of URLs from each category that satisfy the same URL pattern, and manually label whether each pair of URLs contain contents relevant to the query intent in the same format. The results are shown in Table 4, and we can see the URL Pattern Summarizer achieves high accuracy in finding web pages in same format. In fact even a URL pattern contains web pages in a few different formats, the Information Extractor can build different wrappers to extract information from each of them.

We also study the scalability of URL Pattern Summarizer, by applying it on URL sets with different sizes. We randomly select 1000, 2000, 4000, …, 32000 URLs from all URLs in

[5] We exclude a set of domains which have huge number of pages but usually do not provide structured information for entities: "google.*", "bing.*", "flickr.*", "myspace.*", "ask.*", "youtube.*", "wikipedia.org", "amazon.*", "ebay.*", "answer.com", "shopping.yahoo.*", "answer.yahoo.*", "video.yahoo.*".

Table 4: Accuracy of URL Pattern Summarizer

Category of queries	*#pairs*	*#correct*	*Accuracy*
actor movies	20	20	100%
musician songs	20	20	100%
city tourism	20	18	90%
national park lodging	20	19	95%
Total	80	77	96.25%

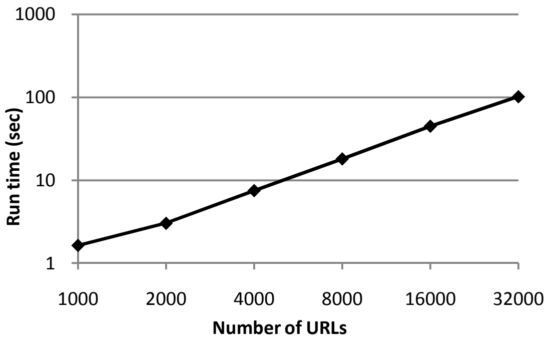


Figure 3: Scalability of URL Pattern Summarizer

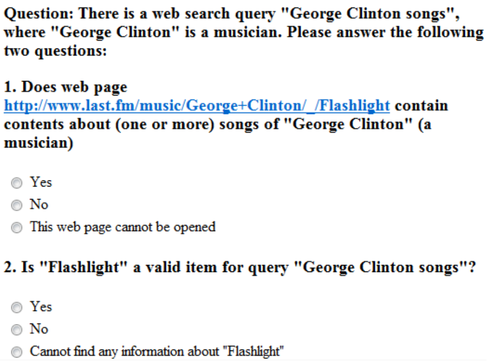
www.imdb.com with follow-up clicks. The runtime of URL Pattern Summarizer on each set of URLs is shown in Figure 3. We can see the runtime grows slightly faster than linear (growing about 128% when number of URLs doubles). Because most domains contain several or tens of URL patterns, it is usually sufficient to use thousands of URLs from each domain to infer URL patterns. In this experiment the same set of URL patterns are generated when the number of URLs is no less than 2000. URL Pattern Summarizer can finish in about 100 seconds for 32000 URLs, which is efficient enough for most domains.

4.3 Information Extractor and Authority Analyzer

4.3.1 Accuracy

Information Extractor learns wrappers from users' search trails, and extracts structured information from all web pages satisfying each URL pattern. Then Authority Analyzer infers the relevance of each wrapper, and combines items from different web domains to create an ordered list of items for each entity. Authority Analyzer has two parameters: μ for controlling the weight of coherence with labels, and γ for controlling the weight of each follow-up click. We set $\mu = 0.1$ and $\gamma = 10$. In fact we try various values and find they do not have significant influence on the output.

We test the accuracy of the final lists of items by human judgments via Amazon Mechanical Turk [1]. For each category of queries, we randomly select 50 or 100 queries (100 for actor movies and musician songs, 50 for city tourism and national park lodging), and the probability a query being selected is proportional to its frequency in the year of 2008. For each selected query, we get the top 10 items found by STRUCLICK, and ask three Mechanical Turk workers to judge whether each item is relevant for the query. Figure 4 shows an example of our questions on Mechanical Turk, which is about whether "Flashlight" (with URL http://www.last.fm/music/George+Clinton/_/Flashlight) is relevant for query {George Clinton songs}. We consider a name or URL to be relevant if majority of workers think that way[6].



Question: There is a web search query "George Clinton songs", where "George Clinton" is a musician. Please answer the following two questions:

1. Does web page http://www.last.fm/music/George+Clinton/_/Flashlight contain contents about (one or more) songs of "George Clinton" (a musician)

○ Yes
○ No
○ This web page cannot be opened

2. Is "Flashlight" a valid item for query "George Clinton songs"?

○ Yes
○ No
○ Cannot find any information about "Flashlight"

Figure 4: An example of Mechanical Turk question

In order to see if the workers are really making judgments or simply clicking on "Yes", we add about 10% of noise by replacing each item at rank 10 with a randomly selected item that is irrelevant to the query. All workers who label more than half of noise items as relevant are ignored.

We measure the accuracy of the names and URLs of top-k items for each category of queries, as shown in Table 5. Each row contains average accuracy of all items with rank through 1 to k ($k = 1$, …, 9). It can be seen that STRUCLICK achieves very high accuracy on city's tourism and musicians' songs. It is a little less accurate on actor movies, because it mixes starring with other types of contributions such as singing songs. The accuracy on national park lodging is lower because the workers often consider an item to be irrelevant if he cannot find any information about the national park on the web page of a hotel, although the hotel is actually very close to the park.

Table 5: Accuracy of STRUCLICK

	Actor movies		Musician songs		City tourism		National park lodging	
k	*Name*	*URL*	*Name*	*URL*	*Name*	*URL*	*Name*	*URL*
1	.970	.979	.978	.989	1.00	1.00	1.00	.978
2	.964	.974	.984	.995	1.00	1.00	.978	.966
3	.959	.962	.982	.989	1.00	.993	.978	.954
4	.962	.958	.981	.984	.990	.985	.960	.924
5	.967	.962	.978	.982	.992	.988	.954	.924
6	.967	.960	.977	.980	.993	.990	.950	.905
7	.969	.961	.975	.978	.991	.988	.939	.890
8	.968	.962	.973	.977	.992	.990	.910	.866
9	.962	.955	.973	.975	.991	.991	.906	.849
Noise	0	0	0	0	.044	.114	.024	.073

One advantage of STRUCLICK is its capability of extracting a large amount of structured information for each category of queries based on a very small amount of user clicks. Table 6 shows the numbers of entities and items with follow-up clicks, numbers of entities and items extracted by Information Extractor, and those of the final results by Authority Analyzer (with items combined for each entity). We can see the final results of STRUCLICK contain hundreds or thousands times more entities and items than the user clicked contents.

In order to test the capability of STRUCLICK in distinguishing relevant items and wrappers from irrelevant ones, we randomly sample items clicked by users and items extracted by Information Extractor, and use Mechanical Turk to judge their relevance. We sample 200 user clicked items, 200 filtered user clicked items, and 200 extracted items for actor movies and musician songs, and 100 for city tourism and national park lodging. The results are shown

[6] If equal number of workers vote for "Yes" and "No", this case is ignored. This happens in about 5% of cases.

Table 6: Numbers of entities and items clicked by users and numbers of those extracted by STRUCLICK

	Actor movies		Musician songs		City tourism		National park lodging	
	entity	*item*	*entity*	*item*	*entity*	*item*	*entity*	*item*
User clicked	1834	27906	962	10562	170	1097	18	68
Extracted	1.24M	121M	121K	21.2M	25932	5.8M	32027	19.4M
Final result	1.23M	11.7M	97232	1.75M	20789	285K	23338	955K

in Table 7. We can see a significant portion of user clicked items are irrelevant to the queries. For example, when searching for movies of an actor, a user often clicks on links such as “more”, “by year”, and other irrelevant items such as photos and user comments. Some of these links point to pages containing relevant movies, and therefore the Mechanical Turk workers consider the URLs as relevant, which makes the URLs have higher relevance than the item names. The data extracted by Information Extractor has comparable accuracy with user clicked items. On one hand, Information Extractor removes many irrelevant wrappers having repeated items from many pages (as described in Section 3.3). On the other hand, it introduces many new irrelevant items, because when an irrelevant wrapper is built, even from a very small number of clicked items, it can extract many irrelevant items. Fortunately Authority Analyzer can identify the relevant wrappers and generates good lists of items. By comparing Table 5 and Table 7, we can see the accuracy of STRUCLICK is much higher than the user clicked items and extracted items in most cases.

Table 7: Accuracy of user clicked items and extracted items

	Actor movies		Musician songs		City tourism		National park lodging	
	Name	*URL*	*Name*	*URL*	*Name*	*URL*	*Name*	*URL*
User clicked	.713	.970	.527	.964	.770	.962	.842	.967
Extracted	.735	.724	.747	.789	.780	.840	.932	.857
Noise	0	0	0	0	.148	.148	.053	.053

4.3.2 Examples of Output

Here we show some examples of the list of items combined from different sources by STRUCLICK. If different sources have slightly different names for each item, we choose the name of the source with highest relevance. The ranked lists of items (together with URLs for the first item) for four queries are shown in Table 8. We can see STRUCLICK successfully extracts relevant and popular items for each query.

4.3.3 Scalability and converging speed

We monitor the run-time of Information Extractor on building wrappers for each URL pattern, and using such wrappers to extract information from web pages. Figure 5 shows the run-time and number of URLs for building wrappers and extracting information for each URL pattern. Please note there are much less URLs for wrapper building because only URLs with follow-up clicks are useful. In general Information Extractor is linear scalable on both tasks. On the other hand, these two tasks can both be easily computed in distributed ways.

Because Authority Analyzer uses an iterative procedure which converges to optimal solution, we test how fast it converges. After each iteration Authority Analyzer generates a relevance score for each item. Let v_k be the vector of relevance scores after k^{th} iteration. We monitor the relative change of v_k, i.e., $\|v_k - v_{k-1}\| / \|v_{k-1}\|$, which is shown in Figure 6. It can be seen that Authority Analyzer converges quickly, and the relative change drops to 10^{-7} or 10^{-8} at iteration 100. We stop the procedure after the relative change is less than 10^{-7}, with at least 100 iterations.

Table 8: Example output of STRUCLICK

Query: {Britney Spears songs}

1. Baby One More Time
   http://www.kissthisguy.com/1874song-Baby-One-More-Time.htm
   http://www.poemhunter.com/song/baby-one-more-time/
   http://new.music.yahoo.com/britney-spears/tracks/baby-one-more-time--1486500
   http://album.lyricsfreak.com/b/britney+spears/baby+one+more+time_20001894.html
   http://www.mtv.com/lyrics/spears_britney/baby_one_more_time/1492102/lyrics.jhtml
   http://www.lyred.com/lyrics/Britney%20Spears/%7E%7E%7EBaby+One+More+Time/
2. Oops I Did It Again
3. Circus
4. (You Drive Me) Crazy
5. Lucky
6. Satisfaction
7. Everytime
8. Piece of Me
9. Radar
10. Toxic

Query: {Leonardo DeCaprio movies}

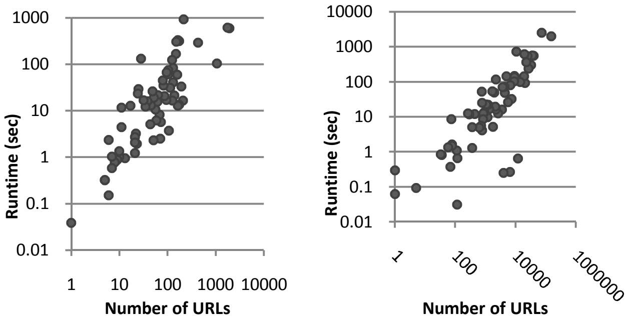
1. Body of Lies
   http://www.netflix.com/Movie/Body_of_Lies/70101694
   http://movies.yahoo.com/movie/1809968047/info
   http://www.hollywood.com/movie/Penetration/3482012
   http://us.imdb.com/title/tt0758774/
   http://movies.msn.com/movies/movie/body-of-lies/
   http://www.imdb.com/title/tt0758774/
2. Shutter Island (2009)
3. Revolutionary Road (2008)
4. Catch Me If You Can
5. Blood Diamond
6. The Departed
7. The Aviator
8. Conspiracy of Fools
9. Confessions of Pain (Warner Bros.)
10. The Low Dweller

Query: {Mount Rainier National Park lodging}

1. Crystal Mountain Village Inn
   http://www.tripadvisor.com/Hotel_Review-g143044-d1146125-Reviews-Crystal_Mt_Hotels-Mount_Rainier_National_Park_Washington.html
2. Cougar Rock Campground
3. Alta Crystal Resort at Mount Rainier
4. Travelodge Auburn Suites
5. Holiday Inn Express Puyallup (Tacoma Area)
6. Tayberry Victorian Cottage B&B
7. Crest Trail Lodge
8. Auburn Days Inn
9. Paradise Inn
10. Copper Creek Inn

Query: {Los Angeles tourism}

1. Universal Studios
   http://www.planetware.com/los-angeles/universal-studios-us-ca-uns.htm
   http://www.igougo.com/attractions-reviews-b80978-Universal_City-Universal_Studios_Hollywood.html
2. J. Paul Getty Center
3. Hollywood - Sunset Strip
4. Hollywood - Grauman's Chinese Theatre / Mann Theaters
5. Bunker Hill
6. El Pueblo de Los Angeles Historical Monument
7. Farmers Market
8. J Paul Getty Museum
9. Hollywood - Walk of Fame
10. Map of Los Angeles- Downtown



(a) Building wrappers (b) Extracting information

Figure 5: Run-time of Information Extractor

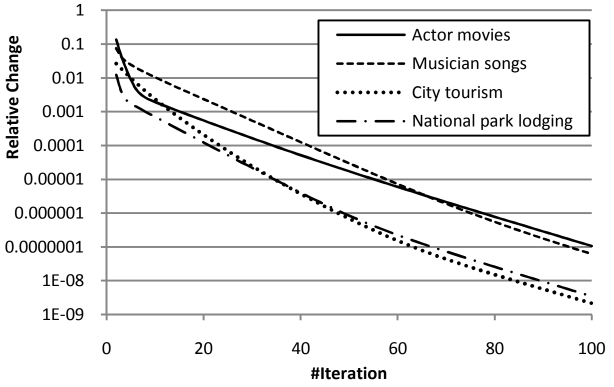


Figure 6: Convergence of Authority Analyzer

Then we test the scalability of Authority Analyzer. We randomly select 10000 to 50000 entities in musicians, and apply Authority Analyzer to their items. The number of items and run-time are shown in Figure 7, which shows linear scalability.

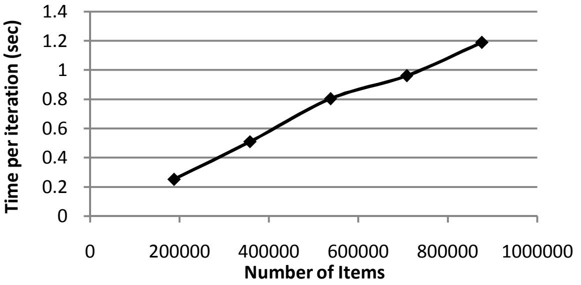


Figure 7: Scalability of Authority Analyzer

5. CONCLUSIONS

In this paper we present STRUCLICK, a fully automated system for extracting structured information from the web to answer web search queries. Comparing with existing approaches, it does not require manually labeled data, and can assign semantics to extracted data according to user queries. STRUCLICK utilizes users' search trails as implicit labels for wrapper building and information extraction, and can overcome the problem of high noise rate in such implicit labels. As many web sites provide uniformly formatted web pages for certain categories of name entities, STRUCLICK is capable of extracting large amounts of high-quality data for web search.

6. REFERENCES

[1] Amazon Mechanical Turk. https://www.mturk.com/mturk/

[2] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. *SIGMOD'03*.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. Semantic web. *Scientific American*, 1(1):68–88, 2000.

[4] M. Bilenko, R. W. White. Mining the search trails of surfing crowds: Identifying relevant websites from user activity. *WWW'08*.

[5] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. *IJCAI'07*.

[6] M. J. Cafarella, A. Halevy, N. Khoussainova. Data Integration for the Relational Web. *VLDB'09*.

[7] C. Chang and S. Lui. IEPAD: Information extraction based on pattern discovery. *WWW'01*.

[8] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large web sites. *VLDB'01*.

[9] S. Dill et al. SemTag and Seeker: Bootstrapping the semantic web via automated semantic annotation. *WWW'03*.

[10] X. Dong, L. Berti-Equille, and D. Srivastava. Truth discovery and copying detection in a dynamic world. *VLDB'09*.

[11] J. Guo, G. Xu, X. Cheng, and H. Li. Named entity recognition in query. *SIGIR'09*.

[12] N. Kushmerick. Wrapper induction for information extraction. PhD thesis (1997).

[13] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of Web data extraction tools. *ACM SIGMOD Record*, 31(2):84-93, 2002.

[14] X. Li, Y.-Y. Wang, and A. Acero. Learning query intent from regularized click graphs. *SIGIR'08*.

[15] B. Liu. Mining data records in Web pages. *KDD'03*.

[16] G. Miao, J. Tatemura, W.-P. Hsiung, A. Sawires, and L. E. Moser. Extracting data records from the web using tag path clustering. *WWW'09*.

[17] S. Mukherjee and I.V. Ramakrishnan. Automated semantic analysis of schematic data. *World Wide Web Journal*. 11(4): 427-464 (2008).

[18] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. *AGENTS'99*.

[19] M. Paşca. Organizing and searching the world wide web of facts - step two: harnessing the wisdom of the crowds. *WWW'07*.

[20] X. Yin, J. Han, and P. S. Yu. Truth discovery with multiple conflicting information providers on the web. *KDD'07*.

[21] Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. *WWW'05*.

[22] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, B. Schölkopf. Learning with local and global consistency. *NIPS'03*.

[23] D. Zhou, J. Huang, B. Schölkopf. Learning from labeled and unlabeled data on a directed graph. *ICML'05*.