# Combining URL and HTML Features for Entity Discovery in the Web

EDIMAR MANICA, Federal Institute of Rio Grande do Sul, Brazil
CARINA FRIEDRICH DORNELES, Federal University of Santa Catarina, Brazil
RENATA GALANTE, Federal University of Rio Grande do Sul, Brazil

The web is a large repository of entity-pages. An entity-page is a page that publishes data representing an entity of a particular type, for example, a page that describes a driver on a website about a car racing championship. The attribute values published in the entity-pages can be used for many data-driven companies, such as insurers, retailers, and search engines. In this article, we define a novel method, called **SSUP**, which discovers the entity-pages on the websites. The novelty of our method is that it combines URL and HTML features in a way that allows the URL terms to have different weights depending on their capacity to distinguish entity-pages from other pages, and thus the efficacy of the entity-page discovery task is increased. **SSUP** determines the similarity thresholds on each website without human intervention. We carried out experiments on a dataset with different real-world websites and a wide range of entity types. **SSUP** achieved a 95% rate of precision and 85% recall rate. Our method was compared with two state-of-the-art methods and outperformed them with a precision gain between 51% and 66%.

CCS Concepts: • **Information systems** → **Wrappers (data mining)**; **Web crawling**; *Site wrapping*;

Additional Key Words and Phrases: URL and HTML features, entity-pages, crawler, web structure mining

**ACM Reference format:**
Edimar Manica, Carina Friedrich Dorneles, and Renata Galante. 2019. Combining URL and HTML Features for Entity Discovery in the Web. *ACM Trans. Web* 13, 4, Article 20 (December 2019), 27 pages.
https://doi.org/10.1145/3365574



## 1 INTRODUCTION

The web is a vast repository of data about real-world entities, such as people, products, books, and organizations. A web page that publishes data representing an entity of a particular type is named entity-page [5]. Entity-pages are among the most common forms of structured data available on the web [9]. A significant fraction of entity-pages are dynamically generated by populating fixed HTML (HyperText Markup Language) templates with content from a DBMS (database

Some of the material in this article was originally presented at the ICWE conference [23].
Authors' addresses: E. Manica, Federal Institute of Rio Grande do Sul, Rua Nelsi Ribas Fritsch, 1111, Ibirubá, Brazil, 98200-000; email: edimar.manica@ibiruba.ifrs.edu.br; C. F. Dorneles, Federal University of Santa Catarina, Campus Universitário Trindade, 410, Florianópolis, Brazil, 88049-900; email: dorneles@inf.ufsc.br; R. Galante, Federal University of Rio Grande do Sul, Av. Bento Gonçalves, 9500, Porto Alegre, Brazil, 91501-970; email: galante@inf.ufrgs.br.

1559-1131/2019/12-ART20 $15.00
https://doi.org/10.1145/3365574


Fig. 1. Example of an entity-page from the Formula 1 website. This page describes an entity (Fernando Alonso) of an entity-type (driver) through the following attributes: name, team, country, and podiums, among others.

management system) [14]. Figure 1 presents a fragment of the template-based entity-page[1] on the Formula 1 website that describes the driver *Fernando Alonso* by means of the following attributes: *name*, *team*, *country*, *podiums*, *points*, *date of birth*, and *place of birth*, among others.

The abundance of template-based entity-pages on the web has attracted the attention of data-driven companies, such as insurers, retailers, and search engines. These modern organizations use these types of pages as a source for data acquisition, which is carried out by web scrapers. A web scraper (wrapper or data extractor) is a program that turns web content into structured data using techniques ranging from partial tree alignment [8] to visual analysis of the rendered page [10, 11], regular expressions [30], and DOM (Document Object Model) tree mining [7, 21, 22, 27]. For example, given the entity-page shown in Figure 1, a web scraper returns the data presented in Table 1.

Web scraping is often the only viable data collection method for websites with template-based entity-pages, in particular when no API (Application Programming Interface) is available. Recent advancements in the field made accurate and fully automated wrapper induction at scale [26], thus making it a good complement to direct data purchase, crowdsourcing, and free-text extraction. One signal of the relevance of the area is the growing number of web scraping startups, for example, Import.io,[2] DiffBot,[3] and ScrapingHub.[4]

This article focuses on one problem of the application of web scraping on template-based entity-pages: the need for finding the entity-pages on the websites. For instance, consider the owner of a website about sports that desires to provide information about the drivers that participate in the

[1] Available at: https://www.formula1.com/en/championship/drivers/fernando-alonso.html. Accessed: 02/19/2017.
[2] Available at: https://www.import.io/. Accessed: 02/05/2019.
[3] Available at: https://www.diffbot.com/. Accessed: 02/05/2019.
[4] Available at: https://scrapinghub.com/. Accessed: 02/05/2019.

Table 1. Attribute Values Extracted from the Page Presented in Figure 1

| Attribute | Value |
|---|---|
| Name | Fernando Alonso |
| Team | McLaren |
| Country | Spain |
| Podiums | 97 |
| Points | 1814 |
| Grands Prix entered | 268 |
| World Championships | 2 |
| Highest race finish | 1 (x32) |
| Highest grid position | 1 |
| Date of birth | 29/07/1981 |
| Place of birth | Oviedo, Spain |

Formula 1 car racing championship. To achieve this goal, the developers apply a web scraper on the entity-pages of the Formula 1 official website to collect data about the drivers. However, they need to implement a way of finding the entity-pages about drivers on the website. In general, the developers must (1) crawl all the pages of the website and filter the pages that satisfy a URL pattern that was manually defined or (2) define manually a path from the homepage to the entity-pages that is composed of URL patterns and/or DOM patterns. However, an extensive crawler can interfere with the normal operation of the website [17], which is not desirable, and the manual annotations are unappealing in real-world scenarios because they are effort-consuming and error-prone. With the solution described in this article, a nonexpert user only needs to provide an entity-page of the website, and our method finds the other entity-pages taking into account the way the websites arrange the entity-pages to avoid browsing through unproductive regions. The gap of related work that we deeply treat is to define a method independent of specific HTML tags, and the pages do not need to be rendered in a web browser.

In this article, we extend our previous work [23] to develop **SSUP** (**S**tructurally **S**imilar **U**RLs and **P**ages), a novel method to discover entity-pages. **SSUP** combines features extracted from the URL and HTML of the pages. Our method does not require the user to provide similarity thresholds because it automatically defines them in each website. **SSUP** is independent of specific HTML tags and human-compiled encyclopedias, and the pages do not need to be rendered in a web browser. Given an example of an entity-page, **SSUP** returns the set of entity-pages in the website. For example, given the entity-page about *Fernando Alonso* on the Formula 1 website, **SSUP** discovers the set of entity-pages about drivers on the website. The originality of our method is that it combines the HTML and URL features by assigning different weights to the URL terms on the basis of their capacity to discriminate between entity-pages and other pages, and thus the efficacy of the entity-page discovery task is increased. We carried out experiments on a dataset with different real-world websites and a wide range of entity types. **SSUP** achieved a 95% rate of precision and an 85% recall rate. Our method was compared with two state-of-the-art methods and outperformed them with a precision gain between 51% and 66%.

## 2 RELATED WORK

This section presents the related works in the area of **SSUP**. First, we describe the methods that concentrate on entity-page discovery. After this, we introduce methods with other goals: page

classification, page clustering, entity ranking, and entity discovery in HTML tables. It is shown how they can be adapted to perform entity-page discovery, as well as the limitations of this kind of adaptation. Finally, we conduct a comparative analysis of the methods.

**Entity-page discovery.** *INDESIT* [4], *SDC* [31], and *GPP* [32] discover entity-pages on the websites of the surface web. These methods receive an example of an entity-page of a website and return the set of entity-pages of the same type that belong to the website. *INDESIT* explores only the HTML features. *SDC* combines the URL and HTML features. *GPP* combines the HTML and visual features extending the original HyLien algorithm [10]. *EOC* [16] crawls entity-pages in the deep web (pages obtained by HTML form submission). Given a list of websites, *EOC* returns the entity-pages of these websites. *EOC* combines the URL and content features as well as exploring query logs and knowledge bases. *DIADEM* [11] extracts attribute values published on websites of the deep or surface web. From the URL of the main page of a website, *DIADEM* discovers the pages that describe entities on this website and extracts the attribute values published on these pages. *DIADEM* combines the HTML and visual features and explores knowledge about the ontology and phenomenology of the entity type, that is, knowledge about entities and the representation of these entities in the textual, structural, and visual language of a website of this entity type. *DEXTER* [28] extracts product specifications on the web. A product specification is a set of attribute-value pairs. *DEXTER* finds websites about products on the web, detects entity-pages that describe products on these websites, and extracts the product specifications from these pages. *DEXTER* combines the HTML and content features, as well as making use of supervised classifiers, a search engine, and heuristics that take into account patterns that frequently occur on the web.

**Page classification or clustering.** *PEBL* [34] and *HCUD* [12] classify web pages. These methods explore the URL and content features and require the user to provide a training set with manually annotated examples (*PEBL* only needs positive examples). *PEBL* and *HCUD* can be adapted to carry out entity-page discovery since the user provides a training set. From this training set, they classify the pages of the website as either entity-pages or non-entity-pages. The limitations of this kind of adaptation include the need to (1) download all the pages of the website and (2) provide many examples of entity-pages because a single example is not enough for them to learn the features of the entity-pages. *CALA* [17] and *MDL-UC* [6] cluster the pages from a given website so that the pages generated by the same template are in the same cluster. *CALA* is only based on the URL features, while *MDL-UC* extracts features from URL, HTML, and the content of the pages. *CALA* and *MDL-UC* can be adapted to carry out entity-page discovery since the user annotates the cluster that contains the entity-pages. The limitations of this adaptation include the following: (1) the need for user annotation and (2) the need to download several pages of the website because a large training set, which includes both entity-pages and non-entity-pages, must be provided.

**Other methods.** *W-ER* [19] finds pages representing entities that are relevant to a query. *W-ER* explores language models [18], Wikipedia, and a search engine. The limitations of using *W-ER* to discover entity-pages include the following factors: (1) the user must formulate a proper query (i.e., the user must describe the entity-pages of interest through keywords, rather than providing an example of an entity-page), and (2) *W-ER* only returns pages that describe entities that have pages on Wikipedia. *EDAT* [29] is designed to discover and annotate entities in HTML tables. Given an HTML table, *EDAT* (1) identifies the rows that contain information on entities of a particular type and (2) determines the cells that contain the names of those entities. *EDAT* combines ontology, a search engine, a text classifier (e.g., SVM), and heuristics that are based on specific patterns that often occur in HTML tables. *EDAT* can be adapted to perform entity-page discovery. If a table contains a column with links to the entity-pages that have more details about the entities described in the table, it is only necessary to identify this column. *ERD* [20] also explores hyperlinks that are

Table 2. Comparison of the Methods

| | PEBL | INDESIT | SDC | SSM | ERD | W-ER | MDL-UC | EDAT | EOC | GPP | DIADEM | HCUD | CALA | DEXTER | SSUP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $C_1$ discovery task | ~ | ✓ | ✓ | X | ~ | ✓ | ~ | ~ | ✓ | ✓ | ✓ | ~ | ~ | ✓ | ✓ |
| $C_2$ crawling strategy | X | ✓ | ~ | N/A | X | ~ | X | X | N/A | ~ | N/A | X | N/A | ✓ | ✓ |
| $C_3$ supervision-independent | X | ✓ | ✓ | X | X | ✓ | ~ | X | ✓ | ✓ | X | X | ~ | X | ✓ |
| $C_4$ rendering-independent | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | X | X | ✓ | ✓ | ✓ | ✓ |
| $C_5$ type-independent | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | X | ✓ | X | ✓ |
| $C_6$ search engine-independent | ✓ | ✓ | ✓ | ✓ | ✓ | X | ✓ | X | X | ✓ | ✓ | ✓ | ✓ | X | ✓ |
| $C_7$ knowledge base-independent | ✓ | ✓ | ✓ | ✓ | ✓ | X | ✓ | X | X | ✓ | X | X | ✓ | ✓ | ✓ |
| $C_8$ markup-independent | ✓ | ✓ | ✓ | ✓ | X | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| $C_9$ threshold-independent | ✓ | X | X | N/A | ✓ | ✓ | X | ✓ | X | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

*Note*: **SSUP** is the method described in this article.

contained in web tables for discovering new entity relations. The limitation of *EDAT* and *ERD* is that the links to the entity-pages must be in the HTML tables. *SSM* [8] is an approach to extract attribute-value pairs from entity-pages. This approach uses the partial tree alignment method of DEPTA [35] to detect data fields. The limitation of *SSM* is that it requires, as input, the entity-pages of the websites. Therefore, our method **SSUP** is complementary to *SSM* since **SSUP** finds the entity-pages and *SSM* extracts the attribute-value pairs from these pages.

**Comparison.** First, we set out the characteristics used in the comparison. Following this, the results of the analysis of the characteristics are shown; these are summarized in Table 2. In that table, the following symbols denote (1) ✓: the method supports the characteristic; (ii) ×: the method does not support the characteristic; (iii) *N/A*: the characteristic is not applicable to the method; and (iv) ~ : the characteristic is partly supported by the method.

The characteristics have been selected to determine if the methods can be successfully applied in real-world scenarios, which is our focus. The following characteristics are analyzed and indicate if the method ($C_1$) discovers the entity-pages on the website, ($C_2$) includes a crawling strategy that avoids browsing through unproductive regions of the website, that is, a strategy that crawls the maximum number of entity-pages by traversing the minimum number of pages of the website; ($C_3$) - is "supervision-independent", i.e., the user does not need to provide annotations; ($C_4$) is "rendering independent", that is, the pages do not need to be rendered in a web browser; ($C_5$) is "type independent", that is, the method is able to deal with different entity types; ($C_6$) is "search engine independent", that is, the method does not use a search engine; ($C_7$) is "knowledge base independent", that is, the method does not need to use a thesaurus, ontology, lexical base, or knowledge base; ($C_8$) is "markup independent", that is, the method does not depend on specific HTML markups (e.g., <*table*>, <*ul*>, etc.); and ($C_9$) is "threshold independent," that is, the user does not need to define similarity thresholds.

*INDESIT*, *SDC*, *W-ER*, *EOC*, *GPP*, *DIADEM*, and *DEXTER* discover the entity-pages on the websites. *SSM* only extracts attribute-value pairs from the entity-pages, and then it is necessary to apply a method like **SSUP** to find these pages previously. Other methods can be adapted to carry out this task. However, the adaptations are subject to the previously described limitations.

*PEBL*, *ERD*, *MDL-UC*, *EDAT*, and *HCUD* require the application of an extensive crawler to the website over analysis in order to obtain the pages that are evaluated. However, this crawler can interfere with the normal operation of the website [17], which is not desirable. *W-ER* uses a search engine to find the entity-pages. In view of this, the search engine must collect the pages of the website. *SDC* and *GPP* adopt a strategy that involves crawling the entity-pages through breadth-first searches, from the main page of the website to the entity-pages. As a result of these searches, several unproductive paths are traversed. *EOC*, *DIADEM*, and *CALA* focus on the deep web, and then they employ HTML form submissions instead of a crawler that traverses the pages of the

website. *SSM* requires the entity-pages as input in order to extract the attribute-value pairs from them. *INDESIT* and *DEXTER* take into account the way the websites arrange the entity-pages to avoid browsing through unproductive regions.

*PEBL*, *SSM*, *ERD*, *EDAT*, *DIADEM*, *HCUD*, and *DEXTER* require *a priori* supervision; that is, the user must provide annotations before the execution of the method. *MDL-UC* and *CALA* need *a posteriori* supervision; that is, the user must provide annotations after the execution of the method. Annotations are unappealing in real-world scenarios because they are effort-consuming and error-prone. *INDESIT*, *SDC*, *W-ER*, *EOC*, and *GPP* do not require the user to provide any annotations.

*GPP* and *DIADEM* employ visual features. To extract these features, the pages must be rendered in a web browser, which loads all the images, CSS, and JavaScript. However, this rendering is not suitable for real-world scenarios because it significantly increases the processing time (as shown in the experiments). The other methods do not require page rendering.

*HCUD* only deals with entity-pages about researchers, while *DEXTER* focuses on products. These methods can only be employed in a determined entity type. The other methods can be applied to different entity types and then can be used in several application domains.

*EOC* requires the query logs of a search engine. However, it is difficult for people who are not employed in a commercial search engine to obtain a large set of query logs [33]. *W-ER*, *EDAT*, and *DEXTER* submit queries to a search engine. But the search engines restrict the daily number of free queries. The other methods do not require either query logs or queries in a search engine.

*W-ER*, *EDAT*, *EOC*, *DIADEM*, and *HCUD* use some kind of knowledge base. Different types of knowledge bases are employed, such as a lexical base (WordNet), ontology, and a human-compiled encyclopedia (Wikipedia). The need for a knowledge base limits the application of the method to a language that has this kind of resource [17]. Weninger et al. [32] state that the vast majority of entities found on the web are not in Wikipedia. We also observed this situation. For example, São Paulo, the largest Brazilian city, has 55 councilors, but only 9 of them have pages on Wikipedia. The other methods do not require knowledge bases.

*EDAT* and *ERD* require the entities to be arranged in HTML tables. However, CSS has enabled designers to move away from table-based layouts, and many of the structural cues used in table-based methods have been eradicated from HTML [25]. The other methods are not limited to specific HTML markups.

*INDESIT*, *SDC*, *MDL-UC*, and *EOC* require the user to define the similarity threshold that distinguishes entity-pages from other pages manually. This definition is an effort-consuming task since it requires an expert user to set the threshold. Moreover, each website has a different proper similarity threshold (as shown in the experiments). *SSM* requires the entity-pages as input, then does not need a similarity threshold to distinguish entity-pages from other pages. The other methods do not require the user to manually define the similarity threshold.

The main contribution of **SSUP** is to fulfill all the listed characteristics. **SSUP** discovers the entity-pages on the websites by using a crawling strategy with the aim of visiting the minimum number of pages of the website. Our method requires neither the page rendering nor the user to provide annotations. **SSUP** is independent of search engines, knowledge bases, and specific HTML markups. The described method is applicable to different entity types. Finally, in each website **SSUP** automatically defines the similarity thresholds that distinguish the entity-pages from other pages. Although our method is threshold independent, it is not parameter independent since it requires tuning a parameter $H$, which estimates the height of the entity-tree. We found in the experiments that the value 5 for this parameter produces an adequate efficacy in all the websites. However, in websites that have an entity-tree with a height lower than 5, if we decrease the value of the $H$ parameter, the processing time of **SSUP** decreases.

To emphasize the contributions of our method, we provide a conceptual comparison between **SSUP** and the closely related methods. **SSUP** reuses the definitions of *INDESIT* that are related to the representation of HTML of the pages. However, in contrast with *INDESIT*, **SSUP** combines URL and HTML features. **SSUP** represents the URLs in a very similar way to *MDL-UC* but treats this representation by following the TF-IDF principle [3], while *MDL-UC* uses the MDL principle [13]. Like *GPP* and *SDC*, **SSUP** assumes that the path from the homepage to an entity-page is usually designed to navigate the user through a logical hierarchy of topics and increasingly finer subtopics. **SSUP** finds this hierarchy by starting at a leaf of the hierarchy, while *GPP* and *SDC* start at the root. *SDC* and **SSUP** exploit the URL and HTML features of the pages, which are handled in a different way. *SDC* computes the HTML similarity by means of a tree edit distance. **SSUP** computes the HTML similarity using a syntactic similarity function (Jaccard coefficient) in the paths of the DOM tree that start at the root and end at the anchor nodes. The technique used by **SSUP** is less costly with regard to processing time and is less influenced by misalignment among the entity-pages. **SSUP** gives a different weight to URL terms depending on their discriminatory capacity, while *SDC* gives the same weight to all the URL terms. *SDC* requires the user to manually provide the similarity thresholds that distinguish entity-pages from other pages, while **SSUP** automatically defines the thresholds in each website.

*INDESIT* and *GPP* were chosen as baselines because (1) they employ a crawler strategy that seeks to download the minimum number of pages in the website to find the entity-pages, (2) they are independent of specific HTML markups (e.g., <*table*>, <*ul*>, etc.), (3) they do not depend on human-compiled encyclopedias, and (4) they do not require a large training set.

## 3 FUNDAMENTAL CONCEPTS

In this section, we present some background information that is necessary to properly understand our method, since some of the used concepts are recent and nontrivial. The concept `site` refers to a rooted directed graph, where (1) the nodes are the pages on the web that exist within the same URL domain, (2) the edges correspond to the hyperlinks, and (3) the root is the main page [32]. A `page` is a pair $< d, u >$, where $d$ is the Document Object Model (DOM) tree and $u$ is the URL. The `link-path` is the name for the path in the DOM tree of a page that starts from the root and terminates at an anchor node (tag $< a >$) [4]. We say that the link-path (1) "`is on`" the page that contains the anchor node and (2) "`is of`" the page that is pointed by the anchor node.

A part of a website is designed to allow the user to navigate from the homepage until the pages that describe the entities of the same type, through a logical hierarchy of topics [32]. This hierarchy is referred to as the `entity-tree`. There are many categories of entities, called `entity types`. The entities of the same type are entities that have the same attributes published on the website. The pages that are part of the entity-tree are classified as either entity-pages or index-pages. `Entity-pages` are pages that publish data describing an entity of a particular type [4]. Entity-pages are the leaves of the entity-tree and entity-pages of the same type are entity-pages that describe entities of the same type. `Index-pages` are pages that point to (i.e., that have hyperlinks to) entity-pages or other index-pages, and their role is to group entity-pages/index-pages to allow the user to navigate through the logical hierarchy of topics. Index-pages are all the pages in the entity-tree that are not entity-pages. The pages that do not belong to the entity-tree are called `noise-pages`.

Figure 2 describes a website of a fictitious racing championship.[5] This website, referred to as $S$, is used to exemplify the concepts in this section and explain the method in Section 4. The rectangles correspond to the pages of the $S$ website and the edges correspond to the hyperlinks. The root of

[5]We use a fictitious website as an example in order to include features of various real-world websites.

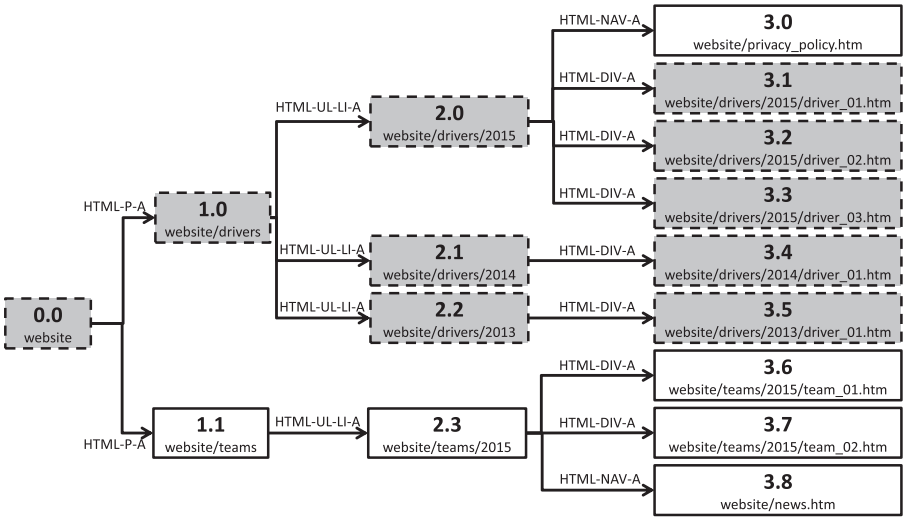



Fig. 2. Example of a website of a fictitious racing championship.

the $S$ website is Page 0.0. The pages are identified with $l.p$, where $l$ represents the level of the page from the root and $p$ is an ID that distinguishes the pages that are at the same level. For instance, Page 2.1 is the page with ID 1 at level 2. The $S$ website describes two types of entities: driver and team. There is an entity-tree for each type. We concentrate on the driver type because we use it to explain **SSUP**. The entity-tree about drivers is highlighted with a solid background. Pages 3.$i$ ($1 \leq i \leq 5$) are entity-pages that describe a specific driver. Pages 2.$j$ ($0 \leq j \leq 2$) are index-pages pointing to entity-pages about drivers in a determined year. Page 1.0 is the index-page that points to all the index-pages at level 2. Page 0.0 is the index-page that points to the index-page at level 1. The other pages are noise-pages in the driver context. The label at the edges is the link-path. The link-path on the edge from a page $p_x$ to a page $p_y$ is the link-path that is on the $p_x$ page pointing to the $p_y$ page. We say that the link-path is on the $p_x$ page and is of the $p_y$ page.

Table 3 summarizes the concepts defined in this section. This summary is a means of guiding the reader to understand our method.

## 4 SSUP: A URL-BASED METHOD TO DISCOVER ENTITY-PAGES

**SSUP** is a method for entity-page discovery, which finds the set of entity-pages of the same type on a website given an example of an entity-page. The input of the method is an entity-page, called `sample page`. The output consists of the entity-pages of the same type as the `sample page`. Two stages are carried out: (1) *Identification of the Page-Path*—finds and returns the `page-path`, that is, the path that starts at the root of the entity-tree and ends at the `sample page`, and (2) *Traversal of the Entity-Tree*—traverses the entity-tree that contains the `sample page` and returns its entity-pages.

The *Identification of the Page-Path* stage receives as input a `sample page`. The `sample page` is added to a stack that represents the page-path. The index-page of the page that is at the top of the stack (i.e., the `sample page`) is obtained and added to the stack. If the page that is at the top of the stack (i.e., the last obtained index-page) is not the root of the entity-tree, the index-page of the page that is at the top of the stack is obtained and added to the stack. If the page that is at the top of the stack is the root of the entity-tree, the *Identification of the Page-Path* stage is finalized.

Table 3. Summary of the Concepts

| Concept | Definition |
|---|---|
| Site | a rooted directed graph whose the nodes are the pages on the Web that exist within the same URL domain, the edges correspond to the hyperlinks, and the root is the main page |
| Page | a pair <*d*,*u*>, where *d* is the Document Object Model (DOM) tree and *u* is the URL |
| link-path | the path in the DOM tree of a page that starts from the root and terminates into an anchor node |
| a link-path **is on** a page | when the page contains the anchor node |
| a link-path **is of** a page | when the page is pointed by the anchor node |
| entities of the same type | entities that have the same attributes published on the site |
| entity-tree | a part of a site that is designed to allow the user to navigate from homepage until the pages that describe the entities of the same type through a logical hierarchy of topics |
| entity-page | a page that publishes data describing an entity of a particular type |
| entity-pages of the same type | entity-pages that describe entities of the same type |
| index-page | a page with links to entity-pages or to other index-pages, whose role is to group entity-pages/index-pages to allow the user to navigate through the logical hierarchy of topics |
| noise-page | a page of the site that do not belong to the entity-tree |

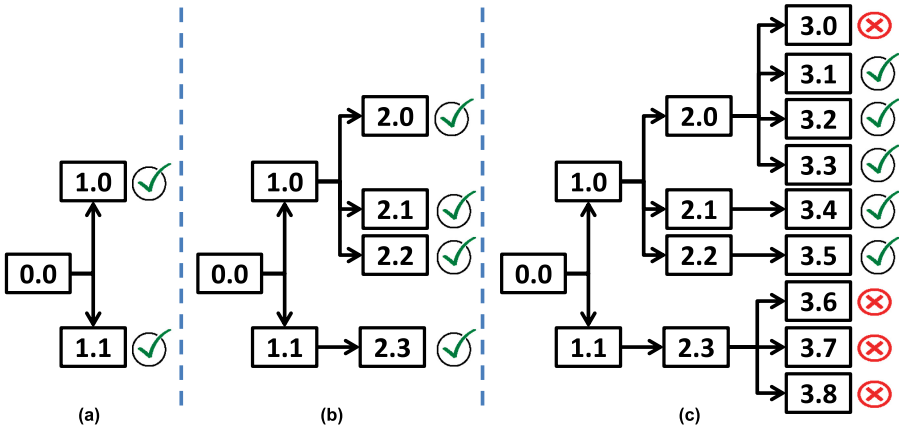



Fig. 3. An execution of the traversal of the entity-tree stage.

The output is a stack that represents the page-path. The top of the stack contains the root of the entity-tree, while the bottom of the stack contains the `sample page`.

The *Traversal of the Entity-Tree* stage receives as input a stack that represents the page-path. The root of the entity-tree (i.e., the top) is removed from the stack. The pages pointed by the root of the entity-tree are obtained, and these are called `child-pages`. The child-pages are filtered with regard to their similarity with the page that is at the top of the stack. The child-pages that remain are called `similar-pages`. The top is removed from the stack. If the stack is not empty, the pages pointed by the similar-pages are obtained and filtered; also, the top is removed from the stack. If the stack is empty, the similar-pages obtained in the last iteration are returned.

Here, we provide an example of the execution of **SSUP** by using the *S* website described in Figure 2. Figure 3(c) presents a simplified version of this website. Given Page 3.2 as `sample page`, **SSUP** returns pages 3.$k$ ($1 \leq k \leq 5$). The details of the example are as follows. The input is Page 3.2. First, Page 3.2 is added to the stack that represents the page-path. Then, the index-page of

Page 3.2 is obtained (2.0) and added to the stack. After this, the index-page of Page 2.0 is obtained (1.0) and added to the stack. Next, the index-page of Page 1.0 is obtained (0.0) and added to the stack. Thus, the page-path is complete and formed by "$0.0 \rightarrow 1.0 \rightarrow 2.0 \rightarrow 3.2$". At this moment, the *Identification of the Page-Path* stage is finished and the *Traversal of the Entity-Tree* stage starts. First (Figure 3(a)), the child-pages of the root of the entity-tree are obtained: 1.0 and 1.1. Then, the child-pages are filtered with regard to their similarity with Page 1.0. The two child-pages are similar. Next (Figure 3(b)), the child-pages of Pages 1.0 and 1.1 are obtained: $2.i$ ($0 <= i <= 3$). After this, the child-pages are filtered with regard to their similarity with Page 2.0. The four child-pages are similar. Next (Figure 3(c)), the child-pages of pages $2.i$ ($0 <= i <= 3$) are obtained: $3.j$ ($0 <= j <= 8$). The child-pages are filtered with regard to their similarity with Page 3.2. Only pages $3.k$ ($1 <= k <= 5$) are similar. Finally, these similar pages are returned.

In the following subsections, we describe both stages of our method in detail. Section 4.2 is concerned with the *Identification of the Page-Path* stage, while Section 4.3 describes the features of the *Traversal of the Entity-Tree* stage. Before this, Section 4.1 defines the functions used to compute the URL and HTML similarity between the pages. These functions are employed in both stages of our method.

### 4.1 The Structural Similarity Functions

In this subsection, we present the functions that have been used to measure the structural similarity between two pages. We specify one function based on the URL, called `Strong URL Similarity`. Another function used is based on the HTML, called `HTML Similarity`, designed by Blanco et al. [4].

*4.1.1 Strong URL Similarity.* The Strong URL Similarity compares two pages based on the terms that belong to their URLs. This function assigns a different weight to each URL term depending on its capacity to distinguish the entity-pages of the same type from the other pages. **SSUP** uses the Strong URL Similarity function in the *Identification of the Page-Path* and *Traversal of the Entity-Tree* stages. The Strong URL Similarity is explained in the following order: (1) how the terms are extracted from URLs, (2) how the URL terms are weighted, and (3) how the weights of the URL terms are used to compute the similarity between two pages.

The URL is tokenized to extract URL terms. A URL is regarded as a sequence of substrings (called `tokens`) split by the "/", "?" or "&" characters. Each token is a set of substrings (called `subtokens`) split by nonalphanumeric characters, changing from letter to digit and vice versa. Thus, a URL term is a subtoken that is concatenated with the position of the token that contains it.

*Definition 4.1 (`URL Term`).* Let $T = \{t_1, \ldots, t_n\}$ be a sequence of tokens of a URL $u$, where $t_i$ occurs in $u$ before $t_{i+1}$. Each token $t_i = \{s_i[1], \ldots, s_i[m]\}$ is a set of subtokens in which each $s_i[j]$ is a subtoken of the $t_i$ token. Each subtoken $s_i[j]$ concatenated with $i$ is a URL term. An additional URL term is the size of $T$.

Figure 4 shows the URL of Page 3.2 in the $S$ website (Figure 2). All the tokens, subtokens of the $t_4$ token, and URL terms are presented. The additional URL term is highlighted as well.

The weight of a URL term is based on two observations about the entity-pages of the same type: (1) they usually share a common URL structure, and (2) they are usually pointed by the index-page through the same link-path. From these observations, we assume that terms that occur in many URLs pointed by the index-pages through the same link-path as the `sample page`, and only occur in a few link-paths from the index-pages, are more important because they have a high discriminating capacity.

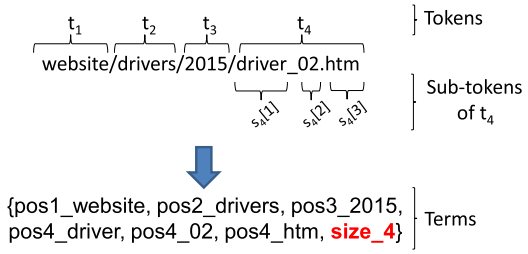



Fig. 4. Example of tokens, subtokens, and URL terms. The tokens of the URL *website/drivers/2015/driver_02.htm* are $t_1$, $t_2$, $t_3$, and $t_4$. The subtokens of $t_4$ are $s_4[1]$, $s_4[2]$, and $s_4[3]$. The terms of this URL are *pos1_website*, *pos2_drivers*, *pos3_2015*, *pos4_driver*, *pos4_02*, and *pos4_htm*. The additional URL term is *size_4*.

*Definition 4.2 (*`URL Term Weight`*).* Let *ip* be the index-page of a `sample page` *sp*. Let $P = \{p_1, \ldots, p_n\}$, where there is a $p_i$ for each link-path on *ip*. Let $p_i = \{p_i[1], \ldots, p_i[m]\}$, where there is a $p_i[j]$ for each URL pointed by *ip* through $p_i$. Let $p_i[j] = \{p_i[j][1], \ldots, p_i[j][o]\}$, where there is a $p_i[j][k]$ for each term of URL $p_i[j]$. Let $p_{sp}$ be the $p_i$ link-path that points to *sp*. Let *t* be a URL term from *sp*. The weight of *t* from *sp* in *ip* is defined as:

$$W_t(ip, sp) = TF_{t,p_{sp}} \times IDF_{t,P}, \tag{1}$$

where $TF_{t,p_{sp}}$ is the number of URLs in $p_{sp}$ that contains the *t* term and $IDF_{t,P}$ is defined as:

$$IDF_{t,P} = log\left(\frac{|P|}{|p_i \in P : p_i[j] \in p_i : t \in p_i[j]|}\right).$$

For example, take Page 3.2 on the *S* website (Figure 2) as the `sample page`. The terms "*pos2_drivers*" and "*pos4_driver*" must have the highest weight because they occur in five URLs (pages 3.1, 3.2, 3.3, 3.4, and 3.5) pointed by the index-pages (pages 2.0, 2.1, and 2.2) through the same link-path ("*HTML-DIV-A*") as the `sample page` and do not occur in (URLs pointed by the index-pages through) other link-paths.

The Strong URL Similarity accumulates the weight of the URL terms shared by the `sample page` and the target page (i.e., the page that is being compared with the `sample page`). The URL terms are assumed to be all mutually independent. The URL of the `sample page` is represented as a vector of URL term weights in an *n*-dimensional space, in which *n* is the total number of URL terms.

*Definition 4.3 (*`Strong URL Similarity`*).* Let $K = \{k_1, \ldots, k_n\}$ be the set of URL terms of a `sample page` *sp*. Let $L = \{l_1, \ldots, l_m\}$ be the set of URL terms of a target page *tp*. Let *ip* be the index-page of *sp*. Let $W_t$ be the weight of the URL term *t*. The Strong URL Similarity between *sp* and *tp* is defined as:

$$strongsim_{url}(ip, sp, tp) = \frac{\sum_{t \in K \cap L} W_t(ip, sp)}{\sum_{t \in K} W_t(ip, sp)}. \tag{2}$$

For example, assume that Page 3.2 on the *S* website (Figure 2) is the `sample page`. The Strong URL Similarity between Page 3.2 and Page 3.1 is 0.94. The Strong URL Similarity between Page 3.2 and Page 3.7 is 0.62.

*4.1.2 HTML Similarity.* HTML Similarity, which is defined here in accordance with Blanco et al. [4], is a similarity function that compares two pages on the basis of the link-paths belonging to their HTML. This function gives the same importance to each link-path. **SSUP** employs the HTML

Similarity function in the *Traversal of the Entity-Tree* stage. The HTML similarity is defined by $sim_{html}$ as follows:

$$sim_{html}(p_1, p_2) = \frac{|L^1 \cap L^2|}{|L^1 \cup L^2|}, \tag{3}$$

where $p_1$ and $p_2$ are pages; $L^i = \{l_1^i, \ldots, l_n^i\}$ is the set of link-paths of a page $p_i$.

It should be noted that Strong URL Similarity and HTML Similarity use the concept of link-path in different contexts. In the Strong URL Similarity, the link-path of a page $p_x$ is the link-path that points to $p_x$ in its index-page. In the HTML Similarity, the set of link-paths of a page $p_x$ contains the link-paths in the HTML of $p_x$.

### 4.2 Identification of the Page-Path

The goal of the *Identification of the Page-Path* stage is to find the `path-page`, that is, the path from the root to the `sample page` in the entity-tree. The page-path is obtained in a reverse order. For example, taking the *S* website (shown in Figure 2), **SSUP** starts with the `sample page` (3.2), finds its index-page (2.0), discovers the index-page of the index-page (1.0), and so on until it reaches the root of the entity-tree (0.0).

**SSUP** determines whether the root of the entity-tree has been reached by means of a parameter that estimates its height. If this parameter is less than the real height of the entity-tree, some entity-pages may be missing. If this parameter is greater than the actual height of the entity-tree, some unnecessary pages are downloaded, but the effectiveness of the method is not affected since the noise-pages are filtered out in the next stage. **SSUP** has an additional stopping criterion: when the selected index-page is already in the stack that represents the page-path.

**SSUP** finds the index-page of a given page by assuming that the index-page of a given page *gp* is the page that delivers the largest number of more structurally similar URLs to the *gp* page. This number is represented through the *index_score* that accumulates the URL similarity between the given page and the pages pointed by a candidate index-page.

*Definition 4.4 (Index_Score).* Let *cp* be a candidate index-page for the given page *gp*. Let $C = \{c_1, \ldots, c_n\}$ be a set of pages pointed by *cp* through the same link-path as *gp*. The *index_score* of *cp* for *gp* is defined as:

$$index\_score(cp, gp) = \sum_{c_i \in C} strongsim_{url}(cp, gp, c_i). \tag{4}$$

This article specifies Algorithm 1, `Top Index-page`, to find the index-page of a given page. The algorithm collects the pages pointed by the given page (line 1) and by these collected pages (lines 2 to 4). The pages that do not point to the given page are removed (line 5). The pages pointed through the same link-path as the given page are removed (line 6). Finally, the algorithm returns the page with the maximum *index_score* (line 7). This article extends the algorithm of the previous version published at the ICWE [23] by including lines 2 to 4 and 6. We added lines 2 to 4 after observing that some entity-pages do not have a direct link to their index-pages. Line 6 was included to filter out pages that are probably not index-pages.

### 4.3 Traversal of the Entity-Tree

The goal of the *Traversal of the Entity-Tree* stage is to find the entity-pages that belong to the entity-tree where the page-path was found in the previous stage. These pages are obtained by traversing the entity-tree from the root to the leaves. In each iteration (i.e., level of the entity-tree), the pages pointed by the pages selected in the previous iteration (or by the root of the entity-tree in the first iteration) are obtained and filtered. The pages are filtered with regard to their similarity with the

**ALGORITHM 1:** Top Index-page Algorithm

**Input**: a given page ($gp$).
**Output**: the index-page of $gp$.

```
P = A = GetLinks(gp);
for each a_i ∈ A do
    add GetLinks(a_i) to P;
end
remove p_i from P where p_i does not have a link to gp;
remove p_i from P where the link-path of p_i is equal to the link-path of gp;
return p_i from P with max index_score(p_i, gp); // Equation 4
```

page that is at the top of the stack (i.e., the page in the page-path that is at the same level in the entity-tree as the pages that are being analyzed).

This article defines Algorithm 2, `Filter`, to filter the pages obtained in each iteration of the *Traversal of the Entity-Tree* stage. The input consists of a set of pages obtained in the current iteration and a `sample page` (the page that is at the top of the stack). The output consists of the pages that are similar to the `sample page`. The similarity is evaluated in terms of URL and HTML.

**ALGORITHM 2:** Filter Algorithm

**Input**: a set of pages ($L$), a `sample page` $sp$.
**Output**: the set of pages in $L$ that are similar to $sp$.

```
add sp to rs;
remove from L each page l_i where link-path-of(l_i) <> link-path-of(sp)
create a list of groups G = {G_1, ..., G_n}, where each group G_i contains each page l_i ∈ L (except sp)
    with the same strongsim_url between sp and l_i;
sort G in decreasing order of strongsim_url;
add all the pages from G_1 to rs;
minsim_html = min_{l_i ∈ G_1} sim_html(sp, l_i);
remove G_1 from G ;
for each G_i IN G do
    if max_{l_i ∈ G_i} sim_html(sp, l_i) >= minsim_html then
        add all the pages from G_i to rs;
    else
        break;
    end
end
return rs;
```

The `Filter` algorithm can be clarified through an example of its execution where Level 3 of the entity-tree about drivers in the $S$ website (Figure 2) is analyzed. Figures 5(a) and 5(b) guide the example. Figure 5(a) describes the input that comprises nine pages. The highlighted page (3.2) is the `sample page`. The following information of each page is included: (1) ID: the identifier of the page; (2) link-path: the link-path in the index-page that points to the page; (3) $strongsim_{URL}$: the Strong URL similarity between the page and the `sample page`; and (4) $sim_{HTML}$: the HTML similarity between the page and the `sample page`. Figure 5(b) shows the value of four significant variables along the execution of the algorithm: (1) $L$: list of pages that must be analyzed; (2) $rs$: set of pages that is returned as output; (3) $G$: list of groups of the pages, where each group contains

| ID | Link-path | strong $sim_{URL}$ | $sim_{HTML}$ |
|---|---|---|---|
| 3.0 | HTML-NAV-A | - | - |
| 3.1 | HTML-DIV-A | 0.94 | 0.85 |
| **3.2** | **HTML-DIV-A** | **1.00** | **1.00** |
| 3.3 | HTML-DIV-A | 0.94 | 0.75 |
| 3.4 | HTML-DIV-A | 0.75 | 0.80 |
| 3.5 | HTML-DIV-A | 0.75 | 0.82 |
| 3.6 | HTML-DIV-A | 0.56 | 0.45 |
| 3.7 | HTML-DIV-A | 0.62 | 0.40 |
| 3.8 | HTML-NAV-A | - | - |

(a) input

| Line(s) in the Algorithm | L | G | $minsim_{HTML}$ | rs |
|---|---|---|---|---|
| Initialization | (3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8) | () | - | () |
| 01 | (3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8) | () | - | (**3.2**) |
| 02 | **(3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7)** | () | - | (3.2) |
| 03-04 | - | **$G_1$ $G_2$ $G_3$ $G_4$ ((3.1, 3.3), (3.4, 3.5), (3.7), (3.6))** | - | (3.2) |
| 05 | - | ((3.1, 3.3), (3.4, 3.5), (3.7), (3.6)) | - | (3.2, **3.1, 3.3**) |
| 06 | - | ((3.1, 3.3), (3.4, 3.5), (3.7), (3.6)) | **0.75** | (3.2, 3.1, 3.3) |
| 07 | - | **$G_2$ $G_3$ $G_4$ ((3.4, 3.5), (3.7), (3.6))** | 0.75 | (3.2, 3.1, 3.3) |
| 08-14 (1st iteration) | - | ((3.4, 3.5), (3.7), (3.6)) | 0.75 | (3.2, 3.1, 3.3, **3.4, 3.5**) |
| 08-14 (2nd iteration) | - | ((3.4, 3.5), (3.7), (3.6)) | 0.75 | (3.2, 3.1, 3.3, 3.4, 3.5) |

(b) step by step of the Filter algorithm

Fig. 5. An execution of the Filter algorithm.

pages with the same URL similarity (between them and the `sample page`); and (4) $minsim_{HTML}$: the HTML similarity threshold, which is defined automatically. The first column identifies the lines in the `Filter` algorithm and the following columns show the value of the variables.

The details of the example are as follows. In line 01 of the `Filter` algorithm (following through the first column of the table presented in Figure 5(b)), the `sample page` (3.2) is added to the *rs* set. In line 02, Pages 3.0 and 3.8 are removed from the *L* list because they have a link-path that is different from the `sample page`. In line 03, four groups are created ($G_1$, $G_2$, $G_3$, and $G_4$), each of which contains the pages of the *L* list (except the `sample page`) with the same $strongsim_{URL}$. In line 04, the groups are sorted in decreasing order of $strongsim_{URL}$. The $G_1$ group has the pages with the largest $strongsim_{URL}$, while the $G_4$ group has the pages with the lowest $strongsim_{URL}$. In line 05, all the pages from the $G_1$ group (3.1 and 3.3) are added to the *rs* set. In line 06, the minimal HTML similarity (called $minsim_{html}$), between the `sample page` and the pages from the $G_1$ group, is computed. In this example, the $minsim_{html}$ is the HTML similarity between the `sample page` and Page 3.3 (0.75). The $minsim_{html}$ represents the HTML similarity threshold. In line 07, the $G_1$ group is removed from the *G* list. The first iteration of the "*for*" (lines 08 to 14) analyzes the $G_2$ group since it has the largest URL similarity after $G_1$ has been removed. As there is a page in $G_2$ that satisfies the HTML similarity threshold, all the pages from $G_2$ are added to the *rs* set. The second iteration of the "*for*" (lines 08 to 14) analyzes the $G_3$ group. Since there are no pages in $G_3$ that satisfy the HTML similarity threshold, all the pages of this group (3.7) are discarded. All the remaining groups ($G_4$) are discarded too. The URL similarity of $G_3$ (0.62) represents the URL similarity threshold because our method assumes that the entity-pages of the same type have a larger score of URL similarity and HTML similarity than the other pages. Finally, the *rs* set (with Pages 3.1, 3.2, 3.3, 3.4, 3.5) is returned in line 15.

## 5 EXPERIMENTS

This section presents the experiments that were conducted to evaluate the effectiveness and efficiency of **SSUP** by comparing it with two baselines. Section 5.1 defines metrics, datasets, and baselines. Section 5.2 shows the experiments that were carried out to find the best parameter

configuration of each method. Section 5.3 presents the experiments that compare our method with the baselines. Finally, Section 6 discusses the main cases of failure.

### 5.1 Experimental Setup

**Metrics.** The evaluation of the effectiveness measures the ability of the method to return the relevant pages of a website, that is, the ability of the method to achieve its goal. A page is `relevant` if it is an entity-page of the same type as the `sample page`. A page is `irrelevant` if (1) it is not an entity-page or (2) it is an entity-page of a type different from the `sample page`. Standard metrics were used such as recall, precision, and F1. Recall is the fraction of the relevant pages in the website that are retrieved. Precision is the fraction of the retrieved pages that are relevant. F1 is the harmonic mean of recall and precision that assigns the same weight to the two metrics. These metrics were computed in the following way:

$$Recall = \frac{|Rel \cap Ret|}{|Rel|}, Precision = \frac{|Rel \cap Ret|}{|Ret|}, F1 = \frac{2 \times Precision \times Recall}{Precision + Recall},$$

where $Rel$ is the set of relevant pages on the website, $Ret$ is the set of pages retrieved by the method, and $|S|$ denotes the number of elements of a set $S$.

The evaluation of the efficiency allows assessing if a given method (to discover entity-pages) can be successfully applied in real-world scenarios, which is our concern. This evaluation is important because an entity-page discovery method must be rerun periodically for two reasons. First, the data on the entity-pages can change over time. Second, new entity-pages can be made available on the website over time. The adopted metrics to assess the efficiency were the number of downloaded pages and the processing time. The number of downloaded pages indicates the number of pages that have had their content accessed during the entity-page discovery task. The method must download these pages locally to analyze their content. A method that downloads several pages could interfere with the normal operation of the website [17], which is not desirable. The processing time indicates the total time spent running the method to discover entity-pages. We downloaded the pages from the websites in advance to avoid network interference in the processing time.

The Student's t-test [1] with the standard significance level ($\alpha = 0.05$) was employed to determine if the average values of two distributions are statistically different. When the $p$-value returned by the t-test is lower than $\alpha$, there is a significant difference between the results of the methods. With regard to effectiveness, the best outcome is obtained from the method with the highest average distribution, while in the case of efficiency, the best outcome is obtained from the method with the lowest average distribution.

**Datasets.** The evaluation was carried out over 38 real-world websites. For each website $s_x$, (1) the set of pages in the $s_x$ website were collected, (2) an entity type $t_y$ (e.g., driver in a website about a car racing championship) was defined, and (3) the ground truth (the set of entity-pages of the $t_y$ type that are in the $s_x$ website) was obtained through hand-crafted rules. These websites were grouped into three datasets (see Appendix A for further details[6]):

- `Multiple types`: comprises 10 websites with entity-pages of different types
- `Drivers`: contains 10 websites with entity-pages about drivers
- `Council members`: has 18 websites with entity-pages about council members. The official websites of the city council of the 26 Brazilian state capitals were analyzed. We excluded four websites because they did not have an entity-page for each council member, three websites

[6]The datasets are available at https://github.com/edimarmanica/ssup/.

because all the entity-pages were internal frames of a single page, and one website because it did not allow crawling its pages.

The *MIT EECS* website includes entity-pages about professors, administrative staff, and students. These pages were regarded as being of the same type (*person*) because they share attributes such as *position*, *phone*, and *e-mail*. The *Stanford* website includes entity-pages about *professors* and *administrative personnel*. These pages were regarded as being of the same type (*employee*) because they share attributes such as *education*, *department*, *e-mail*, and *phone*. The *F1*, *F3*, *GP2*, and *Motor GP* websites include entity-pages about active drivers and nonactive drivers. These pages were regarded as being of the same type (*driver*) because they share the following attributes: *team*, *nationality*, *date of birth*, *height*, *weight*, and *homepage*.

These datasets were chosen because they meet the following requirements: (1) the websites should have a subset of pages, where each page publishes data representing an entity of a particular type, since **SSUP** and the baselines aim to find entity-pages; (2) the entity-pages should be dynamically generated by populating fixed HTML templates with content from a DBMS because this article focuses on template-based entity-pages; (3) all the pages of the websites should be available to evaluate the methods because they navigate through the websites to find the entity-pages; and (4) the pages need to be labeled as relevant or irrelevant to compute the metrics of effectiveness.

**Baselines. SSUP** was compared with two baselines: *INDESIT* [4] and *Growing Parallel Paths* (here called *GPP*) [32]. We chose these baselines because they (1) are state-of-the-art methods for entity-page discovery, (2) are automatic, (3) have a crawling strategy that is aimed at downloading the minimum number of pages of the website to find the entity-pages, (4) do not require the entity-pages of all the websites to have a specific HTML markup (e.g., *table* or *ul*), (5) do not depend on a human-compiled encyclopedia (e.g., Wikipedia), and (6) only require an example of an entity-page as input.

**Metodology. SSUP** and the baselines require an entity-page (called `sample page`) as input. To avoid that the chosen `sample page` biased the results, we executed the methods $n$[7] times for each website, by using a different `sample page` in each execution. The shown results correspond to the average results of all the executions.

The experiments are divided into two groups: (1) *calibration*: aims to define the best parameter configuration for the **SSUP** method and the baselines, and (2) *evaluation*, aims to compare the **SSUP** method with the baselines. For each website, 10% of the entity-pages were randomly selected to be used as `sample pages` in the *calibration* experiments and 10% in the *evaluation* experiments. The intersection between the set of `sample pages` used in the *evaluation* experiments and the set of `sample pages` employed in the *calibration* experiments is an empty set.

The experiments were carried out on a computer with Intel Core 2 Quad 2.66GHz, operating system Ubuntu 14.04, 8GB of main memory, and 5TB of disk space. *INDESIT* was implemented as described by Blanco et al. [4]. **SSUP** and *INDESIT* were developed in Java using the JSoup library[8] to handle the HTML of the pages. The authors provided the module of *GPP* that extracts the visual features of the pages. This module is a Java code and uses the API Selenium Web Driver.[9]

### 5.2 Calibration

This subsection shows the *calibration* experiments.

[7] $n$ is equal to 10% of the number of relevant pages on the website.
[8] Available at: http://jsoup.org/. Accessed: 02/06/2019.
[9] Available at: http://docs.seleniumhq.org/projects/webdriver/. Accessed: 02/06/2019.

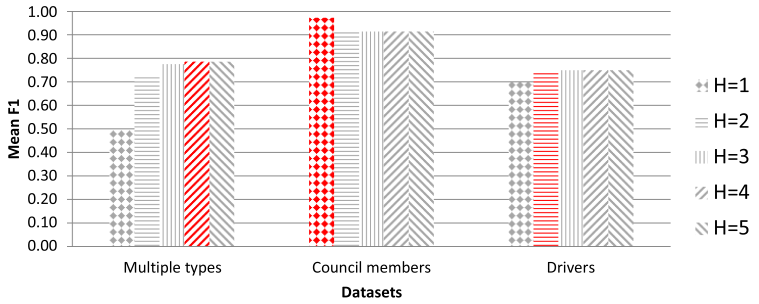



Fig. 6. Mean F1 of **SSUP** for each parameter configuration.

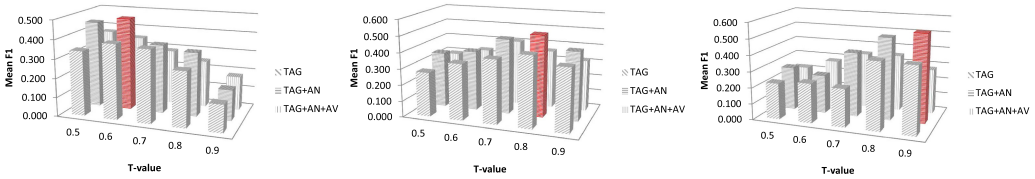
*5.2.1 Best Parameter Configuration for **SSUP**.* The purpose of this experiment is to answer the following question: *what is the best parameter configuration for* **SSUP***?* **SSUP** only has one parameter, $H$, which defines the estimated height of the entity-tree. We tested the values 1, 2, ..., 5 and confirmed manually that no site in our datasets has an entity-tree with a greater height than 5.

The best $H$-value was defined through the efficacy, which was measured by the F1 metric. The higher the F1 value, the greater the efficacy. When two or more $H$-values produced the same F1 value, we chose the smallest one because it has the lowest processing time. Figure 6 presents the mean F1 of **SSUP** for each $H$-value in each dataset. The best $H$-value was four for *multiple types*, one for *council members*, and two for *drivers*. These values are highlighted in red in the figure and correspond to the $H$-values that we have used in all the *evaluation* experiments.

The best $H$-value in the *multiple types* dataset was higher than in the other datasets because the websites in this dataset have the largest number of entity-pages. The larger the number of entity-pages a website has, the greater the height of its entity-tree. In the *multiple types* and *drivers* datasets, an $H$-value greater than the actual height of the entity-tree did not affect the efficacy of **SSUP** because the irrelevant pages were correctly filtered out. In the *council members* dataset, an $H$-value greater than the actual height of the entity-tree affected the efficacy of **SSUP** because four websites in this dataset have irrelevant pages (pages about news, sections, etc.) with HTML and URL similar to the relevant pages, and thus **SSUP** did not correctly filter out the irrelevant pages. For example, in the *Campo Grande* website, the URL of the pages with news only differs from the URL of the pages about council members by the value of one parameter, and their HTML structures are also similar.

*5.2.2 Best Parameter Configuration for INDESIT.* This experiment aims to answer the following question: *what is the best parameter configuration for INDESIT? INDESIT* has two parameters, which define (1) $T$: the HTML similarity threshold between two entity-pages, which varies from 0 to 1, and (2) *notation*: the components of a *link-path*. The following $T$-values were tested: 0.5, 0.6, ..., 0.9. The following notations were tested: (1) *TAG*: the *link-path* is only formed by tags, (i) *TAG+AN*: the *link-path* is formed by tags and attribute names, and (3) *TAG+AN+AV*, the *link-path* is formed by tags, attribute names, and attribute values.

The best parameter configuration was defined through the efficacy, which was measured by the F1 metric. Figure 7 shows the mean F1 of each combination of $T$-values and notations in each dataset. The best parameter configuration was (1) $T = 0.6$ and $notation = TAG + AN$ in the *multiple types* dataset, (2) $T = 0.8$ and $notation = TAG + AN$ in the *council members* dataset; and (3) $T = 0.9$ and $notation = TAG + AN$ in the *drivers* dataset. These configurations are highlighted in red in the figure and correspond to the configurations we have used in the *evaluation* experiments.




(a) Multiple types (b) Council members (c) Drivers

Fig. 7. Mean F1 of *INDESIT* for each parameter configuration.

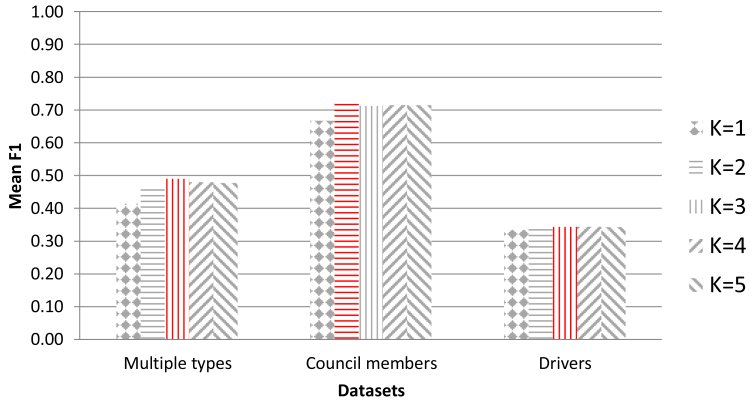



Fig. 8. Mean F1 of *GPP* for each parameter configuration.

*INDESIT* produced the highest F1 value with the greatest $T$-value in the *drivers* dataset because the entity-pages in most sites of this dataset follow a strict template. The best notation in the three datasets was the one that includes tags and attribute names. This notation has a discriminatory capacity (i.e., a capacity to distinguish relevant pages from irrelevant pages) that is higher than the notation that only uses tags since it is able to distinguish, for example, a tag *TD* with the *colspan* attribute from a tag *TD* with the *rowspan* attribute. The notation that includes tags, attribute names, and attribute values has the highest discriminatory capacity. However, this notation seriously affects the recall in sites that have an entity identifier in the value of an attribute. For example, all the entity-pages in the *Olympic-A* site have an entity identifier in the value of the *class* attribute of the *body* tag (<*body class* = *"country PageID30785 klp"*>). When an account is taken of the notation using tags, attribute names, and attribute values, the entity-pages of the same type in the *Olympic-A* website do not share any identical link-paths and, hence, their HTML similarity is zero.

*5.2.3 Best Parameter Configuration for GPP.* This experiment aims to answer the following question: *what is the best parameter configuration for GPP? GPP* has only one parameter, K, which defines the number of iterations. We tested 1, 2, . . . , 5 iterations, while the experiments in the original paper [32] only evaluated *GPP* with five iterations.

The best $K$-value was defined through the efficacy, which was measured by the F1 metric. When two or more $K$-values produced the same F1 value, we chose the smallest one because it has the lowest processing time. Figure 8 presents the mean F1 of *GPP* for each $K$-value in each dataset. The best $K$-value was three in the *multiple types* dataset, two in the *council members* dataset, and three in the *drivers* dataset. These values are highlighted in red in the figure and correspond to the $K$-values that we have used in all the *evaluation* experiments.

Table 4. Recall and Precision of *INDESIT* and **SSUP**

| Datasets | Recall | | | Precision | | |
|---|---|---|---|---|---|---|
| | INDESIT | SSUP | % | INDESIT | SSUP | % |
| Multiple types | 0.52 | 0.82 | ↑ **57** | 0.60 | 0.93 | ↑ 54 |
| Council members | 0.73 | 0.97 | ↑ 32 | 0.57 | 0.99 | ↑ **73** |
| Drivers | 0.53 | 0.68 | ↑ 28 | 0.76 | 0.90 | ↑ 19 |
| Mean | 0.62 | 0.85 | ↑ 37 | 0.63 | 0.95 | ↑ 51 |

The best $K$-value in the *multiple types* and *drivers* datasets was higher than the best $K$-value in the *council members* dataset because some sites in these datasets have links (on the main page to some entity-pages) that work as shortcuts. For example, a site about a car racing championship has a link on the main page to the entity-pages about the top three drivers. This path is the first ($K = 1$) found in the breadth-first search. Other paths ($K$) must be analyzed to find all the entity-pages about drivers.

### 5.3 Evaluation

This subsection shows the *evaluation* experiments. **SSUP** is compared with only one baseline in each experiment to provide an in-depth analysis of the results, which describes the impact of the features used by the method in terms of efficacy and efficiency.

*5.3.1* ***SSUP** versus INDESIT.* This experiment compares **SSUP** with *INDESIT* to answer the following questions: (i) *which method is more effective?* and (ii) *which method is more efficient?*

Effectiveness was measured through the recall and precision metrics. Table 4 presents the recall and precision of the methods in each dataset and the overall mean from an examination of the 38 websites. The percent change (%) is also shown and the best results are highlighted in bold.

On average, **SSUP** and *INDESIT* obtained a recall of 0.85 and 0.62, respectively. The mean gain of **SSUP** in terms of recall was 37%. The t-test shows that this increase is statistically significant because the $p$-value ($2.23 \times 10^{-03}$) is less than the standard level of significance ($\alpha = 0.05$). This gain occurred because **SSUP** automatically determines the HTML similarity threshold in each website. The HTML similarity threshold in *INDESIT* is defined by the user. We used the best $T$-value for each dataset, but this value is not appropriate for all the websites in the dataset. This is because some relevant pages do not satisfy the threshold and are erroneously filtered out by *INDESIT*. With regard to recall, the highest gain of **SSUP** was in the *multiple types* dataset (57%) because this dataset has websites from different entity types and, hence, the use of a fixed HTML similarity threshold (as in the case of *INDESIT*) is not effective.

On average, **SSUP** and *INDESIT* achieved a precision of 0.95 and 0.63, respectively. The mean gain of **SSUP** in terms of precision was 51%. The t-test shows that this gain is statistically significant because the $p$-value ($1.24 \times 10^{-06}$) is less than the standard level of significance ($\alpha = 0.05$). This gain was obtained because **SSUP** combines URL features with HTML features to distinguish relevant pages from irrelevant pages. *INDESIT* only uses HTML features, which are not enough to separate relevant pages from irrelevant pages in determined websites. With regard to precision, the highest gain of **SSUP** was in the *council members* dataset (73%) because the entity-pages in most websites of this dataset have a URL term (e.g., "*member*") that is only found in the relevant pages. In these websites, the URL similarity plays a crucial role in distinguishing the relevant pages from the irrelevant pages.

Efficiency was measured by the number of downloaded pages and processing time metrics. Table 5 presents the mean processing time (in seconds) and the mean number of downloaded pages of

Table 5. Processing Time and Number of Downloaded Pages of *INDESIT* and **SSUP**

| | INDESIT | SSUP | % |
|---|---|---|---|
| Time (seconds) | 3.98 | 0.66 | ↓ 83 |
| Nr. downloads | 5.48 | 1.73 | ↓ 68 |

Table 6. Recall and Precision of *GPP* and **SSUP**

| Datasets | Recall | | | Precision | | |
|---|---|---|---|---|---|---|
| | GPP | SSUP | % | GPP | SSUP | % |
| Multiple types | 0.56 | 0.82 | ↑ **47** | 0.60 | 0.93 | ↑ 54 |
| Council members | 0.82 | 0.97 | ↑ 18 | 0.72 | 0.99 | ↑ 37 |
| Drivers | 0.52 | 0.68 | ↑ 29 | 0.27 | 0.90 | ↑ **236** |
| Mean | 0.67 | 0.85 | ↑ 27 | 0.57 | 0.95 | ↑ 66 |

the methods obtained from the 38 websites. This table shows the relative values; i.e., the absolute values were divided by the number of relevant pages that the method retrieved.

On average, the processing times of **SSUP** and *INDESIT* per retrieved relevant page were 0.66 and 3.98 seconds, respectively. The processing time of **SSUP** was 83% lower than *INDESIT*. The t-test shows that this difference is not statistically significant because the $p$-value (≈0.36) is greater than the standard level of significance ($\alpha = 0.05$). **SSUP** obtained a processing time lower than *INDESIT* in the websites where the most irrelevant pages can be filtered out by URL features, since the processing of the URL features is faster than the processing of the HTML features. In other websites, *INDESIT* obtained a processing time lower than **SSUP** because *INDESIT* has a strategy to minimize the number of pages that have their HTML accessed, which is based on the link-path of the anchor nodes that point to the pages.

On average, **SSUP** and *INDESIT* downloaded 1.73 and 5.48 pages per retrieved relevant page, respectively. The number of downloaded pages by **SSUP** was 68% lower than by *INDESIT*. The t-test shows that this difference is not statistically significant because the $p$-value (≈0.39) is greater than the level of significance ($\alpha = 0.05$). **SSUP** downloaded fewer pages than *INDESIT* in the sites where the most irrelevant pages can be filtered out by URL features, since in this case, **SSUP** does not need to access the content of the pages. In other websites, *INDESIT* downloaded fewer pages than **SSUP** because *INDESIT* has a strategy to minimize the number of downloaded pages.

The results show that **SSUP** is more effective than *INDESIT* because **SSUP** was able to (1) retrieve more relevant pages than *INDESIT* (recall) and (2) retrieve fewer irrelevant pages than *INDESIT* (precision). **SSUP** is as efficient as *INDESIT* because there was no statistically significant difference between the two methods with regard to the processing time and number of downloaded pages.

*5.3.2* ***SSUP*** *versus GPP.* This experiment compares **SSUP** with *GPP* to answer the following questions: (i) *which method is more effective?* and (ii) *which method is more efficient?*

Effectiveness was measured through the recall and precision metrics. Table 6 presents the recall and precision of the methods in each dataset and the overall mean when examining the 38 websites. The percent change (%) is also shown and the best results are highlighted in bold.

On average, **SSUP** and *GPP* achieved a recall of 0.85 and 0.67, respectively. The mean gain of **SSUP** with regard to recall was 27%. The t-test shows that this increase is statistically significant because the $p$-value ($4.18 \times 10^{-03}$) is less than the standard level of significance. This gain was

Table 7. Processing Time and Number of Downloaded Pages of *GPP* and **SSUP**

| | GPP | SSUP | % |
|---|---|---|---|
| Time (seconds) | 14.73 | 0.66 | ↓ 96 |
| Nr. downloads | 0.50 | 1.73 | ↑ 245 |

achieved because (in contrast with *GPP*) **SSUP** does not require the links to the entity-pages to be in web lists. The recall of *GPP* is affected in the websites where the path from the main page to the entity-pages (1) does not consist of web lists because this is a requirement of *GPP* or (2) includes a nested list because *GPP* is not able to extract the nested lists. With regard to recall, the greatest gain of **SSUP** was in the *multiple types* dataset (47%) because this dataset has the highest number of websites where the path from the main page to the entity-pages does not consist of web lists.

On average, **SSUP** and *GPP* obtained a precision of 0.95 and 0.57, respectively. The mean gain of **SSUP** in terms of precision was 66%. The t-test shows that this gain is statistically significant because the $p$-value ($1.37 \times 10^{-08}$) is less than the level of significance ($\alpha = 0.05$). This gain was obtained because **SSUP** analyzes the features of the index-pages and entity-pages, while *GPP* only evaluates features of the index-pages. Irrelevant pages with index-pages that have similar features to the index-page of the `sample page` are returned by *GPP*. This situation was found with pages that publish photos, videos, and so forth. With regard to precision, the greatest gain of **SSUP** was in the *drivers* dataset (236%) because most websites in this dataset have a strict template that makes index-pages of entity-pages about drivers and index-pages of pages with photos have similar features. Unlike *GPP*, **SSUP** is not influenced by this situation.

Efficiency was measured by the number of downloaded pages and processing time metrics. Table 7 shows the mean processing time (in seconds) and the average number of downloaded pages of the methods by examining the 38 websites. This table shows the relative values; that is, the absolute values were divided by the number of relevant pages that the method retrieved.

On average, the processing times of **SSUP** and *GPP* per retrieved relevant page were 0.66 and 14.73 seconds, respectively. The processing time of **SSUP** was 96% lower than *GPP*. The t-test shows that this difference is statistically significant because the $p$-value ($8.93 \times 10^{-04}$) is less than the standard level of significance. **SSUP** had a processing time lower than *GPP* because (unlike *GPP*) **SSUP** does not need to render the pages in a web browser. The page rendering requires the browser to load all the images, CSS, and Javascripts and, hence, significantly increases the processing time.

On average, **SSUP** and *GPP* downloaded 1.73 and 0.50 pages per retrieved relevant page, respectively. The t-test shows that this difference is not statistically significant because the $p$-value ($\approx 0.87$) is greater than the standard level of significance ($\alpha = 0.05$). On average, *GPP* downloaded fewer pages than **SSUP** because *GPP* is able to filter the pages by means of the features in their index-pages. On average, *GPP* downloaded fewer pages than the number of retrieved relevant pages because its filter strategy does not require the entity-pages to be downloaded.

The results show that **SSUP** is more effective than *GPP* because **SSUP** was able to (1) retrieve more relevant pages (recall) than *GPP* and (2) retrieve fewer irrelevant pages (precision) than *GPP*. **SSUP** is more efficient than *GPP* because (1) **SSUP** had a processing time lower than *GPP* and (2) there was no statistically significant difference between the two methods with regard to the number of downloaded pages.

## 6 DISCUSSION

In this subsections, we analyze the cases of failure in **SSUP**. This analysis might be very useful for developers of new methods to discover entity-pages because it shows the difficulties of handling

Table 8. Sites Whose Entity-Pages Have Two-Token URLs

| Dataset | Site ID | URL of an Entity-Page |
|---|---|---|
| Multiple types | Olympic-S | http://www.olympic.org/snowboard |
| Drivers | Champ | http://champcar-ws.com/james_hinchcliffe |
| Council member | Fortaleza | http://www.cmfor.ce.gov.br/acrisiosena1.html |

their particular features. We have detected five main cases of failure, which are described in the following subsections, and we point the way to detect the fail or how to solve it.

### 6.1 Two-Token URLs

In some websites, the URL of the entity-pages is very short, with only two tokens. In these websites, the precision of **SSUP** is affected because relevant pages should be distinguished from irrelevant pages only by HTML features. This situation took place in the following websites: *Olympic-S*, *Champ*, and *Fortaleza*. Table 8 shows the URL of an entity-page of each one of these websites.

It is important to note that this situation occurred in less than 10% of the websites in the datasets. This situation can be identified (1) manually, where a user analyses the URL of the `sample page` and verifies the number of tokens, or (2) automatically, where a software tokenizes, in accordance with **SSUP**, the URL of the `sample page` and counts the number of tokens. When the entity-pages of the website have two-token URLs, it is possible to use a strategy that explores HTML and content features (e.g., [12, 16]) or a method that combines HTML and visual features (e.g., [32]).

### 6.2 Handcrafted Entity-Pages

There are websites in which the entity-pages are created manually without following a template. In this case, the precision of our method is affected because **SSUP** determines a quite low HTML similarity threshold, and then irrelevant pages also satisfy the threshold. This kind of page is not within the scope of this study, but the *MIT EECS* website has an optional attribute *biography* where it is possible to add textual content and HTML tags; that is, the user can change the HTML structure of the page. The precision of **SSUP** was not affected in this website because **SSUP** used URL features to distinguish relevant pages from irrelevant pages.

This case usually occurs with entity-pages about people. For example, Figure 9 illustrates the fragment of two handcrafted entity-pages about Computer Science professors on the website of the Cornell University: (a) Dr. Rachit Agarwal[10] and (b) Dr. Lorenzo Alvisi.[11] These entity-pages do not follow a template, and then they have a HTML similarity close to zero. Handcrafted entity-pages can be identified (1) manually, by a user analyzing two entity-pages, because the presentation of handcrafted entity-pages is strongly different, or (2) automatically, computing the HTML similarity between two entity-pages and verifying if the score is less than 0.5. When the entity-pages do not follow a template, it is possible to use a strategy that explores content features (e.g., [12, 16]).

### 6.3 Different Link-Paths

Relevant pages, in some websites, are pointed by the index-pages through different link-paths. On these websites, the recall of **SSUP** is affected because relevant pages with a different link-path from the `sample page` are not retrieved. This was the situation in the following websites: *Olympic - S*

[10] Available at: http://www.cs.cornell.edu/~ragarwal/. Accessed: 4/15/2018.
[11] Available at: http://www.cs.cornell.edu/lorenzo/. Accessed: 4/15/2018.


(a) (b)

Fig. 9. Two handcrafted entity-pages about Computer Science professors on the website of Cornell University: (a) Dr. Rachit Agarwal and (b) Dr. Lorenzo Alvisi. These entity-pages do not follow a template, and then they have a HTML similarity close to zero.

Table 9. Sites That Spread the Relevant Pages in Different Logical Hierarchies

| Dataset | Site ID | Entity-Type of Interest | Logical Hierarchies |
|---|---|---|---|
| Multiple Types | Stanford EE | Employee | Professor, administrative personnel |
| Drivers | F1 | Driver | Active driver, nonactive driver |
| Drivers | F3 | Driver | Active driver, nonactive driver |
| Drivers | GP2 | Driver | Active driver, nonactive driver |
| Drivers | Motor GP | Driver | Active driver, nonactive driver |

and *Recife*. For example, in the *Recife* website, the links to all the relevant pages are in the same HTML table. However, some of them have the tag "*p*" in their link-path and others do not.

We have analyzed the occurrence of different DOM-paths in template-based entity-pages in our previous study [22]. A total of 74,247 template-based entity-pages from 120 websites were analyzed. Only 15% of the entity-pages publish the value of an attribute with a DOM-path different from the most frequent DOM-path for the attribute in the website. We observed that this percentage is even lower when considering the link-paths in the index-pages that point to the entity-pages. About 5% of the websites in the datasets used in the experiments presented this case. We recommend to use **SSUP** in this case, because the impact on efficacy is low.

### 6.4 Different Logical Hierarchies

Some websites spread the relevant pages in different logical hierarchies of topics. In these websites, the recall of **SSUP** is affected because our method can only find entity-pages that are in the logical hierarchy of topics containing the `sample page`. This situation took place in the following websites: *F1*, *F3*, *GP2*, *Motor GP*, and *Stanford EE*. Table 9 shows the websites that have more than one logical hierarchy for the entity-type of interest defined in the experiments.

We highlighted that this case of failure occurred in less than 15% of the websites in the datasets. This situation can be identified (1) *a priori*, where the user navigates through the website and verifies that the entities of interest are scattered in different logical hierarchies of topics, or (2) *a posteriori*, where the user analyzes the results and verifies that only entities of a subtype of the type of interest were returned. When the user identifies this situation, he or she must provide a

`sample page` for each logical hierarchy of topics. We found interest-type entities scattered in at most two logical hierarchies. In this case, the user must provide only more one `sample page`.

### 6.5 Client-Side JavaScript

There are websites that load the attribute values using Client-Side JavaScript. These attribute values can only be captured after the browser loads, renders the pages, and executes the JavaScript. In these websites, the recall of **SSUP** is not affected if the hyperlinks are not built by JavaScript because our method identifies the relevant pages based on the URL and HTML features; that is, we do not use content features. If the hyperlinks to the entity-pages are built by JavaScript, an approach that renders the webpages is required (e.g., [32]). This page rendering can be done by the API Selenium Web Driver and it significantly increases the processing time.

## 7 CONCLUSION

In this article, a new method has been devised to discover entity-pages, called **SSUP**. Given an example of an entity-page, our method returns the set of entity-pages of the same type that belong to the website. For example, given the page that describes *Fernando Alonso* on the Formula 1 website, **SSUP** discovers the set of entity-pages about drivers that belong to this website. **SSUP** has two key stages. The first (*Identification of the Page-Path*) finds the path starting from the root until the example of entity-page in the entity-tree. The second (*Traversal of the Entity-Tree*) traverses the entity-tree and returns its entity-pages. The novelty of our method is that it combines URL and HTML features in a way that allows the URL terms to have different weights depending on their discriminatory capacity. We specify a URL similarity function and reuse an HTML similarity function. **SSUP** determines the URL and HTML similarity thresholds on each site by assuming that "entity-pages of the same type have greater URL similarity and HTML similarity than other pages."

We carried out experiments on 38 real-world websites and the results of **SSUP** were compared with two baselines (*INDESIT* and *GPP*). As a result, we showed that **SSUP** outperformed the baselines regarding effectiveness (recall and precision) while keeping its level of efficiency (number of downloaded pages and processing time). **SSUP** retrieved more relevant pages than the baselines (recall) and was more effective in filtering out noise-pages (precision). **SSUP** did not increase, significantly, either the number of downloaded pages or the processing time. We also conducted an analysis of the main cases of failure in **SSUP**. This analysis describes the datasets where they occurred, the behavior of **SSUP** when they appeared, and possible solutions. This analysis might be of considerable value for the development of new methods because it shows the difficulties of discovering entity-pages.

We recommend that future work should include the following: (1) extracting the data published in the entity-pages retrieved by **SSUP** using a web scraper, (2) storing the data in light of factors such as the uncertainty and temporality, (3) providing direct answers to web searches about less common entities from the data extracted from the template-based entity-pages, (4) carrying out experiments to verify if data extraction from template-based entity-pages (combining **SSUP** with a web scraper) leads to better entity information to populate knowledge bases than free-text extraction (e.g., [2, 15, 24, 36]), and (5) extending the **SSUP** to cluster web pages based on both URL and HTML features.

## APPENDIX

## A DETAILS ABOUT THE DATASETS

| Dataset | Site ID | Main page | Entity type | NP |
|---|---|---|---|---|
| Multiple types | Fifa | http://www.fifa.com/ | Association | 209 |
| | Guitar | http://www.guitaretab.com/ | Band | 32318 |
| | Inter | http://www.inter.it/en/hp/ | Soccer player | 28 |
| | MIT EECS | http://www.eecs.mit.edu/ | Person | 1032 |
| | Olympic-A | http://www.olympic.org/ | Association | 204 |
| | Olympic-S | http://www.olympic.org/ | Sport | 56 |
| | Pgfoundry | http://pgfoundry.org/ | Software project | 382 |
| | Senado | http://www.senado.gov.br/ | Senator | 121 |
| | Stanford EE | http://engineering.stanford.edu/ | Employee | 473 |
| | Supercar | http://www.supercarsite.net/ | Car | 512 |
| Drivers | Champ | http://champcar-ws.com/ | Driver | 20 |
| | F Truck | http://www.formulatruck.com.br/ | Driver | 26 |
| | F1 | http://www.formula1.com/ | Driver | 22 |
| | F3 | http://www.formula3.co/ | Driver | 31 |
| | GP2 | http://www.gp2series.com/ | Driver | 34 |
| | Indycar | http://www.indycar.com/ | Driver | 39 |
| | Motor GP | http://www.motogp.com/ | Driver | 1288 |
| | Nascar | http://www.nascar.com/ | Driver | 349 |
| | Stock car | http://www.stockcar.com.br/ | Driver | 34 |
| | WRC | http://www.wrc.com/ | Driver | 133 |
| Council members | Belém | http://cmb.pa.gov.br/ | Council member | 22 |
| | Belo Horizonte | http://www.cmbh.mg.gov.br/ | Council member | 42 |
| | Campo Grande | http://camara.ms.gov.br/ | Council member | 29 |
| | Cuiabá | http://camaracba.mt.gov.br/ | Council member | 25 |
| | Curitiba | http://www.cmc.pr.gov.br/ | Council member | 38 |
| | Florianópolis | http://cmf.sc.gov.br/ | Council member | 23 |
| | Fortaleza | http://cmfor.ce.gov.br/ | Council member | 50 |
| | Goiânia | http://www.camara.go.gov.br/ | Council member | 35 |
| | João Pessoa | http://cmjp.pb.gov.br/ | Council member | 27 |
| | Macapá | http://cmm.ap.gov.br/ | Council member | 23 |
| | Natal | http://cmnat.rn.gov.br/ | Council member | 29 |
| | Porto Velho | http://portovelho.ro.leg.br/ | Council member | 21 |
| | Recife | http://www.recife.pe.leg.br/ | Council member | 39 |
| | Rio Branco | http://riobranco.ac.leg.br/ | Council member | 17 |
| | Rio de Janeiro | http://www.camara.rj.gov.br/ | Council member | 51 |
| | Salvador | http://www.cms.ba.gov.br/ | Council member | 43 |
| | Teresina | http://teresina.pi.leg.br/ | Council member | 29 |
| | Vitória | http://www3.cmv.es.gov.br/ | Council member | 15 |

*Note*: NP = number of entity-pages.

## REFERENCES


[1] T. W. Anderson and J. D. Finn. 1996. *The New Statistical Analysis of Data*. Springer.

[2] Akari Asai, Sara Evensen, Behzad Golshan, Alon Y. Halevy, Vivian Li, Andrei Lopatenko, Daniela Stepanov, Yoshihiko Suhara, Wang-Chiew Tan, and Yinzhan Xu. 2018. HappyDB: A Corpus of 100, 000 crowdsourced happy moments. In *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC'18)*.

[3] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. 2011. *Modern Information Retrieval - The Concepts and Technology Behind Search*.2nd ed. Pearson Education, Harlow, England. Retrieved from http://www.mir2ed.org/.

[4] Lorenzo Blanco, Valter Crescenzi, and Paolo Merialdo. 2005. Efficiently locating collections of web pages to wrap. In *International Conference on Web Information Systems and Technologies (WEBIST'05)*. INSTICC Press, Miami, FL, 247–254.

[5] Lorenzo Blanco, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. 2008. Supporting the automatic construction of entity aware search engines. In *International Workshop on Web Information and Data Management (WIDM'08)*, Chee Yong Chan and Neoklis Polyzotis (Eds.). ACM, 149–156.

[6] Lorenzo Blanco, Nilesh Dalvi, and Ashwin Machanavajjhala. 2011. Highly efficient algorithms for structural clustering of large websites. In *Proceedings of the 20th International Conference on World Wide Web (WWW'11)*. ACM, New York, NY, 437–446.

[7] Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. 2013. Extraction and integration of partially overlapping web sources. *PVLDB* 6, 10 (2013), 805–816.

[8] Andrew Carlson and Charles Schafer. 2008. Bootstrapping information extraction from semi-structured web pages. In *Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I (ECML PKDD'08)*. Springer-Verlag, Berlin, 195–210.

[9] Eric Crestan and Patrick Pantel. 2011. Web-scale table census and classification. In *Proceedings of the 4th ACM International Conference on Web Search and Data Mining (WSDM'11)*. ACM, New York, NY, 545–554.

[10] Fabio Fumarola, Tim Weninger, Rick Barber, Donato Malerba, and Jiawei Han. 2011. HyLiEn: A hybrid approach to general list extraction on the web. In *Proceedings of the 20th International Conference Companion on World Wide Web (WWW'11)*. ACM, New York, NY, 35–36.

[11] Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, Christian Schallhart, and Cheng Wang. 2014. DIADEM: Thousands of websites to a single database. *PVLDB* 7, 14 (2014), 1845–1856.

[12] Sujatha Das Gollapalli, Cornelia Caragea, Prasenjit Mitra, and C. Lee Giles. 2015. Improving researcher homepage classification with unlabeled data. *ACM Transactions on the Web* 9, 4, Article 17 (Oct. 2015), 32 pages.

[13] Peter D. Grünwald. 2007. *The Minimum Description Length Principle*. MIT Press, London, England.

[14] Pankaj Gulhane, Rajeev Rastogi, Srinivasan H. Sengamedu, and Ashwin Tengli. 2010. Exploiting content redundancy for web information extraction. *PVLDB* 3, 1 (2010), 578–587.

[15] Alon Y. Halevy. 2017. Technical perspective: Building knowledge bases from messy data. *Communications of the ACM* 60, 5 (2017), 92.

[16] Yeye He, Dong Xin, Venkatesh Ganti, Sriram Rajaraman, and Nirav Shah. 2013. Crawling deep web entity pages. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining (WSDM'13)*. ACM, New York, NY, 355–364.

[17] Inma Hernández, Carlos R. Rivero, David Ruiz, and Rafael Corchuelo. 2016. CALA: ClAssifying links automatically based on their URL. *Journal of Systems and Software* 115 (2016), 130–143.

[18] Djoerd Hiemstra. 2001. *Using Language Models for Information Retrieval*. Ph.D. Dissertation. Centre for Telematics and Information Technology, University of Twente, Enschede.

[19] Rianne Kaptein, Pavel Serdyukov, Arjen De Vries, and Jaap Kamps. 2010. Entity ranking using wikipedia as a pivot. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM'10)*. ACM, New York, NY, 69–78.

[20] Cindy Xide Lin, Bo Zhao, Tim Weninger, Jiawei Han, and Bing Liu. 2010. Entity relation discovery from web tables and links. In *Proceedings of the 19th International Conference on World Wide Web (WWW'10)*. 1145–1146.

[21] Edimar Manica, Carina F. Dorneles, and Renata Galante. 2017. Orion: A cypher-based web data extractor. In *Database and Expert Systems Applications - 28th International Conference (DEXA'17), Proceedings, Part I*. 275–289.

[22] Edimar Manica, Carina F. Dorneles, and Renata Galante. 2017. R-Extractor: A method for data extraction from template-based entity-pages. In *41st IEEE Annual Computer Software and Applications Conference (COMPSAC'17). Volume 1*. 778–787.

[23] Edimar Manica, Renata Galante, and Carina F. Dorneles. 2014. SSUP - A URL-based method to entity-page discovery. In *Web Engineering, 14th International Conference (ICWE'14), Proceedings*. 254–271.

[24] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D.

Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. 2015. Never-ending learning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*.

[25] Alfonso Murolo and Moira C. Norrie. 2016. Revisiting web data extraction using in-browser structural analysis and visual cues in modern web designs. In *Web Engineering - 16th International Conference (ICWE'16), Proceedings*. 114–131.

[26] Stefano Ortona, Giorgio Orsi, Marcello Buoncristiano, and Tim Furche. 2015. WADaR: Joint wrapper and data repair. *PVLDB* 8, 12 (2015), 1996–1999.

[27] Stefano Ortona, Giorgio Orsi, Tim Furche, and Marcello Buoncristiano. 2016. Joint repairs for web wrappers. In *Proceedings of theInternational Conference on Data Engineering*. IEEE Computer Society, Washington,1146–1157.

[28] Disheng Qiu, Luciano Barbosa, Xin Luna Dong, Yanyan Shen, and Divesh Srivastava. 2015. Dexter: Large-scale discovery and extraction of product specifications on the web. *Proceedings of the VLDB Endowment* 8, 13 (Sept. 2015), 2194–2205.

[29] Gianluca Quercini and Chantal Reynaud. 2013. Entity discovery and annotation in tables. In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT'13)*. ACM, New York, NY, 693–704.

[30] Hassan A. Sleiman and Rafael Corchuelo. 2014. Trinity: On using trinary trees for unsupervised web data extraction. *IEEE Transactions on Knowledge and Data Engineering* 26, 6 (2014), 1544–1556.

[31] Márcio L. A. Vidal, Altigran S. da Silva, Edleno S. de Moura, and João M. B. Cavalcanti. 2006. Structure-driven crawler generation by example. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'06)*. ACM, New York, NY, 292–299.

[32] Tim Weninger, Thomas J. Johnston, and Jiawei Han. 2013. The parallel path framework for entity discovery on the web. *ACM Transactions on the Web* 7, 3, Article 16 (Sept. 2013), 29 pages.

[33] Naoki Yoshinaga and Kentaro Torisawa. 2007. Open-domain attribute-value acquisition from semi-structured texts. In *Proceedings of the 6th International Semantic Web Conference (ISWC'07), Workshop on Text to Knowledge: The Lexicon/Ontology Interface (OntoLex'07)*. Springer, Busan, South Korea, 55–66.

[34] Hwanjo Yu, Jiawei Han, and Kevin Chen-Chuan Chang. 2004. PEBL: Web page classification without negative examples. *IEEE Transactions on on Knowledge and Data Engineerings* 16, 1 (Jan. 2004), 70–81.

[35] Yanhong Zhai and Bing Liu. 2005. Web data extraction based on partial tree alignment. In *Proceedings of the 14th International Conference on World Wide Web (WWW'05)*. 76–85.

[36] Ce Zhang, Christopher Ré, Michael J. Cafarella, Jaeho Shin, Feiran Wang, and Sen Wu. 2017. DeepDive: Declarative knowledge base construction. *Communications of the ACM* 60, 5 (2017), 93–102.

Received February 2017; revised February 2019; accepted September 2019