SSUP – A URL-Based Method to Entity-Page Discovery

Edimar Manica[1,2], Renata Galante[2], and Carina F. Dorneles[3]

[1] Campus Avançado Ibirubá - IFRS - Ibirubá, RS, Brazil
edimar.manica@ibiruba.ifrs.edu.br
[2] PPGC - INF - UFRGS - Porto Alegre, RS, Brazil
galante@inf.ufrgs.br
[3] INE/CTC - UFSC - Florianópolis, SC, Brazil
dorneles@inf.ufsc.br

Abstract. Entity-pages are Web pages that publish data representing one only instance of a certain conceptual entity. In this paper we propose **SSUP**, a new method to entity-page discovery. Specifically, given a sample entity-page from a Web site (e.g., `Jolyon Palmer` entity-page from `GP2` Web site) we aim to find all same type entity-pages (driver entity-pages) from this Web site. We propose two structural URL similarity metrics and a set of algorithms to combine URL features with HTML features in order to improve the quality results and minimize the number of downloaded pages and processing time. We evaluate our method in real world Web sites and compare it with two baselines to demonstrate the effectiveness of our method.

Keywords: entity-pages, structural similarity, URL features, HTML features.

1 Introduction

The Web contains an increasing number of Web sites that can be considered a repository of pages with valuable information about real world entities. These pages are called *entity-pages* (or object-pages). Weninger et al. [8] define an entity-page as a Web page that describes a specific entity. For example, a Web page that describes a GP2 driver or a city council member.

Making use of the data presented in the entity-pages is an opportunity to create knowledge useful to several real applications (such as: comparative shopping, vertical search, named entity recognition and query suggestion). For instance, when we type the query "`players of real madrid`" on Google[1], it shows a list with all the players of Real Madrid Football Club including the attributes: `name`, `position` and `image`. If we click in one player of this list, the original query is replaced by the player name. The data for this kind of suggestion can be extracted from Wikipedia[2]. However, the Wikipedia does not contain all entities.

[1] `http://www.google.com/`
[2] `http://en.wikipedia.org/`

S. Casteleyn, G. Rossi, and M. Winckler (Eds.): ICWE 2014, LNCS 8541, pp. 254–271, 2014.

For example, the council members of Natal (a Brazilian state capital) are not presented in Wikipedia, but there is an official Web site with an entity-page for each council member describing the attributes: `name`, `political party`, `image`, `phone`, among others. As in general the entity-pages in the same Web site share a common template, HTML patterns can be learned to extract their data.

One important problem in this context is how to discover the entity-pages of the same type in one Web site. This is not a trivial task due to the variety of Web sites and structures. Some sites contain a page with a link to all entity-pages, while others divide these links in pages organized in a hierarchical manner and others divide these links in discrete pages (pagination). Moreover, some entity-pages have more details about an entity while others have less influencing in the HTML structure of the entity-pages. On the other hand, how the data in the entity-pages can change and new entity-pages can become available, it is necessary to re-execute the method to entity-page discovery periodically. Then, the number of downloaded pages and the processing time are important metrics to choose an effective method.

We found some methods to entity-page discovery in the literature. Methods based on HTML-tables [7] or human-compiled encyclopedias [6] are very restrictive. The method proposed by Weninger et al. [8] is based on the HTML and visual features, but in order to analyze the visual information is necessary to render the page in a browser loading all images, css and javascripts, then increasing the processing time. On the other hand, the method proposed by Blanco et al. [2] uses only the HTML features, then collect the information required is faster, but being based on only one kind of feature makes the method very sensitive to the parameter configuration.

Our goal is to find the set of entity-pages of the same type given a sample entity-page minimizing the number of downloaded pages and the processing time. For example, given the entity-page about `Jolyon Palmer` on the `GP2` Web site, we intent to find the set of driver entity-pages on the `GP2` Web site. The main contributions of this paper can be summarized as follows: (i) two structural URL similarity metrics ($weaksim_{url}$ and $strongsim_{url}$); (ii) **SSUP**: a new method to entity-page discovery that combines URL features with HTML features; (iii) show through experiments that combining URL features with HTML features improves the quality results of entity-page discovery and decreases the number of downloaded pages and the processing time; (iv) create datasets from real World Web sites for experiments.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 specifies the key concepts at the basis of our method. Section 4 presents the two structural URL similarity metrics proposed and the **SSUP** method. Section 5 presents a set of experiments we have conducted to evaluate our method on real Web sites comparing it with two baselines. Section 6 concludes the paper and presents future directions.

2 Related Work

Yu et al. [9] uses an SVM-based method to classify Web pages according to features from the content and URL of a Web page. Blanco et al. [3] presents a method for structurally clustering Web pages using only the URLs of the Web pages and simple content features. Although both methods analyze the URL of the Web page, their assumptions are different from **SSUP**, since they have as input all Web pages of the Web site in order to classify or cluster them while **SSUP** wants to download the minimum number of Web pages as possible to discover the entity-pages. **SSUP** represents URL in a way very similar to Blanco et al. [3], but **SSUP** treats this representation using the principle of TF-IDF [1] while Blanco et al. [3] uses the principle of minimum description length [4].

Kaptein et al. [6] present a Wikipedia-based method for entity-page discovery searching the Wiki-page for a link to the entity's home page. Lerman et al. [7] present a method that relies on the common structure of many Web sites, which present information as a list or a table, with a link in each entry pointing to a detail page containing additional information about that item. **SSUP** differs from these methods because it depends neither on the specific HTML markup nor on a human-compiled encyclopedia. He et al. [5] focus on deep Web, then the entity-pages are found through HTML form submissions while **SSUP** focuses on surface Web, then the entity-pages are found through browsing the Web site by its hyperlinks.

`Indesit` [2] and `GPP` (Growing Parallel Paths) [8] aim to find the set of entity-pages of the same type given a sample entity-page. `Indesit` models Web pages in terms of the DOM-structure of the HTML of the Web page and measure the structural similarity between two Web pages with respect to this feature. `GPP` combines both DOM-structure of the HTML and visual information. `SSUP` combines both DOM-structure of the HTML and URL information that is more robust that consider only DOM-structure of HTML and less costly that consider visual information. We choose `Indesit` and `GPP` as our baselines in the experiments because they are the most similar methods to **SSUP** and they depend neither on the specific HTML markup nor on a human-compiled encyclopedia. It is important to note that we reuse the definitions of `Indesit` related to the HTML of the Web pages because our contribution is how to use the URL features and to combine them with HTML features. The use of HTML features to entity-discovery seems to be consolidated since it was proposed in `Indesit` and reused by `GPP`.

3 Definitions

Here, we specify the key concepts at the basis of our method. First, we define the basic structures of a Web site (Section 3.1). Finally, we define the basic structures of a Web page (Section 3.2).

3.1 Web Site

A Web site is a directed graph whose nodes correspond to the pages of the Web that exist within the same domain and the edges correspond to the hyperlinks. According to our intuition, a part of a Web site is designed to allow the user to navigate in it from homepage until the entity-pages through a logical hierarchy of topics. The pages that are part of this logical hierarchy are classified as entity-pages or index-pages. Entity-pages were defined in Section 1. Index-pages are pages with links to the entity-pages or with links to other index-pages, whose role is to group entity-pages/index-pages in order to allow the user to navigate through a logical hierarchy of topics.

Definition 1. *(*Index page*) Let p be a Web page in the Web site s, EP be the set of all entity-pages in s, sEP be a subset of EP, IP be the set of all index-pages in s, sIP be a subset of IP. p is an index-page if it contains at least one link for each entity-page in sEP (or index-page in sIP) and its functional role in s is group sEP (or sIP).*

Each index-page and entity-page has a degree of topic specificity that represents the number of attributes that the page restricts. Entity-pages have the highest degree of topic specificity, since always entity-pages restrict more attributes that their index-pages, i.e., they represent the more specific topic in the logical hierarchy.

Definition 2. *(*Degree of topic specificity*) Let an entity e with two attributes a_1 and a_2. Let $P_x[w]$ ($P_y[z]$) be an index-page with links to entity-pages that describe instances where the value of a_1 is x (y) and the value of a_2 is w (z). Let P_x be an index-page with links to index-pages with links to entity-pages that describe instances where the value of a_1 is x, then we say that $P_x[w]$ and $P_y[z]$ has the same degree of topic specificity (because they restrict the same number of attributes) and $P_x[w]$ has higher degree of topic specificity that P_x (because $P_x[w]$ restricts more attributes that P_x).*

Example 1. In Figure 1, `Page A` has the lowest degree of topic specificity restricting the value of the attribute `type` to `car`. The pages `B`, `C` and `D`, besides restricting the `type` attribute value, restricts the value of the `manufacture year` attribute, then they have a degree of topic specificity higher than `Page A`. The pages `E`, `F`, `G`, `H`, `I` and `J` have the highest degree of topic specificity because they are entity-pages.

The logical hierarchy of topics is a tree, named entity-tree, whose root is the index-page with the lowest degree of topic specificity, the leaves are the entity-pages and non-leaves are index-pages.

Definition 3. *(*Entity-tree*) Let Et be a tree where the node set and the edge set are subsets of that of a given Web Site. Et is an entity-tree if it satisfies the following properties: (1) all leaf nodes are* entity-pages *of the* same type*; (2) all leaf nodes are at the same level; (3) all non-leaf nodes are index-pages; (4) the*

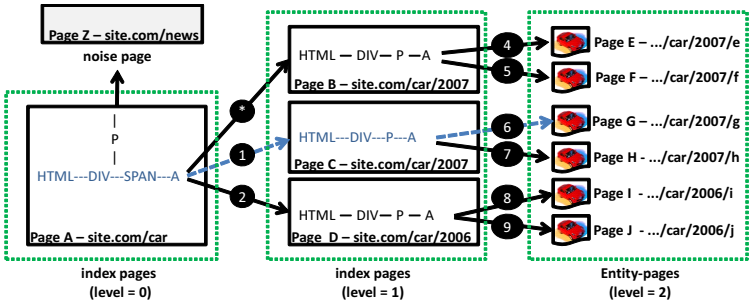


Fig. 1. Entity-tree example. The `Page Z` is highlighed because it is not part of the entity-tree.

root is the index-page with links to all index-pages of the next level (level+1); (5) all pages at a same level have the same degree of topic specificity*; (6) the higher the level of a page the higher its degree of topic specificity; (7) all index-pages at a same level have links to their child nodes in a same link DOM path; and (8) all nodes at the same level of a same entity-tree share a similar URL and HTML structure.*

Example 2. The illustration in Figure 1 describes an entity-tree with height two. The pages that are not part of the logical hierarchy of topics (e.g. `Page Z`) do not belong to the entity-tree and are referred as `noise pages`.

A Web Site can have more than one entity-tree to the same entity-type with a different topic structure. For example, a Web Site with software project entity-pages can be designed in such way that user may navigate through a logical hierarchy of licenses or programming languages. For each type of entity-pages available in a Web site must be at least one entity-tree.

In real Web sites usually occurs a situation that violates the property 5: "*all pages at a same level have the same degree of topic specificity*". This situation, called *pagination*, occurs when an index-page is pointed by an index-page of the same degree of topic specificity instead of an index-page of the immediately preceding degree of topic specificity (more general topic). In this case, we have a pagination page and need a pagination operation.

Definition 4. *(*Pagination page*) Let $p1$ be an index-page, $pp1$ be the parent node of $p1$, the Web page $p2$ is a pagination page if $p1$ and $p2$ have the same degree of topic specificity, $p1$ points to $p2$ and there is not an index-page of the same degree of topic specificity of $pp1$ pointing to $p2$.*

Definition 5. *(*Pagination operation*) Let $p1$ be an index-page, $pp1$ be the parent node of $p1$, and $p2$ be a pagination page pointed by $p1$, then pagination operation states that we need to remove the edge from $p1$ to $p2$ and add an edge from $pp1$ to $p2$.*

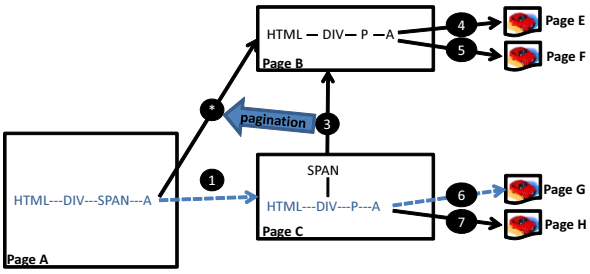


Fig. 2. Example of pagination operation

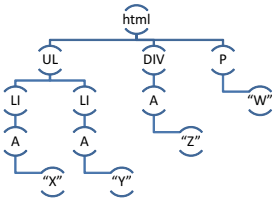


Fig. 3. Example of the DOM tree of a Web page

Example 3. The illustration in Figure 2 shows the pagination operation that generates the valid entity-tree presented in Figure 1. Originally the Web site had the edge 3 and did not have the edge *, violating the property 5. Then, after pagination operation, the edge 3 was removed and the edge * was added.

3.2 Web Page

We consider that the structure of a Web page is defined by its URL and HTML features. The URL feature is defined through a url-schema. A url-schema is the set of terms of the URL.

Definition 6. *(*url-schema*) A url-schema of a Web page p, denoted* $v(p)$*, is the set of terms of the URL of the p.*

In order to understand our concept of URL term is necessary to know that we see a URL as a sequence of substrings (called *tokens*) split by "/", "?" or "&" characters. Each token is a set of substrings (*sub-tokens*) split by non-alphanumeric characters, changing from letter to digit and vice versa. Then, a URL term is a sub-token associated with the position of the token that contains it. We do not consider the position of each sub-token in a token, since URLs of the Web pages with same entity-type can have different number of sub-tokens.

Definition 7. *(*URL Term*) Let* T *be a sequence of tokens of a URL* u *(*T_1*,* T_2*, ...,* T_n*), where* T_i *occurs in* u *before* T_{i+1}*. Each token* T_i *is a set of sub-tokens (*$S_i[1]$*,* $S_i[2]$*, ...,* $S_i[n]$*), where* $S_i[j]$ *is the* j*th sub-token of the token* T_i*. Each sub-token* $S_i[j]$ *associated with* i *is a URL term. An additional URL term is the size of* T*.*

Example 4. Figure 4 shows an example of a URL, describing all tokens, the sub-tokens of the token T_3 and all URL terms. The additional URL term is highlighted.

The HTML feature is defined through a html-schema. An html-schema is the set of link DOM paths of the HTML.

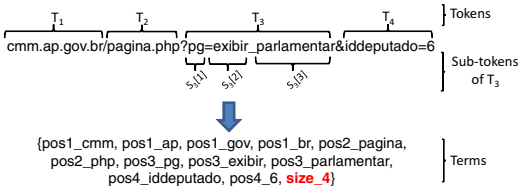


Fig. 4. Example of tokens, sub-tokens and URL terms

Definition 8. *(*html-schema*) A html-schema of a Web page p, denoted $\Delta(p)$, is the set of link DOM paths in p.*

In order to understand our concept of link DOM path is necessary to know that we see the HTML of Web pages as a Document Object Model (DOM) tree. Then, the link DOM path is a path from the root node to an anchor node.

Definition 9. *(*Link DOM Path*) Let p be a Web page, a link DOM path is a path through the DOM-tree of p that starts from the root and terminates into an anchor node.*

Example 5. The html-schema of the page presented in Figure 3 is $(HTML - UL - LI - A, HTML - DIV - A)$. The path $HTML - P$ does not belong to the html-schema because it does not terminate into an anchor node.

4 SSUP – A URL-Based Method to Entity-Page Discovery

The **SSUP** (**S**tructurally **S**imilar **U**RLs and **P**ages) is a method for entity-page discovery. Specifically, **SSUP** aims to find the set of same type entity-pages in a Web site given a sample entity-page.

We present an overview of our method using the Figure 1. **SSUP** starts with an entity-page as sample page (`Page G`) and find its index-page (`Page C`). Then, a new instance of our problem is recursively triggered using now the index-page found as sample page (finding `Page A` as index-page of the `Page C`). This process is performed until to find the root of the entity-tree (`Page A` in the case). Then, **SSUP** obtains a *sample page path* from root until the given entity-page (`Page A -> Page C -> Page G`). We call *sample page* each page from the sample page path. We call *sample link DOM path* each link DOM path that points to a sample page. The `Page C` is the sample page of the level 1 and "`HTML-DIV-SPAN-A`" is its sample link DOM path.

After, **SSUP** transverses the sample page path from root until the leaf (which is the sample entity-page) catching same level pages. To perform this, in each level x of the entity-tree, **SSUP** catches all pages pointed by the pages of that level and analyzes the structural URL and HTML similarity between these pages and the sample page in the level $x+1$ in order to discard pages that do not belong

to the entity-tree. For example, consider Figure 1, analyzing the level 0, **SSUP** catches all pages pointed by `Page A`, including pages `C`, `D` and `Z`. Remember that `Page B` is actually pointed by `Page C`. **SSUP** analyzes the structural URL and HTML similarity between each page with the `Page C` (sample page of level 1) in order to prune noise pages (`Page Z`). By this time, we have pages `C` and `D` in the level 1. Then, **SSUP** needs to discover if these pages contain pagination pages. To perform this, **SSUP** catches all pages pointed by pages `C` and `D` that are structural URL and HTML similar to them (finding the `Page B` as a pagination page of the `Page C`). The process is performed to the next level catching all pages pointed by the pages `B`, `C` and `D`, analyzing the structural URL and HTML similarity between each page with `Page G` (sample page of level 2) and so on.

In the next subsections, we present the similarity metrics used to determine the structural URL and HTML similarity between Web pages (Section 4.1). Finally, we propose four algorithms (Section 4.2). The first algorithm finds the index-page of a given entity/index-page. The second algorithm catches same level pages. The third algorithm catches pagination pages. The last algorithm combines the previous algorithms to perform entity-page discovery.

4.1 Structural Similarity Metrics

In this subsection we present two metrics based on the Web page URL (Weak URL Similarity and Strong URL Similarity) and reuse a metric based on the Web page HTML (sim_{html}) proposed by Blanco et al. [2] in order to measure the structural similarity between two Web pages. The similarity metrics Weak URL Similarity and HTML Similarity are used to determine if a page is a pagination of other. The similarity metrics Strong URL Similarity and HTML Similarity are used to determine if two pages are in the same level in the entity-tree.

Weak URL Similarity is a simple similarity metric that compares two Web pages based on the terms that belong to their `url-schema`. This metric gives the same importance to each term. So, if two URLs share the term "`www`" or the term "`driver`" has the same impact in the result.

Definition 10. *(*Weak URL Similarity*) Let $\upsilon(p_1)$ be the url-schema of the Web page p_1 and $\upsilon(p_2)$ be the url-schema of the Web page p_2, the weak URL similarity between p_1 and p_2 is defined as:*

$$weaksim_{url}(p_1, p_2) = \frac{|\upsilon(p_1) \cap \upsilon(p_2)|}{|\upsilon(p_1) \cup \upsilon(p_2)|} \quad (1)$$

The Strong URL Similarity also compares two Web pages based on the terms that belong to their `url-schema`. However, it is more robust than Weak URL similarity since it assigns a different weight to each URL term according to its importance to distinguish same level pages from non-same level pages. In this metric, the URL terms are assumed to be all mutually independent. The URL of the sample page is represented as vector of URL term weights in a n-dimensional space, in which n is the total number of URL terms.

Definition 11. *(*Strong URL Similarity*) Let $\upsilon(sp)$ be the url-schema of the sample page sp, $\upsilon(p)$ be the url-schema of the Web page p, which we desire to compare with sp, and ip be the index-page of sp, the strong URL similarity between sp and p is defined as:*

$$strongsim_{url}(ip, sp, p) = \frac{\sum_{t \in \upsilon(sp) \cap \upsilon(p)} W_t(ip, sp)}{\sum_{t \in \upsilon(sp)} W_t(ip, sp)} \quad (2)$$

The weight of a URL term used in the Strong URL Similarity is based on the observation of Weninger et al. [8] "lists usually contain items which are similar in type or in content" and our observation "entity-pages of the same type usually share a common URL structure". In our context, we have a sample page and we want to know how are other web pages similar to sample page based on their URLs. The idea is that terms that occur in many URLs, from links of the index-pages, in the same link DOM path that the sample page, and occur in few link DOM paths from the index-pages are more important because they have a high discriminating power. For example, in Table 1, considering `L1` as sample page and $P1$ as its index-page, the term "`pos2_car`" should have the highest weight because it occurs in five URLs that are in the same link DOM path of the sample page ("`HTML-UL-LI-A`") and does not occur in other link DOM paths.

Definition 12. *(*URL Term Weight*) Let ip be the index-page of the sample page sp. Let P be a set of link DOM paths (P_1, P_2, ..., P_n) from ip. Let $P_i = (P_i[1], P_i[2], ..., P_i[n])$, where $P_i[j]$ is the url-schema of the jth URL in the link DOM path P_i. Let P_{sp} be the P_i that contains $\upsilon(sp)$, and t a URL term from $\upsilon(sp)$, the weight of t from sp in ip is defined as:*

$$W_t(ip, sp) = TF_{t,P_{sp}} \times IDF_{t,P} \quad (3)$$

where, $TF_{t,P_{sp}}$ is the number of url-schemas in P_{sp} that contains the term t and $IDF_{t,P}$ is defined as:

$$IDF_{t,P}(ip) = log(\frac{|P|}{|P_i \in P : t \in P_i|})$$

Example 6. The Table 1 presents an example of same level index-pages with their links and the link DOM path that points to each link. For each link, we show its strong URL similarity with the sample page $L1$. The highest strong URL similarity is with itself (`1.00`), followed by pages `L2` and `L3` that share with $L1$ the sub-token "`car`" in the second token and the sub-token "`ford`" in the third token. These sub-tokens occur only in the link DOM path of the sample page. The strong URL similarity of the `Page L7` is 0 because all sub-tokens shared between this page and the sample page (e.g., "www") occurs in all link DOM paths and all occurrences are in the first token. Pages L0 and L8 do not have ss_u because they are in a link DOM path different from sample page.

HTML Similarity is a simple similarity metric that compares two Web pages based on the link DOM paths that belong to their `html-schema`. This metric gives the same importance to each link DOM Path.

Table 1. An example of same level index-pages with their link DOM paths and the pages pointed by each link DOM path (column `Links`). The column ss_u shows the strong URL similarity between each page in the column `Links` with the sample page `L1`.

Same Level Index-pages	Link DOM Path	Links	ss_u
P1	HTML-DIV-A	L0 = www.website.com/privacy_policy.htm	-
	HTML-UL-LI-A	L1 = www.website.com/car/ford/ba-falcon-rx-8	1.00
		L2 = www.website.com/car/ford/svt-mustang-cobra-coupe	0.68
		L3 = www.website.com/car/ford/mustang-mach-1	0.68
P2		L4 = www.website.com/car/ferrari/360-challenge-stradale	0.53
P3		L5 = www.website.com/car/audi/a4-cabriolet-3.0	0.53
P4		L6 = www.website.com/test_drive/rx-8.htm	0.21
P5		L7 = www.website.com/news.htm	0.00
	HTML-DIV-A	L8 =www.website.com/wallpapers.htm	-

Definition 13. *(*HTML Similarity*) Let $\Delta(p_1)$ be the html-schema of the Web page p_1 and $\Delta(p_2)$ be the html-schema of the Web page p_2, the HTML similarity between p_1 and p_2 is defined as:*

$$sim_{html}(p_1, p_2) = \frac{|\Delta(p_1) \cap \Delta(p_2)|}{|\Delta(p_1) \cup \Delta(p_2)|} \tag{4}$$

4.2 Algorithms

In this subsection we present the algorithms that compose the method **SSUP**. The Algorithm 1 aims to find the index-page of a given page. The intuition behind the algorithm is that the index-page of a given page *sp* is the page delivering the largest number of more structurally similar URLs to *sp*. The solution presented by the algorithm collects the pages pointed by the given page (line 4). Then, the pages that do not have a link to the given page are removed (line 5). Finally, the algorithm returns the page with the max candidate index-page weight (line 6).

The candidate index-page weight accumulates the Strong URL Similarity between its links and a given page. It is considered only the links in the same link DOM path that the given page. The Algorithm 1 uses this weight to find the index-page of a given page, choosing the candidate index-page with the highest weight.

Algorithm 1. Top Index-Page algorithm

1: INPUT: a sample page *sp*;
2: OUTPUT: the top index-page;
3: **begin** *top_index_page(sp)*
4: P = GetLinks(*sp*);
5: remove p_i from P where p_i does not have a link to *sp*;
6: **return** p_i from P with max $W_{ci}(p_i, sp)$; //Equation 5
7: **end**

Algorithm 2. Catching Same Level Pages algorithm

1: INPUT: a set of index-pages of the level x (IP), the sample page of the level $x+1$ (sp), the sample link DOM path of sp (sdp);
2: OUTPUT: the set of pages of the level $x+1$ pointed by IP;
3: **begin** *catching_same_level_pages(IP, sp, sdp)*
4: add sp to rs;
5: **for each** ip_i **IN** IP **do**
6: add GetLinksInPath(ip_i, sdp) to L;
7: **end for**
8: create a list of groups $G = (G_1, G_2, ..., G_n)$, where each group G_i contains each link $l_i \in L$ (except sp) with the same $strongsim_{url}$ with sp;
9: sort G by $strongsim_{url}$ decreasing order;
10: add all links from G_1 to rs;
11: $minsim_{html} = \min_{l_i \in G_1} sim_{html}(sp, l_i)$;
12: remove G_1 from G
13: **for each** G_i **IN** G **do**
14: **if** $\max_{l_i \in G_i} sim_{html}(sp, l_i) >= minsim_{html}$ **then**
15: add all links from G_i to rs;
16: **else**
17: break;
18: **end if**
19: **end for**
20: **return** rs
21: **end**

Definition 14. *(*Candidate index-page weight*) Let p be a candidate index-page for the Web page sp, L be a set of pages pointed by p through the same link DOM path that sp, the candidate index-page weight of p for sp is defined as:*

$$W_{ci}(p, sp) = \sum_{l \in L} strongsim_{url}(p, sp, l) \tag{5}$$

The goal of the Algorithm 2 is given a set of index-pages of level x and the sample page of the level $x+1$, finds all pages of level $x+1$ delivered by the index-pages. The intuition behind the algorithm is that same level pages have structurally similar URLs and HTMLs. The solution proposed is that same level pages have larger $strongsim_{url}$ and sim_{html} than non-same level pages.

The Figure 5 presents an example of the execution of the Algorithm 2, considering the pages `P1`, `P2`, `P3`, `P4` and `P5` as index-pages of level x (called just index-pages); `L1` as the sample page of level $x+1$ (called just sample page) and "`HTML-UL-LI-A`" as the sample link DOM path of the sample page, i.e., the path in the index-page that points to sample page. The Table 1 describes these pages. The goal is to return all pages of level $x+1$. In the line 4, the sample page is added to the result set. In lines 5-9, the Web pages pointed by the index-pages through the sample link DOM path, except sample page, are collected and grouped by their strong URL similarity with the sample page. The

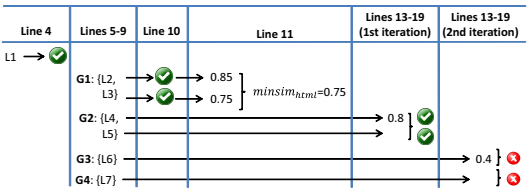


Fig. 5. An execution of the Algorithm 2. Input: IP=(P1, P2, P3, P4, P5), SP=L1, sdp=HTML-UL-LI-A (described in Table 1). Ok icon means that the Web page was added to the result set. Not ok icon means that the Web page was discarded.

groups are sorted by their strong URL similarity decreasing order. Four groups are created: G1, G2, G3, G4. The group G1 (G4) has the Web pages with largest (lowest) strong URL similarity value with the sample page. In line 10, all Web pages from G1 (L2 and L3) are added to result set. In line 11, we compute the min HTML similarity between the sample page and the pages from G1, called $minsim_{html}$. In this example, the $minsim_{html}$ is the HTML similarity between sample page and L3 (0.75). In line 12, omitted in the figure, the G1 is removed from the list of groups. The first iteration of the "for" (lines 13-19) analyzes the group G2, since it has the largest strong URL similarity after G1 has been removed. It is computed the HTML similarity between the first Web page of the G2 (L4) and the sample page. As the HTML similarity value is greater than $minsim_{html}$ all Web pages from G2 are added to result set, without compute the HTML similarity for other Web pages of G2. The second iteration of the "for" (lines 13-19) analyzes the group G3. Since the min HTML similarity between sample page and the pages of G3 is less than $minsim_{html}$, so all pages of the group (L6) are discarded and all remain groups (G4) are discarded too. This follows the idea "the same level pages have larger $strongsim_{url}$ and sim_{html} than non-same level pages". Finally, the result set (with pages L1, L2, L3, L4 and L5) is returned in line 20.

The Algorithm 2 fails in catch pagination pages because these pages are not pointed by index-pages of the preceding level, but by pages of the same level. So, the Algorithm 3 aims to catch these pages. The intuition behind this algorithm is that pages delivered by index-pages of the level x with HTML similarity to pages of the level x larger than to pages of the level $x+1$ are strong candidate to be pagination pages. The solution presented in the algorithm receives as input: (i) a set of index-pages of the level x; (ii) a sample page of the level x (sp_x); and (iii) a sample page of the level $x+1$ (sp_{x+1}). Then, in lines 4-6, the pages pointed by the index-pages are collected. These pages are grouped according to their Weak URL Similarity with sp_{x+1} (line 7). After, one page of each group has its HTML similarity computed with sp_x and with sp_{x+1}, and if the first score is greater or equal to the second score, then all pages of the group are added to the result set (lines 8-13). Finally, the result set is returned (line 14).

It is important to note that this step catches noise pages. But these pages are pruned when a next level is analyzed because noise pages do not point to pages with structurally similar URL and HTML to the sample page.

Algorithm 3. Pagination algorithm

```
1: INPUT: the sample page of the level x (sp_x), the sample page of the level x + 1
   (sp_{x+1}), index-pages of the level x (IP);
2: OUTPUT: a set of pagination pages of the level x pointed by the IP;
3: begin pagination(sp_x, sp_{x+1}, IP)
4: for each ip_i IN IP do
5:       add GetLinks(ip_i) to L;
6: end for
7: create a list of groups G = (G_1, G_2, ..., G_n), where each group G_i contains each link
   l_i ∈ L with the same weaksim_url(sp_{x+1}, l_i) and has the same link DOM path;
8: for each G_i IN G do
9:       l_0 = first link from G_i
10:      if sim_html(sp_x, l_0) >= sim_html(sp_{x+1}, l_0) then
11:            add all links from G_i to rs;
12:      end if
13: end for
14: return rs
15: end
```

The Algorithm 4 shows the overall `SSUP` algorithm. In lines 4-12 the sample page path is created by calling recursively the function *top_index_page*. Note that we do not have an approach to discovery the root node, instead we have a parameter that estimates the height of the entity-tree. If this parameter is less than the real entity-tree height then some entity-pages will not be discovered. If this parameter is greater than the real entity-tree height then some unnecessary pages will be downloaded, but the quality results is not affected since the noise pages are pruned when we analyze the URL and HTML structural similarity. However, we add an additional stopping criterion (lines 7-9): when an index-page discovered for a level x was already discovered for a level y where $y > x$. In lines 13-22, we transverse the entity-tree from root to leaves catching same level entity-pages through the functions *catching_same_level_pages* and *pagination*. It is important to note in lines 19-21 that entity-pages do not have pagination.

5 Experiments

In this section, we describe the experiments we performed in order to evaluate the effectiveness of **SSUP**. We compared **SSUP** with `Indesit` [2] and `GPP` [8]. We chose to compare our method with these baselines among the methods that focus on surface Web because they depend neither on the specific HTML markup [7] nor on a human-compiled encyclopedia [6].

In order to analyze the results of experiments, we evaluated them considering the following measures: (i) recall, (ii) precision; (iii) F1-measure; (iv) number of downloaded pages; (v) processing time and (vi) T-test. T-test is a statistical hypothesis test, and recall, precision and F1-measure are well known quality measure metrics in IR community [1]. We show just F1-measure results in terms

Algorithm 4. SSUP algorithm

```
1: INPUT: one sample entity-page sp, the entity-tree height h;
2: OUTPUT: the set of entity-pages of the same type that sp contained in the Web site;

3: begin ssup(sp, r)
4: add sp to SPP[h]; //SPP = Sample Page Path
5: for i=0 until h − 1 do
6:       tmp = top_index_page(SPP[h − i]);
7:       if SPP contains tmp then
8:             break;
9:       end if
10:      aux = h − i − 1;
11:      add tmp to SSP[aux];
12: end for
13: for i=aux until h − 1 do
14:      if i = aux then
15:            add SPP[i] to IP;
16:      end if
17:      sdp = the link DOM path in SPP[i] that contains the link to SPP[i + 1];
18:      add catching_same_level_pages(IP, SPP[i + 1], sdp) to IP;
19:      if i < h - 2 then
20:            add pagination(SPP[i + 1], SPP[i + 2], IP) to IP;
21:      end if
22: end for
23: return IP
24: end
```

of quality because recall and precision are combined in F1-measure. In the efficiency aspect we evaluated the total number of downloaded pages and the processing time. We previously downloaded the pages of each Web site used in order to avoid the network interference in the processing time.

5.1 Setup

The experiments were performed on a Intel Core 2 Quad 2.66GHz running Ubuntu 9.10, with 8GB of main memory and 5TB of disk space. **SSUP**, `Indesit` and GPP were implemented in Java. We created 28 datasets, each one from a specific Web site, considering a specific entity-type. We created two groups with these datasets: (i) `multiple type group` with 10 datasets of different entity-types (except `association` that repeats once), described in Table 2; and (ii) `council type group` with 18 datasets of council member entity-type. In the last group, we analyzed the official Web sites of the council of the 26 Brazilian state capitals. However, we excluded: 4 Web sites because they do not have an entity-page for each council member; 3 Web sites because the council member entity-pages are internal frames of the index-page; and 1 Web site because it does not allow crawling its pages.

Table 2. Datasets of the multiple type group

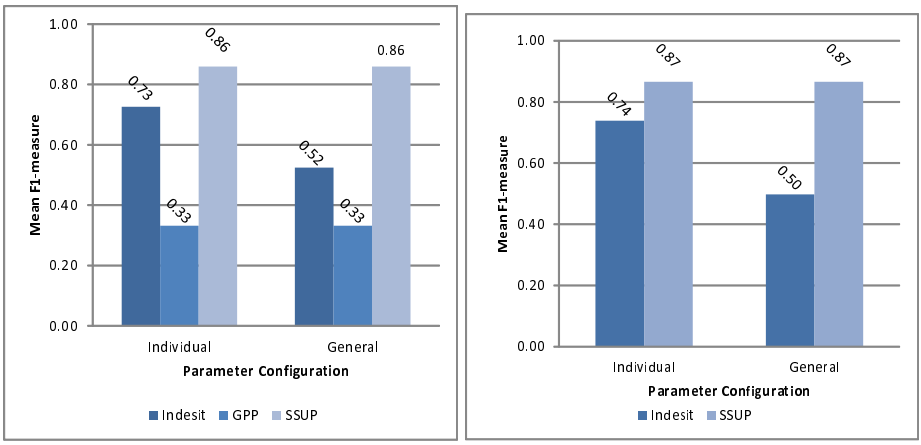
ID	Web site	Entity-type	NEP
Fifa	www.fifa.com	Association	209
GP2	www.gp2series.com	Driver	32
Guitar	www.guitaretab.com	Group	32318
MIT EECS	www.eecs.mit.edu	People	1042
Olympic-A	www.olympic.org	Association	204
Olympic-S	www.olympic.org	Sport	56
Pgfoundry	pgfoundry.org	Software Project	382
Senado	www.senado.gov.br	Senator	121
Stanford EE	engineering.stanford.edu	Staff	473
Supercar	www.supercarsite.net	Car	512

Label: NEP: number of entity-pages

`Indesit` has two parameters: (i) `T-value` - the html-schema similarity threshold between two Web pages varying from 0 to 1; (ii) `Tag features` - defines if the `Indesit` considers as a link DOM path only tags; tags and attribute names; or tags, attribute names and attribute values. We tested all combinations between T-values (0.1; 0.2; ...; 0.9) and tag features. The best parameter configuration related to quality of results in the `multiple type group` was `T-value`= 0.5 and `Tag feature` = tags and attribute names; and in the `council type group` was `T-value`= 0.8 and `Tag feature` = tags and attribute names. **SSUP** has one parameter: `H-value` - the estimate of the entity-tree height. We tested the `H-values`: 1, 2, ..., 5. The best `H-value` related to quality of results was 2 and 1 in the `multiple type group` and `council type group`, respectively. `GPP` has one parameter: `K-value` - the number of iterations. We tested only one iteration because the processing time of `GPP` is too high compared with `Indesit` and `SSUP`. We ran all methods with each configuration three times, each one with a different sample entity-page in order to verify that the behavior of the method does not depend on specific features of the sample entity-page. The three sample entity-pages were chosen randomly from the first page displayed (when there is pagination).

5.2 Comparison

The goal of these experiments is to compare the methods **SSUP**, `Indesit` and `GPP` in terms of quality and efficiency. For **SSUP** and `Indesit` we show the results with the best parameter configuration in terms of quality results in each individual dataset (`individual` configuration) and with the best parameter configuration in terms of quality results considering all datasets of the group analyzed (`general` configuration). We run `GPP` with only one iteration (so, individual and general configuration are the same one) and only on the `multiple type group` because its processing time is too high.



(a) `multiple type group` **(b)** `council type group`

Fig. 6. Mean F1-measure comparison

Figure 6 shows a comparison between the F1-measure of **SSUP**, `Indesit` and `GPP` on the `multiple type group` (Figure 6a) and between **SSUP** and `Indesit` on the `council type group` (Figure 6b). SSUP outperformed `Indesit` and `GPP`. There is a statistical significant difference (p-value < 0.05) between the F1-measure values of the methods (except between `Indesit` and `SSUP` with individual configuration on the `multiple type group` where p-value was 0.50126). Considering the general configuration, **SSUP** had better quality results than `Indesit` because it learns the threshold of html-schema similarity in each dataset from the groups of Web pages with high structurally URL similarity with the sample entity-page and that are in the same link DOM path that the sample entity-page. On the other hand, in `Indesit` the user should define a threshold of html-schema similarity between entity-pages and this threshold varies from dataset to dataset. Considering the individual configuration (the smallest difference in terms of F1-measure between **SSUP** and `Indesit`), **SSUP** only outperformed `Indesit` in datasets that contain an intersection of entity-pages and non-entity pages related to the html-schema similarity. In these cases, in `Indesit` if we increase the T-value the precision increases, but the recall decreases. If we decrease the T-value the recall increases, but the precision decreases because the method catches erroneous pages. On the other hand, **SSUP** uses the structurally URL similarity as an additional feature to discriminate entity-pages from non entity-pages. `GPP` presented the worst values of F1-measure because, in many cases, one iteration was not enough to find most entity-pages. However, even on the `Olympic - S` dataset where `GPP` found all entity-pages (recall = 1) many erroneous pages were returned decreasing the precision to 0.32 because the host city entity-pages have similar HTML and visual features to sport entity-pages.

The main SSUP fail cases occurred when: (i) the entity-pages do not contain a link to their index-page affecting the recall of SSUP; (ii) some entity-pages have different link DOM paths affecting the recall of SSUP; (iii) the URL of the

Table 3. Average of downloaded pages and processing time on the `multiple type group`

	Indesit Individual	Indesit General	GPP	SSUP Individual	SSUP General
Downloaded Pages	1680	980	212	778	781
Processing Time	854	779	2984	46	47

entity-pages and non entity-pages are composed of the domain plus "/" plus one token affecting the precision.

The average of downloaded pages on the `multiple type group` for each method is presented in Table 3. Considering individual configuration, the mean number of downloaded pages of `Indesit` was 1680 while **SSUP** was 778 (53.69% less than `Indesit` and p-value < 0.05). This occurred because **SSUP** filters out some pages just by their URLs, without needing to download them while `Indesit` needs to download a Web page to evaluate it. Considering general configuration, the mean number of downloaded pages of `Indesit` was 980 while `SSUP` was 781 (20.31% less than `Indesit`). However, this difference is not statistically significant (p-value > 0.05). This occurred because `Indesit` with general configuration had a low recall in many datasets, consecutively it downloaded fewer pages. `GPP` presented the lowest number of downloaded pages (72.86% less than **SSUP** with general configuration and p-value < 0.05). The main reason of this big difference is the low recall in most datasets. However, even in the `Olympic - S` dataset where all entity-pages were found, the number of downloaded pages was lower. This occurred because we used only one iteration, but if we increase the number of iterations to increase the recall, more pages are downloaded.

The average of processing time for **SSUP**, `Indesit` and `GPP` on the `multiple type group` is presented in Table 3. **SSUP** had the shortest processing time (with p-value < 0.05 in all cases). This behavior can be explained because **SSUP** filters out some pages just by their URLs, without needing to analyze the HTML of them while `Indesit` needs to analyze the HTML of all downloaded pages. Moreover, after computing the Sample Page Path, **SSUP** analyzes in each Web page only the sample link DOM path while `Indesit` analyzes all link DOM paths in all Web pages downloaded. `GPP` had the worst values of processing time, even having downloaded less Web pages, because it explores visual information and to perform this it is necessary to render the Web page in a browser loading all images, css and javascripts, which greatly increases the processing time.

The experiments showed that **SSUP** outperforms `Indesit` related to the quality results when we define one unique parameter configuration for all datasets. When we use the best parameter configuration for each individual dataset, **SSUP** improves the quality results when the dataset has an intersection between entity-pages and non-entity pages related to the html-schema similarity. In other cases, **SSUP** does not improve the quality results, but it reduces the number of downloaded pages and processing time. Furthermore, the **SSUP** parameter - H - can be defined by a user visually navigating through the Web site

while the `Indesit` parameters (T and tag feature) require a processing of the entity-pages. Then, we can conclude that combining URL and HTML features improves the quality results of entity-page discovery and decreases the number of downloaded pages and the processing time.

6 Conclusions

In this paper we propose a new method to entity-page discovery, called **SSUP**. Specifically, given a sample entity page of a Web site, we want to find the set of same type entity-pages of that Web site. The main contribution of **SSUP** is to combine URL and HTML features to entity-page discovery. We first define the basic structures of a Web site and a Web page according to our method. Second, we propose two structural URL similarity metrics and reuse a HTML similarity metric. Thirdly, we develop a set of algorithms that compose the method. We demonstrated the effectiveness of **SSUP** by comparing it with `Indesit` and `GPP` using real datasets.

The experiments have showed that combining URL and HTML features improves the results in terms of quality, number of downloaded pages and processing time. Entity-pages dynamically generated by populating fixed pages templates with content from a back-end DBMS have the better results. However, we are working on incorporating other features besides URL and HTML to achieve good results in people entity-pages created by different users, since these pages usually do not contain a link to its index-page affecting the recall of SSUP. We are also creating a parallelized version of SSUP.

References

1. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval: The Concepts and Technology Behind Search. Addison Wesley Professional (2011)
2. Blanco, L., Crescenzi, V., Merialdo, P.: Efficiently locating collections of web pages to wrap. In: WEBIST, pp. 247–254. INSTICC Press (2005)
3. Blanco, L., Dalvi, N.N., Machanavajjhala, A.: Highly efficient algorithms for structural clustering of large websites. In: WWW, pp. 437–446. ACM (2011)
4. Grünwald, P.D.: The Minimum Description Length Principle (Adaptive Computation and Machine Learning). The MIT Press (2007)
5. He, Y., Xin, D., Ganti, V., Rajaraman, S., Shah, N.: Crawling deep web entity pages. In: WSDM, pp. 355–364. ACM (2013)
6. Kaptein, R., Serdyukov, P., de Vries, A.P., Kamps, J.: Entity ranking using wikipedia as a pivot. In: CIKM, pp. 69–78. ACM (2010)
7. Lerman, K., Getoor, L., Minton, S., Knoblock, C.A.: Using the structure of web sites for automatic segmentation of tables. In: SIGMOD Conf., pp. 119–130. ACM (2004)
8. Weninger, T., Johnston, T.J., Han, J.: The parallel path framework for entity discovery on the web. ACM Trans. Web 7(3), 16:1–16:29 (2013)
9. Yu, H., Han, J., Chang, K.C.C.: Pebl: Web page classification without negative examples. IEEE Trans. on Knowl. and Data Eng. 16(1), 70–81 (2004)