Orion: a Cypher-based Web Data Extractor

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

[1] II - UFRGS - Porto Alegre, RS - Brazil,
{edimar.manica,galante}@inf.ufrgs.br
[2] Campus Ibirubá - IFRS - Ibirubá, RS - Brazil,
[3] INE/CTC - UFSC - Florianópolis, SC - Brazil,
dorneles@inf.ufsc.br

Abstract. The challenges in Big Data start during the data acquisition, where it is necessary to transform non-structured data into a structured format. One example of relevant data in a non-structured format is observed in entity-pages. An entity-page publishes data that describe an entity of a particular type (e.g. a soccer player). Extracting attribute values from these pages is a strategic task for data-driven companies. This paper proposes a novel class of data extraction methods inspired by the graph databases and graph query languages. Our method, called **Orion**, uses the same declarative language to learn the extraction rules and to express the extraction rules. The use of a declarative language allows the specification to be decoupled from the implementation. **Orion** models the problem of extracting attribute values from entity-pages as Cypher queries in a graph database. To the best of our knowledge, this is the first work that models the problem of extracting attribute values from template-based entity-pages in this way. Graph databases integrate the alternative database management systems, which are taking over Big Data (with the generic name of NoSQL) because they implement novel representation paradigms and data structures. Cypher is more robust than XPath (a query language that is common used to handle web pages) because it allows traversing, querying and updating the graph, while XPath is a language specific for traversing DOM trees. We carried out experiments on a dataset with more than 145k web pages from different real-world websites of a wide range of entity types. The **Orion** method reached 98% of F1. Our method was compared with one state-of-the-art method and outperformed it with a gain regarding F1 of 5%.

Keywords: Data Extraction, Cypher, Graph Database, Graph Query Language, NoSQL

1 Introduction

Heterogeneity, scale, timeliness, complexity, and privacy problems with Big Data impede progress at all the stages of the pipeline that creates knowledge from data. The problems start during data acquisition because much data is not natively in a structured format and transforming such content into a structured format for later analysis is a major challenge [2]. One example of relevant data in

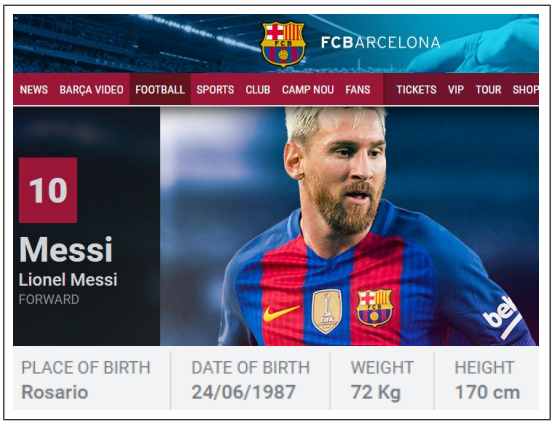

Fig. 1: A real-world entity-page.

Table 1: An output example of the data extraction.

Attribute	Value
Name	Lionel Messi
Position	FORWARD
Number	10
Place of birth	Rosario
Date of birth	24/06/1987
Weight	72 Kg
Height	170 cm

a non-structured format is observed in template-based entity-pages. A template-based entity-page is a page, generated by a server-side template, which publishes data that describe an entity of a particular type. Figure 1 shows a fragment of a real-world entity-page of the Official FC Barcelona Web Site[4]. This entity-page publishes some attributes (*name*, *position*, *number*, among others) of an entity of the *soccer player* type. Extracting data from entity-pages is a strategic task for data-driven companies. For example, Google and Bing use data about entities to provide direct answers. Siri Voice Assistant uses data about entities to answer user queries. Other examples include the use of data about entities to analyze social media, search the deep web, and recommend products [15].

This paper focuses on the problem of extracting attribute values published in template-based entity-pages. This problem involves several challenges because: (i) the presentation formats and the set of attributes that form an entity are subject to a set of variations [12]; (ii) there are several distinct ways of publishing attribute values in template-based entity-pages (horizontal tables, vertical tables, horizontal lists, vertical lists, DIVs, SPANs, etc.); and (iii) the task of discovering the template-based entity-pages in the sites is not entirely precise [21], then the set of entity-pages of a site provided as input to a data extraction method can contain noise pages. Table 1 presents an output example of a data extraction method that receives, as input, the entity-page shown in Figure 1.

The literature provides several methods to extract attribute values from template-based entity-pages. A significant portion [9, 19, 28, 18, 5] of these methods exploit traversal graphs of DOM (Document Object Model) trees that are mapped to XPath expressions for data extraction. Despite the recent popularization of the NoSQL (Not only SQL) databases and the advancement in research and development of graph databases and graph query languages, we did not find in the literature a method that uses a graph query language over

[4] https://www.fcbarcelona.com/football/first-team/staff/players/2016-2017/messi/

a graph database for data extraction. Graph query languages are more robust than XPath because they allow traversing, querying and updating the graph, while XPath is a language specific for traversing DOM trees.

In this paper, we propose a novel class of data extraction methods inspired by the graph databases and the graph query languages. Our method, called **Orion**, uses a declarative graph query language to learn how to extract the attribute values in the template-based entity-pages of a site and carry out the extraction, so its specification is decoupled from its implementation. In particular, **Orion** automatically generates a set of Cypher queries over a graph database so that each query extracts the values of an attribute in all the entity-pages of a site.

We decided to model the problem as queries in a graph database because current graph databases are parallelizable, have an efficient distribution of index and memory usage, and have efficient graph algorithms [11, 4]. We express the queries in the graph database using the Cypher language [1], a declarative graph query language that allows for expressive and efficient querying and updating of the graph store. Holzschuher and Peinl [20] argue that Cypher is a promising candidate for a standard graph query language because: (i) from a readability and maintainability perspective, Cypher seems well suited since its syntax is quite easy to understand for developers familiar with SQL; and (ii) for data with inherent graph-like structures, the code is additionally more compact and easier to read than SQL code.

We carried out experiments on a dataset with more than 145k web pages from different real-world websites of a wide range of entity types. **Orion** reached 98% of F1. We compared our method with one state-of-the-art method: *Trinity* [26]. Our method outperformed *Trinity* with a gain in terms of F1 of 5%.

This study serves at least the following purposes: (i) to provide a concise specification of the Cypher queries that extract the attribute values from template-based entity-pages of a site; (ii) to define an effective and automatic technique that uses Cypher instructions to generate the Cypher queries that extract the attribute values; and (iii) to show the effectiveness of our method through experiments. In our point of view, one important advantage of our method is to use the same declarative language (Cypher) for all the stages that are involved in the data extraction process.

This paper is structured as follows. Section 2 discusses related work. Section 3 defines the key concepts that are on the basis of our method. Section 4 presents the **Orion** method. Section 5 carries out experiments to determine the effectiveness of the **Orion** method and compares it with the baseline. Section 6 concludes the paper and makes recommendations for further studies in this area.

2 Related work

Three of the most popular data extraction methods are *RoadRunner* [13], *ExAlg* [6], and *DEPTA* [27]. *RoadRunner* and *ExAlg* learn a regular expression that models the template used to generate the input documents, while *DEPTA* exploits a partial tree alignment algorithm. Inspired by these methods, other

methods were proposed aiming to fulfill specific gaps. A recent example is the *Trinity* method, which receives a set of template-based entity-pages of a site and returns the attribute values published in these pages. Whenever *Trinity* finds a shared pattern among the entity-pages, it partitions them into the prefixes, separators, and suffixes and analyses the results recursively, until no more shared patterns are found. Prefixes, separators, and suffixes are arranged in a tree that is traversed to build a regular expression with capturing groups that represents the template that was used to generate the entity-pages. *RoadRunner*, *ExAlg*, *DEPTA* and *Trinity* are unsupervised, i.e., they learn rules that extract as much prospective data as they can, and the user then gathers the relevant data from the results. **Orion** is also unsupervised, but it uses declarative languages while *RoadRunner*, *ExAlg*, *DEPTA* and *Trinity* use imperative languages. Declarative languages are more intuitive than imperative languages because declarative languages focus on what the computer is to do, and the imperative languages focus on how the computer should do it, i.e., the declarative languages get away from "irrelevant" implementation details [25].

The most methods that use declarative languages for data extraction exploit traversal graphs of DOM trees that are mapped to XPath expressions [9, 19, 28, 18, 5]. **Orion** also exploits traversal graphs of DOM trees, but it uses Cypher expressions. Cypher is more robust than XPath because it allows traversing, querying and updating the graph, while XPath is a language specific for traversing DOM trees. These features of the Cypher language allow it to be used in all the stages of the process of data extraction. Badica *et al.* [7] propose a data extraction method (*L-wrapper*) that has declarative semantics. *L-wrapper* employs inductive logic programming systems for learning extraction rules, and XSLT technology for performing the extraction. **Orion** employs Cypher for both leaning extraction rules (that are Cypher queries) and performing the extraction. The *W4F* toolkit [24] consists of a retrieval language to identify Web sources, a declarative extraction language (the HTML Extraction Language) to express robust extraction rules and a mapping interface to export the extracted information into some user-defined data-structures. Users guide the generation of the extraction rules using a visual extraction wizard. On the other hand, **Orion** does not require the user intervention to learn the extraction rules.

Other works have goals or strategies that are similar to those of **Orion**, but they are not directly comparable. *WebTables* [10] and *ListExtract* [16] extract data published in HTML tables and lists, respectively. *DeepDesign* [22] extracts data from web pages while browsing. Users guide the extraction process of *DeepDesign* by performing a labeling of fields within an example record and can fine-tune the process as it runs based on an incremental, real-time visualization of results. *ClustVX* [17] extracts attribute values from a page that describes several entities. *SSUP* [21], which is our previous work, discovers entity-pages on the web by combining HTML and URL features. *WADaR* [23] is a post-processing strategy to eliminate noise in the extracted values. **Orion** is more generic applicable than *WebTables* and *ListExtract* since it does not require the data to be arranged in a limited number of specific patterns that frequently

occur on the Web. Moreover, 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 [22]. **Orion** is unsupervised, as opposed to *DeepDesign* that requires the user to label the web page and monitor the entire process of data extraction. **Orion** is effective in extracting attribute values from pages that describe a single entity (entity-pages), as opposed to *ClustVX*. A study [14] showed that entity-pages are among the most common forms of structured data available on the web. *SSUP* and *WADaR* are complementary to **Orion** since *SSUP* finds the entity-pages, **Orion** extracts their attribute values, and *WADaR* eliminates noise in the extracted values.

3 Preliminary Definitions

An `entity` is an object, which is described by its attributes (e.g. *name*, *weight*, and *height*). There are many categories of entities, called `entity types`. A `template-based entity-page` is a web page that describes an entity of a particular type and is dynamically generated by populating fixed HTML templates with content from structured databases. For example, Figure 1 presents a template-based entity-page that describes an entity of the *soccer player* type.

S-graph. Since we are using a graph database, we represent a set of template-based entity-pages of a site as a disconnected graph, called `S-graph`. The S-graph is composed of trees, one for each entity-page. The tree of an entity-page is a modified version of the DOM tree. Our modified version differs from the DOM tree because it only includes tag nodes and textual nodes, as well as, it stores the following properties of each node: (i) *value* - the name of a tag node or the text of a textual node; (ii) *url* - the URL of the entity-page; (iii) *type* - the literal "*TAG*" for tag nodes and the literal "*TEXT*" for textual nodes; (iv) *position* - the information of the position of the node related to its siblings; and (v) *function* - the literal "*TEMPLATE*" for textual nodes that are part of the template of the entity-pages, the literal "*DATA*" for other textual nodes and the absence of this property for tag nodes.

Definition 1. *An `S-graph` is a pair (N, E), where N is a set of nodes and E is a set of relationships (edges) between nodes. There is a node in the S-graph for each tag node or textual node of each template-based entity-page. There is a relationship between two nodes (n_x and n_y) in the S-graph if n_y is a child of n_x in the DOM tree of the entity-page. Each node has the properties: value, url, type, position, and function (except for tag nodes).*

For example, Figure 2 shows the HTML of the entity-page presented in Figure 1. Figure 3 presents the visual representation of the S-graph of the site that contains the entity-page given in Figure 1. We highlighted data nodes using bold face and template nodes using italic.

Cypher wrapper. In web data extraction, a `wrapper` is a program that extracts structured data from a collection of web pages [23]. From a collection of web

```
<HTML>
  …                                                    611      612
    <DIV>
        <DL><DT>PLACE OF BIRTH</DT><DD>Rosario</DD></DL>
        <DL><DT>DATE OF BIRTH</DT><DD>24/06/1987</DD></DL>
        <DL><DT>WEIGHT</DT><DD>72 Kg</DD></DL>
        <DL><DT>HEIGHT</DT><DD>170 cm</DD></DL>
    </DIV>
  ...
</HTML>
```

Fig. 2: HTML of the entity-page presented in Figure 1.

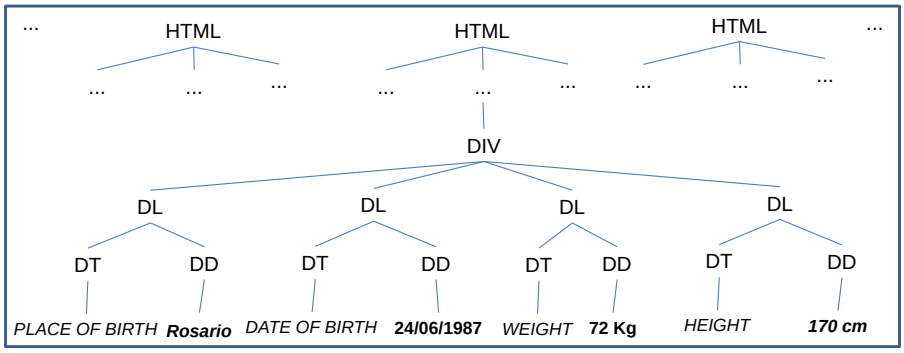

Fig. 3: S-graph of the site that contains the entity-page presented in Figure 1.

pages P, a wrapper produces a relation R with schema Σ. A `relation` R is a set of n-tuples $\{U_1, .., U_n\}$ where n is the number of tuples. A `schema` $\Sigma = \{A_1, .., A_m\}$ is a tuple representing the attributes of the relation, where m is the number of attributes. In our context, the pages in P are entity-pages of a site that follow the same template. We model the wrapper through Cypher queries over the S-graph.

Definition 2. *A `Cypher wrapper` W is a set of queries $\{Q_1, .., Q_m\}$, where each Q_i is a Cypher query over the S-graph that extracts the values of the A_i attribute from all the entity-pages in the S-graph. The application of a Cypher wrapper means to execute its queries over the S-graph and associate the values returned by the queries to the corresponding attribute. The application of a Cypher wrapper W to a collection of entity-pages P produces a relation with schema $\{A_1, .., A_m\}$.*

For example, Figure 7 shows the Cypher query that extracts the *height* attribute in the S-graph given in Figure 3. The application of a Cypher wrapper to the entity-page presented in Figure 1 returns the tuple shown in Table 1.

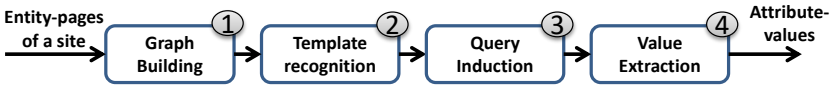

Fig. 4: Overview of the **Orion** method.

This paper focuses on the challenge of inducing a Cypher wrapper from a collection of template-based entity-pages of a site. The **Orion** method carries out this induction automatically, as described in the next section.

4 The Orion method

This section describes **Orion**, a Cypher-based Web Data Extractor that extracts attribute values from template-based entity-pages. Figure 4 presents an overview of the **Orion** method. The input is a set of template-based entity-pages of a site. The output is the set of attribute values published in these entity-pages. The **Orion** method has four main stages: (1) *Graph Building* builds an S-graph from the entity-pages provided as input to the method; (2) *Template Recognition* identifies the nodes in the S-graph that are part of the template behind the entity-pages; (3) *Query Induction* induces Cypher queries from the template nodes; (4) *Value Extraction* extracts the attribute values published in the template-based entity-pages by executing the induced Cypher queries over the S-graph. Next subsections describe these stages in details.

4.1 Graph Building

This stage parses the HTML of the entity-pages provided as input to the **Orion** method and builds the S-graph, without the *function* property, which is added in the next stage (Template Recognition). The S-graph is built using the Cypher language. For example, Figure 5 shows the Cypher instructions that insert in the S-graph: (a) a node representing the tag identified in Figure 2 by ID 611; (b) a node representing the textual content identified in Figure 2 by ID 612; and (c) a relationship representing that the 612 node is a child of the 611 node.

```
(a)  CREATE (n {value: "DD", url: "http://... ", type: "TAG", position:2})  RETURN ID(n)      //611
(b)  CREATE (n {value: "Rosario", url: "http://...", type: "TEXT", position:1}) RETURN ID(n) //612
(c)  MATCH (a), (b) WHERE ID(a) = 611 AND ID(b) = 612 CREATE (a) - [r:contains] -> (b)
```

Fig. 5: Examples of Cypher instructions to create a S-graph. These instructions insert into the S-graph: (a) a tag node; (b) a textual node; and (c) a relationship.

For example, if the input set is composed of the entity-pages of the site that contains the entity-page given in Figure 1, this stage builds the S-graph shown in Figure 3. However, the textual nodes are not classified as template (italic) or data (bold) at this stage, since the *function* property is not defined yet.

4.2 Template Recognition

This stage evaluates the nodes of the S-graph and finds the ones that are part of the template behind the entity-pages. These nodes are called template nodes. A `template node` is a textual node with a value that occurs in, at least, δ entity-pages. Other textual nodes are called `data nodes`. Figure 6(a) presents the Cypher instruction that finds the template nodes and sets their *function* property. Line 1 finds all the textual nodes. Line 2 obtains the values (of the textual nodes) that occur in, at least, δ distinct entity-pages. Line 3 adds the values obtained through Line 2 to a list L. Line 4 finds all the textual nodes with a value that belongs to the L list. Line 5 assigns the literal "*TEMPLATE*" to the *function* property of the nodes obtained through Line 4. Figure 6(b) shows the Cypher instruction that finds the data nodes and sets their *function* property. This instruction must be executed after the Cypher instruction that finds the template nodes. In Figure 6(b), Line 1 finds the textual nodes without the *function* property (at this moment, only the template nodes have the *function* property). Line 2 assigns the literal "*DATA*" to the *function* property of the nodes obtained through Line 1.

(a)

```
1. MATCH (a) WHERE a.type= "TEXT"
2. WITH a.value AS v, COUNT(DISTINCT a.url) AS qt WHERE qt >= δ
3. WITH COLLECT(v) AS L
4. MATCH (b) WHERE b.type="TEXT" AND ANY(x IN L WHERE x=b.value)
5. SET b.function= "TEMPLATE"
```

(b)

```
1. MATCH (a) WHERE a.type= "TEXT" AND a.function IS NULL
2. SET a.function= "DATA"
```

Fig. 6: Cypher instructions to set *function* property.

For example, if the input set is composed of the entity-pages of the site that contains the entity-page presented in Figure 1, this stage classifies the textual nodes of the S-graph (illustrated in Figure 3) as template (highlighted in italic) or data (highlighted in bold). After this stage, the S-graph is complete.

4.3 Query Induction

This stage induces the Cypher queries (over the S-graph) that allow us to extract the attribute values published in the entity-pages. For each data node, a breadth-first search finds the closest template node. The distance is the number of nodes between the template node and the data node. The path from the closest template node to the data node is used to build a Cypher query. The query

filters: (i) the *value* property of all the nodes in the path, except the data node because the *value* property of the data node is what we want to extract; (ii) the *position* property of all the nodes in the path; (iii) the *type* property of all the nodes in the path. The query returns the *value* and *url* properties of the data node. The premise behind this stage is that data nodes from different entity-pages that contain the value of the same attribute generate the same query. The queries that are not generated by, at least, γ data nodes (from different entity-pages) are filtered out.

For example, Figure 7 presents the Cypher query generated from the data node with *value="170 cm"* (in the S-graph illustrated in Figure 3). The closest template node to this data node is the template node with *value="HEIGHT"* (there are three nodes between them). This Cypher query specifies a path in the S-graph, which: (i) starts at a node with *value="HEIGHT"*, *type="TEXT"*, and *position=1*; (ii) follows to a node with *value="DT"*, *type="TAG"*, and *position=1*; (iii) follows to a node with *value="DL"*, *type="TAG"*, and *position=4*; (iv) follows to a node with *value="DD"*, *type="TAG"*, and *position=2*; and (v) ends at a node with *type="TEXT"* and *position=1*. This Cypher query returns the *value* and *url* properties of the data node.

```
MATCH (a)--(b)--(c)--(d)--(e)
WHERE a.value="HEIGHT" AND a.type="TEXT"  AND a.position=1
   AND b.value="DT"    AND b.type="TAG"   AND b.position=1
   AND c.value="DL"    AND c.type="TAG"   AND c.position=4
   AND d.value="DD"    AND d.type="TAG"   AND d.position=2
                       AND e.type="TEXT"  AND e.position=1
RETURN e.url, e.value
```

Fig. 7: An example of a Cypher query.

4.4 Value Extraction

This stage executes the Cypher queries (induced in the previous stage) over the *S-graph* to extract the attribute values published in the entity-pages. The Cypher queries can be applied over the S-graph used to generate them or over another S-graph with other entity-pages of the same template (same site).

For example, the execution of the query given in Figure 7 over the entity-page shown in Figure 1 returns "*170 cm*". The application of all the Cypher queries (induced from the S-graph that is presented in Figure 3) over the entity-page shown in Figure 1 produces the output presented in Table 1.

Table 2: Summary of the datasets.

Dataset	Entity type	Sites	Entity-pages	Attributes
DATASET_S	autos	10	17,923	model, price, engine, fuel_economy
	books	10	20,000	title, author, isbn_13, publisher, publication_date
	cameras	10	5,258	model, price, manufacturer
	jobs	10	20,000	title, company, location, date_posted
	movies	10	20,000	title, director, genre, mpaa_rating
	NBA players	10	4,405	name, team, height, weight
	restaurants	10	20,000	name, address, phone, cuisine
	universities	10	16,705	name, phone, website, type
DATASET_W	books	10	1,315	author, title, publisher, ISBN13, binding, publication date, edition
	stock quotes	10	4,646	last value, day high, day low, 52 wk high, 52 wk low, change %, open, volume, change $
	soccer players	10	5,745	position, birthplace, height, national team, club, weight, birthdate, nationality, number
	videogames	10	12,329	publisher, developer, ESRB, genre
Total		**120**	**148,326**	

5 Experimental Evaluation

The experiments were carried out to evaluate the efficacy of **Orion** and compare it with one state-of-the-art method for data extraction from entity-pages.

5.1 Experimental Setup

Datasets. The experiments were carried out in two datasets ($DATASET\_S$[5] and $DATASET\_W$[6]). Each dataset is a collection of real-world web entity-pages categorized per entity type. These datasets include a ground truth that describes the attribute values in each entity-page. We chose these datasets because they have been used in the literature [19, 9, 23] to evaluate different web data extraction methods. Table 2 summarizes the datasets.

Metrics. We used standard measures such as precision, recall, and F1 [8]. The extracted values were compared with the values in the ground truth to compute the number of true positives (TP), false negatives (FN), and false positives (FP), since this allowed us to assess precision as $P = \frac{VP}{VP+FP}$, recall as $R = \frac{VP}{VP+FN}$, and F1 as $F1 = 2\frac{P \times R}{P+R}$. The Student's t-test [3] with the standard significance level ($\alpha = 0.05$) was employed to determine whether the difference between the results of two methods was statistically significant.

Baselines. We chose *Trinity* as our baseline because it is: (i) a state-of-the-art method for data extraction from template-based entity-pages; (ii) unsupervised;

[5] Available on http://swde.codeplex.com/.
[6] Available on http://www.dia.uniroma3.it/db/weir/.

(iii) not restricted to specific HTML patterns (e.g. HTML tables and lists); (iv) carried out experiments comparing it with four baselines (including RoadRunner [13]) and outperformed all them in terms of effectiveness.

Methodology. The results of *Trinity* were obtained through the prototype provided by the authors[7]. *Trinity* has two parameters (min and max), which represent the minimum and maximum size respectively of the shared patterns for which the algorithm searches. We used the parameter configuration defined in [26], i.e., $min = 1$ and $max = 0.05 \times m$, where m denotes the size in tokens of the smallest entity-page of the site. The **Orion** method has two parameters: (i) δ - the number of entity-pages that a value must occur to be considered part of the template; and (ii) γ - the pruning threshold of the generated queries. We tested the values 10%, 20%, .., 100% to the δ and γ parameters on the *book* type of $DATASET\_W$. The configuration that delivered the highest F1 was using $\delta = 60\%$ and $\gamma = 30\%$. This configuration was employed in all the experiments because it was able to achieve the highest degree of efficacy.

We used the Neo4j 3.0.6 [1] to store the S-graph because it is a highly scalable native graph database. However, another graph database that supports Cypher can be used. Another graph query language can also be used. In this case, the queries presented in this paper must be mapped to the new language. We replace the literals "TEXT", "TAG", "TEMPLATE", and "DATA" with integers.

Orion and *Trinity* are unsupervised techniques, which means that it is the user who has to assign a semantic label to each extraction rule. An extraction rule is a Cypher query in **Orion** and a capturing group in *Trinity*. We adopted the same methodology as *Trinity* [26] to discover the extraction rule that extracts each attribute in the ground truth. Specifically, we found the extraction rule that is the closest to each attribute. To do so, we compared the values extracted by each extraction rule to every attribute in the ground truth. Given an attribute in the ground truth, we considered that the precision and recall to extract them corresponds to the extraction rule (Cypher query or capturing group) with the highest F1 measure.

5.2 Orion *versus* *Trinity*

In this experiment, we aim to answer the following question: *which method is more effective:* ***Orion*** *or* Trinity*?* We evaluate the efficacy through the F1 metric. The higher the F1 value, the greater the degree of efficacy. Table 3 presents the recall, precision, and F1 produced by the methods in each entity type of the two datasets. On average, **Orion** and *Trinity* reached an F1 of 0.98 and 0.93, respectively. The mean gain of **Orion** in terms of F1 was 5%. The scale of the web translates this percentage into a large number of attribute values that can be correctly extracted. The t-test shows that this gain is statistically significant because the p-value (3.12×10^{-7}) is less than the level of significance ($\alpha = 0.05$).

[7] Available at http://www.tdg-seville.info/Download.ashx?id=341.

Table 3: Comparison between **Orion** and *Trinity*.

Dataset	Entity type	Recall		Precision		F1		
		Trinity	Orion	Trinity	Orion	Trinity	Orion	%
DATASET_S	autos	0.96	0.99	0.96	1.00	0.96	0.99	3
	books	0.90	0.97	0.92	1.00	0.90	0.98	**9**
	cameras	0.96	0.91	0.98	1.00	0.97	0.95	-2
	jobs	0.96	0.92	0.97	0.98	0.96	0.94	-3
	movies	0.97	0.97	0.99	1.00	0.97	0.98	1
	NBA players	0.95	1.00	0.95	1.00	0.95	1.00	5
	restaurants	0.91	0.96	0.91	1.00	0.91	0.98	7
	universities	0.93	0.97	0.93	1.00	0.93	0.98	5
DATASET_W	books	0.86	0.98	0.86	0.99	0.86	0.99	**15**
	stock quotes	0.92	1.00	0.95	1.00	0.92	1.00	8
	soccer players	0.92	0.96	0.91	0.97	0.91	0.97	6
	videogames	0.93	0.97	0.94	1.00	0.93	0.98	5
Mean		0.93	0.97	0.94	0.99	0.93	0.98	5

* 148,326 entity-pages from 120 real-world web sites

Orion outperformed *Trinity* because, as opposed to *Trinity*, **Orion** is not affected by: (i) missing attributes (i.e. attributes where the values are only published in some of the entity-pages of the site) that are published in simple elements (i.e. when the element has only one textual node); (ii) attributes published in horizontal tables (i.e. the header is in the first line of the table); and (iii) noise pages (i.e. pages in the input set that follow a template different from the most entity-pages in the set). *Trinity* mistakenly assumes that each value of a missing attribute is a part of the value of the immediately preceding attribute in the HTML of the page. *Trinity* is not able to extract correctly the attribute values that are published in the middle columns of the horizontal tables because these attribute values have the same prefix and suffix. In this case, *Trinity* erroneously regards values from different attributes as being the values of a same multivalued attribute. A noise page in the input set makes *Trinity* creates a wrong ternary tree and, consecutively, generates an erroneous regular expression, which affects its efficacy. This behavior occurs because *Trinity* is based on patterns shared by all the entity-pages provided as input. However, the task of collecting the entity-pages of a site is not trivial and, eventually, some non-entity-pages are collected together. Another situation verified in the datasets is the occurrence of entity-pages (that describe entities of the same type) within the same site, but that follow a different template. For example, in the *job* type, there is a site where the attributes values are published: (i) on the left in some entity-pages; (ii) on the right in some entity-pages; and (iii) on the center in other entity-pages.

Orion outperformed *Trinity* in 83% of the entity types. **Orion** obtained the highest gain regarding F1 in the *book* type (15% in the $DATASET\_W$ dataset) and (9% in the $DATASET\_S$ dataset). This type contains several missing attributes. *Trinity* outperformed **Orion** in the *camera* and *job* types.

However, the difference did not exceed 3%. Section 5.3 explains the cases of failure of the **Orion** method that caused this negative difference.

It is important to note that we performed exhaustive experiments with **Orion** and *Trinity*, which included 148,326 entity-pages from 120 real-world websites from different entity types. On average, each site has 1,236.00 entity-pages. The experiments performed by the authors of *Trinity* included, on average, 37.89 entity-pages per site and the maximum number of entity-pages of a site was 252.

The results show that **Orion** has an efficacy that is significantly higher than *Trinity* because on average it achieved the highest F1 value. The reason for this is that **Orion**, as opposed to *Trinity*, is not affected by the noise pages, the attribute values published in the horizontal tables or the missing attributes. Hence **Orion** is able to extract correct values in more entity-pages and to reduce the number of values that are erroneously extracted. Moreover, the implementation of the **Orion** method is based on a declarative language while the implementation of the *Trinity* method is based on an imperative language. Declarative languages are more intuitive than imperative languages [25].

5.3 Analysis of the cases of failure

To better understand our results, we have analyzed the cases of failure of the **Orion** method. This analysis might be very useful for developers of new methods to extract attribute values from template-based entity-pages because it shows the difficulties of handling the particularities of this kind of pages.

The first case of failure is when the values of an attribute are published in the entity-pages of a site in textual nodes that also contain other information. In some sites, the label and the value of the attribute are published in the same textual node. In other sites, the values of two attributes are published in the same textual node. **Orion** has textual node granularity, i.e., it does not segment the content of a textual node. Therefore, **Orion** extracts noise with the values of the attribute. A possible solution is to apply a post-processing to segment the extracted values and eliminate noise, as carried out by Ortona *et al.* [23].

The second case of failure occurred in sites where the attribute values were published in mixed elements (elements with two or more textual nodes). **Orion** extracted incorrect values when the attribute values were published in mixed elements and one of the following variations occurred: (i) missing attributes - attributes where the values are only published in some of the entity-pages of the site; (ii) multivalued attributes - more than one value is published for the same attribute of the same entity; and (iii) multiordering attributes - the attributes are published in a different order in the entity-pages of the site. These variations did not affect the efficacy of the **Orion** method when the attribute values were published in simple elements (i.e. elements with only one textual node). A possible solution is to apply a pre-processing to mixed elements.

The third case of failure occurred when the values of an attribute are published in the entity-pages of a site using different tags. Let v be the node that contains the value of an attribute and w be the closest template node to v. If the variation in the template occurs in the subtree rooted at the lowest common

ancestor of v and w, then the recall of **Orion** is affected. A possible solution to these cases is to select complementary Cypher queries for each attribute.

6 Conclusion

In this paper, we proposed **Orion**, a novel method to extract attribute values from template-based entity-pages, which is inspired by the graph databases and graph query languages. Our method represents the template-based entity-pages as a graph, named S-graph. The **Orion** method automatically induces a set of Cypher queries over the S-graph where each query extracts the values of an attribute from all the entity-pages of a site. The **Orion** method has declarative semantics, i.e., its specification is decoupled from its implementation. Moreover, the Cypher language is more robust than XPath because it allows traversing, querying and updating the graph, while XPath is a language specific for traversing DOM trees. Our experiments, performed on 120 real-world websites, proved that the **Orion** method achieves very high precision, recall, and F1. Our method outperformed a state-of-the-art method (*Trinity*) regarding F1. As future work, we plan to: (i) define a pre-processing to handle with mixed elements; (ii) a post-processing to eliminate noise from the extracted values; (iii) evaluate the processing time of our method; and (iv) compare the **Orion** method with other methods based on declarative languages (e.g. [9, 19, 28, 18, 7, 5, 24]).

References

1. Neo4j Tecnology. The Neo4j Manual v2.3.3, March 2016. `http://neo4j.com/docs/stable/index.html`.
2. D. Agrawal et al. Challenges and Opportunities with Big Data – A community white paper developed by leading researchers across the United States, March 2012. `http://cra.org/ccc/docs/init/bigdatawhitepaper.pdf`.
3. T. Anderson and J. Finn. *The New Statistical Analysis of Data.* Springer texts in statistics. Springer, 1996.
4. R. Angles and C. Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, Feb. 2008.
5. T. Anton. Xpath-wrapper induction by generating tree traversal patterns. In M. Bauer, B. Brandherm, J. Fürnkranz, G. Grieser, A. Hotho, A. Jedlitschka, and A. Kröner, editors, *Lernen, Wissensentdeckung und Adaptivität (LWA) 2005, GI Workshops, Saarbrücken, October 10th-12th, 2005*, pages 126–133. DFKI, 2005.
6. A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 337–348, New York, NY, USA, 2003. ACM.
7. C. Badica, A. Badica, E. Popescu, and A. Abraham. L-wrappers: concepts, properties and construction. *Soft Comput.*, 11(8):753–772, 2007.
8. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology Behind Search.* Addison Wesley Professional, 2011.
9. M. Bronzi et al. Extraction and integration of partially overlapping web sources. *VLDB Endow.*, 6(10):805–816, 2013.

10. M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: Exploring the power of tables on the web. *VLDB Endow.*, 1(1):538–549, 2008.
11. R. Cattell. Scalable sql and nosql data stores. *SIGMOD Rec.*, 39(4):12–27, May 2011.
12. C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan. A survey of web information extraction systems. *IEEE Trans. on Knowl. and Data Eng.*, 18(10):1411–1428, Oct. 2006.
13. V. Crescenzi and G. Mecca. Automatic information extraction from large websites. *J. ACM*, 51(5):731–779, Sept. 2004.
14. E. Crestan and P. Pantel. Web-scale table census and classification. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 545–554, New York, NY, USA, 2011. ACM.
15. O. Deshpande et al. Building, maintaining, and using knowledge bases: A report from the trenches. In *SIGMOD*, 2013.
16. H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting relational tables from lists on the web. *VLDB*, 20(2):209–226, 2011.
17. T. Grigalis. Towards web-scale structured web data extraction. In *WSDM*, 2013.
18. P. Gulhane, R. Rastogi, S. H. Sengamedu, and A. Tengli. Exploiting content redundancy for web information extraction. In *WWW*, 2010.
19. Q. Hao, R. Cai, Y. Pang, and L. Zhang. From one tree to a forest: A unified solution for structured web data extraction. In *SIGIR*, 2011.
20. F. Holzschuher and R. Peinl. Performance of graph query languages: Comparison of cypher, gremlin and native access in neo4j. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13, pages 195–204, New York, NY, USA, 2013. ACM.
21. E. Manica, R. Galante, and C. F. Dorneles. SSUP - A url-based method to entity-page discovery. In S. Casteleyn, G. Rossi, and M. Winckler, editors, *Web Engineering, 14th International Conference, ICWE 2014, Toulouse, France, July 1-4, 2014. Proceedings*, volume 8541 of *Lecture Notes in Computer Science*, pages 254–271. Springer, 2014.
22. A. Murolo and M. C. Norrie. Revisiting web data extraction using in-browser structural analysis and visual cues in modern web designs. In A. Bozzon, P. Cudré-Mauroux, and C. Pautasso, editors, *Web Engineering - 16th International Conference, ICWE 2016, Lugano, Switzerland, June 6-9, 2016. Proceedings*, volume 9671 of *Lecture Notes in Computer Science*, pages 114–131. Springer, 2016.
23. S. Ortona, G. Orsi, T. Furche, and M. Buoncristiano. Joint repairs for web wrappers. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*, pages 1146–1157. IEEE Computer Society, 2016.
24. A. Sahuguet and F. Azavant. Wysiwyg web wrapper factory (w4f). In *Proceedings of WWW Conference*, 1999.
25. M. Scott. *Programming Language Pragmatics*. Elsevier Science, 2015.
26. H. A. Sleiman and R. Corchuelo. Trinity: On using trinary trees for unsupervised web data extraction. *TKDE*, 26(6):1544–1556, 2014.
27. Y. Zhai and B. Liu. Web data extraction based on partial tree alignment. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 76–85, New York, NY, USA, 2005. ACM.
28. J. Zhang, C. Zhang, W. Qian, and A. Zhou. Automatic extraction rules generation based on xpath pattern learning. In *Proceedings of the 2010 International Conference on Web Information Systems Engineering*, WISS'10, pages 58–69, Berlin, Heidelberg, 2011. Springer-Verlag.