Derzis: A Path Aware Linked Data Crawler

André Fernandes dos Santos
CRACS & INESC Tec LA, Faculty of Sciences, University of Porto, Portugal

José Paulo Leal
CRACS & INESC Tec LA, Faculty of Sciences, University of Porto, Portugal

Abstract

Consuming Semantic Web data presents several challenges, from the number of datasets it is composed of, to the (very) large size of some of those datasets and the uncertain availability of querying endpoints. According to its core principles, accessing linked data can be done simply by dereferencing the IRIs of RDF resources. This is a light alternative both for clients and servers when compared to dataset dumps or SPARQL endpoints. The linked data interface does not support complex querying, but using it recursively may suffice to gather information about RDF resources, or to extract the relevant sub-graph which can then be processed and queried using other methods. We present Derzis[1], an open source semantic web crawler capable of traversing the linked data cloud starting from a set of seed resources. Derzis maintains information about the paths followed while crawling, which allows to define property path-based restrictions to the crawling frontier.

2012 ACM Subject Classification Information systems → Web crawling; Information systems → Structure and multilingual text search

Keywords and phrases Semantic web, linked open data, RDF, crawler

Digital Object Identifier 10.4230/OASIcs.SLATE.2021.2

Supplementary Material *Software (Source Code)*: `https://github.com/andrefs/derzis`
archived at `swh:1:dir:d35f9b43c9b88955c9d6068fcc15e59c4aba816f`

Funding This work is financed by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020.
André Fernandes dos Santos: Ph. D. Grant SFRH/BD/129225/2017 from Fundação para a Ciência e Tecnologia (FCT), Portugal.

1 Introduction

Two features of the Semantic Web (SW) which make it interesting – being distributed and not needing a globally defined schema – also make accessing it not trivial. Two main questions arise from these features: *how* to access the SW and *what* to access.

The SW is built on top of general purpose technologies and standards such as HTTP and Internationalized Resource Identifiers (IRIs), and a small core of additional specifications, such as RDF(S), OWL and SPARQL. This means that the basic protocols and syntaxes are universal. However, there are several possible approaches when making semantic data available (and, consequently, when consuming semantic data). These usually fall into one of the following categories: (i) downloadable triplesets, (ii) queriable SPARQL endpoints or other SW interfaces, (iii) custom APIs which return RDF data, and (iv) dereferenceable resource IRIs.

The same data is frequently available using multiple methods. The requirements for a Semantic Web application (SWA) will heavily influence which method should be used. Conversely, the methods under which data is available strongly dictate the types of applications which can consume it. A detailed explanation will be provided in the next section.

[1] Available at `https://github.com/andrefs/derzis`.

Dereferencing IRIs, also known as the linked data interface, or the *follow-your-nose* approach, can be used in applications which do not require complex querying support, but simply need to fetch information about a given set of resources. It allows fetching data which is up to date and does not require the resources needed to download and process large tripleset dumps. The level of detail can be increased or decreased by adjusting the depth of recursion, or by fine tuning which semantic links should be followed and which IRIs should be dereferenced. It can also be used as as preliminary step on a more complex pipeline, reducing a possibly very large semantic graph (composed of triples originating for many connected triplesets) to a sub-graph containing only the relevant data. This data can then be more easily manipulated and queried.

In this paper we present Derzis, an open source semantic web crawler which, given a set of seed resources, traverses the linked data cloud, following the semantic links extracted from the triples resulting from dereferencing IRIs. Parameters such as maximum depth and property path restrictions can be defined to limit the graph traversal.

This paper is structured as follows. In Section 2 we cover concepts relevant to this work. Section 3 describes and compares other semantic web crawlers and tools following a similar methodology. Our approach is detailed in Section 4, where we provide an example of the crawling algorithm and its formal definition. Section 5 describes the system architecture. Section 6 describes preliminary results obtained using Derzis. Sections 7 and 8 describe, respectively, future developments planned for Derzis and conclusions for this work.

2 Background

The origins of the Semantic Web as a research field can be traced back to the early 2000s, with specifications such as RDF [4] and RDFS [17] being published a little earlier. In a 2001 article in Scientific American [3], Tim Berners-Lee et al. envisioned a web of semantically annotated data, intelligible to machines, in which automated agents could roam autonomously to perform tasks. Despite undeniable contributions to the fields of data sharing, discovery, integration and reuse, the semantic web as a research field has been the subject of much criticism, as compiled by Hogan [13]. The original vision is still far from fulfilled.

The history of this field can be split into three phases [12]: the ontology phase, the linked data phase and the knowledge graphs phase. In the beginning the focus was on developing formal ontologies with the goal of sharing and integrating data. Then the interest shifted to the publication of datasets linked through the reuse of IRIs. More recently, we have seen the industry become interested in semantic technologies, which led to the development of their own knowledge graphs, e.g. Google [22], Microsoft [9], Amazon [16]. In time, more open versions became available, e.g. Wikidata [24] and DBpedia [18]. Nowadays, the semantic web is composed of artifacts inherited from these three phases: domain-specific and general scope ontologies, more shallow vocabularies, linked datasets and knowledge graphs.

Semantic Web Applications can access the semantic web in a number of ways. Choosing one method over another may severely limit what your application can do or how it operates. The same is to say that different applications will have different needs in what regards to consuming semantic data.

One way is to download the knowledge graphs in RDF format in advance. This requires the computational resources to deal with very large files, and the possibility of data becoming eventually outdated. Applications relying on queriable endpoints (usually SPARQL or Triple Pattern Fragments [23]) avoid these shortcomings. However, they become dependent on servers which must be available and capable of responding to potentially complex queries.

Integrating information from different sources may be difficult. Custom APIs are similar, but present one additional disadvantage: they require applications to be even more tightly coupled with them, as the SWA must have built-in the knowledge of how to interact with that non-standard endpoint.

The methods previously described are all better suited for applications where the triplesets to be consumed are known beforehand, either because the triplesets need to be downloaded, their queriable endpoints must be known and integrated with each other, or because the applications need to understand the syntax of custom APIs.

The linked data interface (also known as the *follow-your-nose* approach [25]) allows gathering data about a resource simply by dereferencing (i.e. performing an HTTP request to) its IRI, and parsing the resulting RDF content. This behavior is explicitly stated in the Linked Data principles [2]. This method is light for the servers, as there is no need of software to parse and perform complex queries, and can even be statically served. It is also light for the users as it prevents the need to deal with large files or outdated data, and can reasonably be expected to work more or less uniformly across domains.

A web crawler is a program capable of accessing HTML pages through their URIs, parsing them, extracting hyperlinks and recursively repeating the process [19]. These are typically used by search engines to index web page contents and calculate their relevance. To avoid having crawlers accessing unwanted parts of their sites, domains can indicate which URLs are off-limits by adding directives to their `robots.txt` file according to the Robots Exclusion Protocol [14]. The `Crawl-Delay` directive is commonly used to rate-limit crawlers performing multiple requests. Common crawling policies further specify which pages should be visited, when they should be re-visited, the frequency of the requests and the degree of parallelization of the crawler [5].

An RDF resource is identified by an IRI. An RDF triple specifies a semantic link between two RDF resources (the subject and object of the triple) using another RDF resource (the predicate). Multiple RDF triples make an RDF graph, where the nodes are the subjects and objects of the triples, and the edges are the predicates[2]. According to the Linked Data principles, accessing the IRI of an RDF resource should return an RDF representation of that resource – i.e. triples relevant to that resource's understanding. This usually means triples where the resource appears either as the subject or the object.

A semantic web crawler (SWC) is the application to the semantic web of the web crawler concept. It starts with the IRIs of RDF resources, and *dereferences* them – i.e. fetches the IRIs contents by performing HTTP requests. Instead of a simple HTML page, for each resource it expects in return an RDF document (which can also be an HTML page with RDFa metadata). This document contains triples representing semantic links between the resource and other nodes in the RDF graph. The SWC parses the RDF document and repeats the process for the new resources found. The crawling process can be limited in a number of ways, most commonly by defining which resources should be dereferenced and/or the maximum crawl depth.

[2] Actually, a resource appearing in the predicate position of a triple can be (and frequently is) used as the subject or object of another. This means that the edges of an RDF graph can simultaneously also be nodes.

3 Related Work

There are several semantic web crawlers described in the literature, some of which are publicly available. Their core functionality is similar: following links in the semantic web. However, they differ in more specific features, such as sorting what should be crawled first (e.g. depth-first or breadth-first), white or black-listing resources to be dereferenced, understanding basic RDFS and OWL properties (also referred to as RDFS-Plus [7]) or using domain knowledge to inform the crawl process. Next we present a brief description of notable examples of SWCs. A comparison of their features can be found in Table 1.

Crawler-LD is a semantic web crawler designed to help linked data publishers finding vocabulary terms which can be interlinked with their own data [20]. Given an initial set T of terms, for each Crawler-LD searches the DataHub catalogue of Linked Data triplesets searching terms which are directly or transitively related to it either by sub-classing or using properties such as `owl:sameAs` or `owl:equivalentClass`. Freire and Silva (2020) describe a crawler which is initialized with knowledge of the domain in which it is used [10]. This knowledge is then used to decide which resources and triples to follow and which ones to ignore. Similarly, Bedi et al. (2012) describe a semantic focused crawler which is also enriched with domain knowledge [1]. In this case, semantic distance metrics between the crawled resources and the ones in the domain knowledge are used to prioritize the crawl list. LDSpider is a generic crawler for linked data [15] which is able to handle a large number of formats, perform breadth-first or depth-first crawls, include schema information and limit the set of properties which are followed, or the prefix of resources crawled, or their maximum depth. In KRaider [6], the maximum resource depth cut-off ignores the properties `owl:sameAs`, `owl:equivalentClass` and `owl:equivalentProperty`. Squirrel [21] is a distributed crawler capable of handling RDF formats, compressed files, HTML with RDFa or Microformats, SPARQL endpoints and CKAN addresses. Crawl priority can be defined based on the IP or domain of the resources.

Hartig and Pirrò have proposed a formal foundation to extend to SPARQL that leverages property paths and allows coupling navigation at the data level with navigation on the Web graph [11]. S-Paths [8] is a linked data browser which supports different representations for data. It relies on path characteristics to aggregate data and determine the best view for the results.

Our solution, Derzis, is capable of reducing the size of the graph being crawled without requiring previous domain knowledge. This is achieved by limiting the number of distinct properties in each crawl path. As a result, Derzis focuses on the triples and path expected to contain more relevant information regarding the seed resources.

4 Approach

The goal of this work was the development of a semantic web crawler capable of recursively extracting information from a set of semantic resources. This SWC should allow reducing a large graph to a smaller sub-graph containing the most relevant data regarding the seed resources. The main use case for this is the implementation of graph-based semantic measures. Due to their algorithmic complexity, such measures are impractical to implement in large knowledge graphs, and as such would benefit from the ability of reducing said graphs to more manageable sizes. The base algorithm of this crawler should prove useful in other applications in which there is the need to focus on the relevant parts of a larger graph, e.g. a semantic browser capable of inferring which information should be displayed.

Table 1 Comparison of semantic web crawlers features.

Crawler	Publicly available	Distributed / parallel	Crawl priority	`robots.txt` compliant	RDFS-Plus	Domain knowledge	Crawl limits
Crawler-LD	✘				✔		
Freire and Silva	✘					✔	
Bedi et al.	✘		c			✔	
LDSpider	✔	✔	e	✔	✘	✔	134
KRaider	✘	✔	d		✔	✔	1
Squirrel	✔	✔	ab	✔	✘	✘	12

Crawl priority:
(a) IP
(b) Pay-level domain
(c) Domain knowledge
(d) FIFO
(e) Breadth or depth-first

Crawl limits:
(1) Maximum depth
(2) Black and white lists
(3) Properties followed
(4) Resources prefix

This crawler was developed with the following design goals:

1. recursively dereference RDF graph nodes (subjects and objects),
2. crawl limits based on property path restrictions,
3. parallel and distributed crawling,
4. compliance with `robots.txt` rules.

Next we describe the strategies used to reduce the crawled graph size. Then we present an example of the crawling process with a detailed step-by-step description of the sequence of events that take place. After that we provide a formal definition of the crawling algorithm.

4.1 Reducing the crawled graph

To ensure that priority is given to the relevant parts of the original graph, several strategies are used, which are described next.

Triples where a resource appears as predicate are discarded. Imagine an RDF property `ex:hasColor` which allows specifying the color of a resource, e.g. `ex:Sky ex:hasColor ex:Blue`. If `ex:hasColor` was passed to Derzis as a seed resource, it should find relevant triples which characterize it such as `ex:hasColor rdf:type rdf:Property`, `ex:hasColor rdf:subPropertyOf ex:hasPhysicalProperty` and `ex:colorOf owl:inverseOf ex:hasColor`. These triples are *describing* the property. However, triples such as `ex:Snow ex:hasColor ex:White` or `ex:Grass ex:hasColor ex:Green` are not describing `ex:hasColor`, they are *using* it to describe the other resources. For this reason, when dereferencing a resource, triples where it is used as the predicate are discarded.

Crawling follows only subject and object resources. Imagine an RDF resource `ex:Einstein`. Suppose that, when dereferencing it, we obtain the triples `ex:Einsten ex:wonAward ex:NobelPrize` and `ex:Einstein ex:bornIn ex:Ulm`. Recursively dereferencing the properties `ex:wonAward` and `ex:bornIn` would gather data about these properties. This is arguably less relevant for describing the resource `ex:Einstein` than dereferencing the resources it is related to, in this case, `ex:NobelPrize` and `ex:Ulm`.

Crawling can be limited to paths containing a maximum number of distinct properties. Homogeneous paths between resources are more likely to represent implicit hierarchy or a meaningful chain of connections than heterogeneous ones.

Maximum crawling depth can be defined. Resources too far away from the seed resources are less relevant than those which are closest.

4.2 Crawling example

The IRI `http://dbpedia.org/resource/Mozzarella` is the identifier for Mozzarella cheese on DBpedia. Next we describe step by step an example of the crawling process of Derzis, using this IRI as the only seed. We will limit the crawling to paths with a maximum depth (`maxPathDepth`) of 4 and no more than 2 distinct properties (`maxPathProps`). Figure 1 displays the graph being built by the crawling process. Table 2 presents the triplesets found when dereferencing each resource IRI. For clarity purposes, the triplesets have been truncated and displayed in Turtle format. Additionally, we omit several operations related with job distribution, database housekeeping and request rate-limiting. We also implicitly assume the following prefix definitions:

```
@prefix dbr: <http://dbpedia.org/resource/>
@prefix dbp: <http://dbpedia.org/property/>
@prefix dbo: <http://dbpedia.org/ontology/>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix owl: <http://www.w3.org/2002/07/owl#>
```

1. The seed IRI `http://dbpedia.org/resource/Mozzarella` is added to Derzis as a seed Resource. This triggers the creation of a Path, containing one single node (`dbr:Mozzarell`), and no predicates.
2. The seed resource is then dereferenced. All the triples in which `dbr:Mozzarella` is either the subject or object are added to Derzis. The triple `dbr:Calzone dbo:ingredient dbr:Mozzarella` will lead to the extension of the path. This new path will have `dbr:Mozzarella` as its `seed`, `dbr:Calzone` as its `head`, no other nodes and `dbo:ingredient` as the only `predicate`.
3. The process continues with the dereferentiation of `dbr:Calzone`. The resulting triples will give origin to two new paths. One starts in `dbr:Mozzarella` and reaches `dbr:Salami`, with a total length of 3 nodes and a single predicate (`dbo:ingredient`). Another starts also in `dbr:Mozzarella` but leads to `dbr:Pizza`, also with a length of 3 but containing 2 predicates already: `dbo:ingredient` and `dbo:hasVariant`.
4. Dereferencing `dbr:Salami` will extend the previous path leading here, adding one more predicate (`rdf:type`) and one more node (`owl:Thing`). This new path has reached the maximum number of nodes and distinct predicates. It is marked as finished and, consequently, `owl:Thing` is not dereferenced.
5. The path leading to `dbr:Pizza` is already maxed out on the number of predicates. This means that it can only be extended with triples in which the predicate is either `dbo:ingredient` or `dbo:hasVariant`. From the triples obtained from dereferencing `dbr:Pizza`, `dbr:Pizza dbp:country dbr:Italy` and `dbr:Neapolitan_pizza dbp:type dbr:Pizza` are ignored. It is extended to `dbr:Tomato_sauce` and `dbr:Panzerotti`. Both paths have now also reached the maximum length. They are marked as finished, and these resources are not dereferenced.
6. At this point, there are no more active paths. The crawling process finishes and the crawled graph can be exported.

4.3 Crawling algorithm

Given a set R of seed RDF resources, their linked data graph G_R is the graph obtained by collecting all the triples obtained by dereferencing each resource in R, and recursively applying the same to each resource included in those triples. This potentially results on a

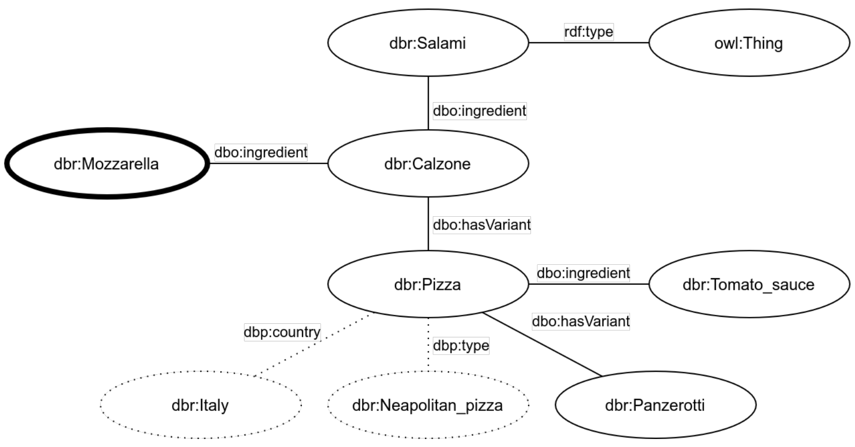

Figure 1 Example of the graph built while crawling.

Table 2 Contents of resources IRIs.

i) `http://dbpedia.org/resource/Mozzarella`

```
dbr:Calzone dbo:ingredient dbr:Mozzarella .
[...]
```

ii) `http://dbpedia.org/resource/Calzone`

```
dbr:Pizza dbo:hasVariant dbr:Calzone .
dbr:Calzone dbo:ingredient dbr:Salami .
[...]
```

iii) `http://dbpedia.org/resource/Salami`

```
dbr:Salami rdf:type owl:Thing .
[...]
```

iv) `http://dbpedia.org/resource/Pizza`

```
dbr:Pizza dbo:ingredient dbr:Tomato_sauce .
dbr:Pizza dbo:hasVariant dbr:Panzerotti .
dbr:Pizza dbp:country dbr:Italy .
dbr:Neapolitan_pizza dbp:type dbr:Pizza .
[...]
```

very large graph (maybe even the whole Linked Data cloud, given its connected nature). We can ignore the properties characterization and classification by dereferencing only the nodes of the graph (subjects and objects of triples). The size of the resulting graph can be further reduced by considering a set of path restrictions PR which determine, for each new node discovered, whether it should be followed (dereferenced) or not. Then $RG_{R,PR}$ is the sub-graph of G_R obtained by traversing it while applying the path restrictions PR. Figure 2 presents a simplified view of the semantic web, with a pair of seed resources (A and B), the linked data graph for A and B (i), the graph reduced by path restrictions (ii), and a disconnected part of the semantic web, unreachable by recursively following links starting from A and B (iii).

Path restrictions require Derzis to maintain information regarding each path being followed during the crawling process: its origin (seed resource), current head (latest resource appended), the total length and total number of distinct properties.

$RG_{R,PR}$ is built iteratively. It is initialized with a set of paths P containing a path for each seed resource. Algorithm 1 presents a simplified pseudocode version of Derzis main behavior (the actual code has small differences mostly regarding optimization of the distributed work). Initially, a path is created for each seed resource. Then, in each iteration, each resource p_{head} which is the head of an active path is dereferenced and marked as *visited*. For each resulting RDF triple, the property and node (either the subject or object, depending on the inverse position occupied by p_{head}) are added to p to create a new path p'. If the path restrictions ($maxPathLength$ and $maxPathProps$) are not met, p' is dropped. Is there is

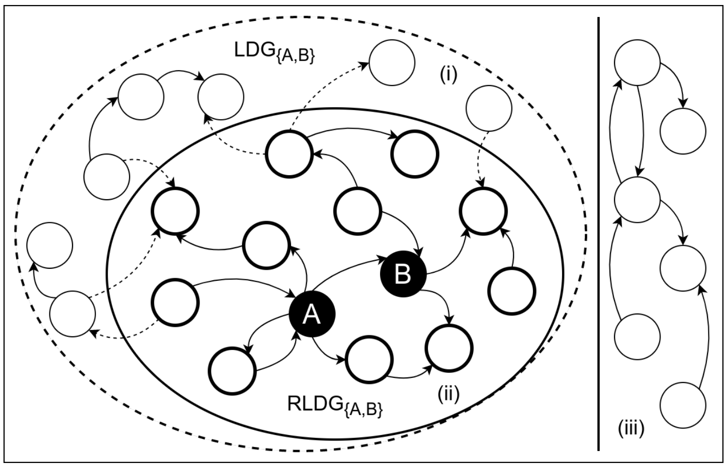

Figure 2 The semantic web and restricted linked data graph of resources A and B.

already a similar path (built with the same properties (or a subset), and with the same or shorter length), p' is dropped. If the resource at the head of the new path p' has already been visited by Derzis, the path is extended using cached information until an unvisited head is reached. The process ends when no more active paths exist. At this point, Derzis can export the restricted graph in RDF format.

Algorithm 1 BUILDRG(P).

```
global $maxPathLength, maxPathProps$
global $R_{visited}$

$P' \leftarrow \varnothing$
$R \leftarrow \{p_{head} : p \in P \wedge p \text{ is active}\}$
if $R = \varnothing$
  then $return$

for each $p_{head} \in R$
do
  $triples \leftarrow dereference(p_{head})$
  $R_{visited} \leftarrow R_{visited} \cup p_{head}$

  for each $node, prop : \{node, prop, p_{head}\} \in triples \vee \{p_{head}, prop, node\} \in triples$
  do
    if $p_{length} \geq maxPathLength \vee prop \notin p \wedge |p_{props}| \geq maxPathProps)$
      then $break$

    $p' \leftarrow p \cup \{prop, node\}$
    if $\exists q : q_{seed} = p'_{seed} \wedge q : q_{head} = p'_{head} \wedge q_{props} \subseteq p'_{props}$
      then $break$

    if $p'_{head} \in R_{visited}$
      then ADDEXISTINGHEAD($p'$)

    $P' \leftarrow P' \cup p'$
    $p \leftarrow finished$

BUILDRG($P'$)
```

5 System architecture

The Derzis tool is built around two main components: the Manager and a pool of Workers. Figure 3 provides a visual representation of Derzis architecture. The Manager is responsible for accessing the database (Figure 3.5), decides what should be crawled next and distributes jobs to the workers (Figure 3.2). Workers handle the tasks of retrieving domain `robots.txt` files and resource IRI contents (Figure 3.3). Both components were written in Node.js; they coordinate with each other exchanging messages over Redis, and the data is persisted on a MongoDB instance. The manager, each of the workers, the database and Redis can all be distributed across different machines.

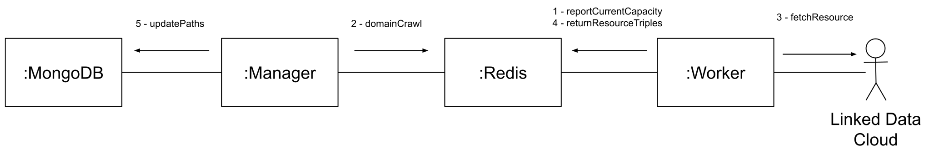

Figure 3 UML Communication diagram of the system architecture.

5.1 Worker

A Worker is an autonomous process which communicates with the Manager over Redis. When it starts, it sends to a configurable channel a message stating the type and number of jobs it is capable of handling (Figure 3.1). This is repeated periodically. Currently these jobs can be either trying to retrieve the `robots.txt` of a domain or dereferencing unvisited resources of a domain. By default, a Worker accepts a maximum of 10 simultaneous `robots.txt` checks and 10 domains with batches of 10 resources each to be dereferenced.

The Worker handles these jobs asynchronously. For each domain, it dereferences each resource sequentially (Figure 3.3), first checking whether the `robots.txt` allows it and what the rate limit is. In the HTTP request that fetches the resource, the Accept header is set to allow several RDF mime types; frequently, however, servers do not comply. In these cases, the Worker checks the `Link` response header in search of alternative URLs (or, if the returned content is HTML, the `<link>` tags). If the Content-Type is still unrecognised an error is raised. The results of each `robots.txt` check and resource dereference are sent back to the Manager also via Redis (Figure 3.4).

5.2 Manager

The Manager starts by posting a message in Redis asking for any Workers listening to report their availability to perform jobs. Anytime it receives an availability message, it retrieves from the database the appropriate number of domains needing their `robots.txt` to be retrieved or resources to be dereferenced. When these jobs are sent to the worker, the Manager marks them with the Worker identifier, making sure that a domain is only worked on by one Worker at a time (preventing multiple concurrent requests to the same domain), and starts a timer for that job. If the Worker does not reply before the timer expires, the domain is reset and made available for other workers.

6 Preliminary results

Derzis is still in early stages of development. For the time being, we do not yet have comparisons of its performance against similar tools or using public benchmarks. Nevertheless, we have already results gathered by running it using a small set of seed resources (5 types of cheese gathered from DBPedia) and combinations of crawling parameters. These were obtained on a laptop with an Intel i7-6500U CPU 2.50GHz and 8GB of memory. The system was configured to spawn 3 workers, each capable of handling 10 `robots.txt` checks and 10 domains with 10 resources each to be dereferenced. Tables 3, 4 and 5 present these results.

Table 3 Total elapsed times for different combinations of *maxPathLength* and *maxPathProps*.

Total elapsed time		Maximum path length 2	Maximum path length 3
Maximum distinct properties	1	44s	2h32m
	2	–	3h12m

Table 4 Total dereferenced resources for different combinations of *maxPathLength* and *maxPathProps*.

Total dereferenced resources		Maximum path length 2	Maximum path length 3
Maximum distinct properties	1	5	860
	2	–	1135

Table 5 Total triples obtained for different combinations of *maxPathLength* and *maxPathProps*.

Total triples		Maximum path length 2	Maximum path length 3
Maximum distinct properties	1	1.7k	510k
	2	–	744k

The time required to run the crawler in each of these setups has been mostly dictated by DBPedia's `Crawl-Delay` directive, which imposes a 10 second delay between HTTP requests. Around 70% of triples in DBpedia connect resources using the property `dbo:wikiPageWikiLink`, which states that the Wikipedia page of the subject resource links to the Wikipedia page of the object resource. The property `dbo:wikiPageUsesTemplate`, which indicates that a Wikipedia pages has a infobox using a specific template, appears in 13% of the triples. More meaningful properties follow: `dbo:birthPlace` (4%), `dbo:deathPlace` (4%), `rdfs:type` (3%) and `owl:sameAs` (3%). By comparison, in Wikidata, the most frequent property is `http://schema.org/about` (40%) followed by `https://www.wikidata.org/wiki/Property:P31` (a Wikidata custom property similar to `rdf:type`, 3%).

7 Future work

We have planned several additional features for Derzis. Better error handling and other types of crawl limits are some examples. The indexes and queries performed to the MongoDB instance can also be improve. Given the small number of publicly available semantic web crawlers, and the distinct features presented by Derzis, we intend to make it easier to use by other people. Derzis is already open-source. However, usability and documentation needs to be improved. Real-time visualization tools to have an overview of the crawling process would also improve the overall experience.

Another focus of improvement is testing and evaluation. Derzis is currently lacking unit and integration tests. Additionally, real world experiments we have made with it so far are still only a proof of concept. We plan on evaluating Derzis using available benchmarks for SWCs and to perform our own comparisons against other SWCs.

Finally, Derzis was developed with the main motivation of being used in the broader context of implementing graph-based semantic measures. It addresses a need we have of being able to reduce semantic graph sizes keeping only the data relevant to a set of resources. We will continue to develop the larger system and evaluate Derzis in the context of this specific task.

8 Conclusions

The semantic web is a field of research with 20 years of existence. The early promise of a web understood by machines has not yet materialized. The large size of current knowledge graphs, the unreliable availability of queriable endpoints and the lack of universal vocabularies all contribute to this problem.

A semantic web crawler is capable of gathering semantic data using the linked data interface, a method of accessing the semantic web which is lighter for the clients and the servers. There have been several SWCs developed, but only a few are both publicly available and actively maintained. These tend to have in common a core set of features, such as limiting the depth of crawling, distributed architectures and complying with the Robots Exclusion Protocol. Other features such as requiring previous domain-knowledge, other types of crawl limits, customizable crawl priority or understanding RDFS and OWL basic constructs are present in specific crawlers.

Derzis is open source and available at `https://github.com/andrefs/derzis`. It presents as a distinctive feature the ability to reduce the crawled graph size. By keeping paths information during the crawling process, Derzis is able to limit the crawling to paths with a maximum number of distinct properties. This means that it can focus on more homogeneous paths, which provide more relevant information than heterogeneous ones. Additionally, Derzis does not dereference predicate resources, which usually provide information more relevant to the predicate than to the other elements of the triples.

References

1 Punam Bedi, Anjali Thukral, Hema Banati, Abhishek Behl, and Varun Mendiratta. A multi-threaded semantic focused crawler. *Journal of Computer Science and Technology*, 27(6):1233–1242, 2012.

2 Tim Berners-Lee. Linked Data – Design Issues, 2006. URL: `http://www.w3.org/DesignIssues/LinkedData.html`.

3 Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific american*, 284(5):34–43, 2001.

4 Dan Brickley, Ramanathan V Guha, and Andrew Layman. Resource Description Framework (RDF) Schemas, 1998. URL: `https://www.w3.org/TR/1998/WD-rdf-schema-19980409/`.

5 Carlos Castillo. Effective web crawling. *SIGIR Forum*, 39(1):55–56, 2005.

6 Giuseppe Cota, Fabrizio Riguzzi, Riccardo Zese, Evelina Lamma, et al. KRaider: a Crawler for Linked Data. In *34th Italian Conference on Computational Logic*, volume 2396, pages 202–216. CEUR-WS. org, 2019.

7 Fabien Gandon Dean Allemang, Jim Hendler. RDFS-Plus. In *Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL. Chapter 7*, 2020. doi:10.1145/3382097.3382107.

8 Marie Destandau, Caroline Appert, and Emmanuel Pietriga. S-Paths: Set-based visual exploration of linked data driven by semantic paths. *Semantic Web*, 12(1):99–116, 2020. doi:10.3233/SW-200383.

9 Michael Färber. The Microsoft Academic Knowledge Graph: A Linked Data Source with 8 Billion Triples of Scholarly Data. In *Proceedings of the 18th International Semantic Web Conference*, ISWC'19, pages 113–129, 2019. doi:10.1007/978-3-030-30796-7_8.

10 Nuno Freire and Mário J Silva. Domain-Focused Linked Data Crawling Driven by a Semantically Defined Frontier. In *International Conference on Asian Digital Libraries*, pages 340–348. Springer, 2020.

11 Olaf Hartig and Giuseppe Pirrò. A context-based semantics for SPARQL property paths over the web. In *European semantic web conference*, pages 71–87. Springer, 2015.

12 Pascal Hitzler. A review of the semantic web field. *Communications of the ACM*, 64(2):76–83, 2021.

13 Aidan Hogan. The Semantic Web: Two decades on. *Semantic Web*, 11(1):169–185, 2020. doi:10.3233/SW-190387.

14 Gary Illyes, Henner Zeller, Lizzi Harvey, and Martijn Koster. Robots Exclusion Protocol. URL: https://tools.ietf.org/html/draft-koster-rep-04#section-2.5.

15 Robert Isele, Jürgen Umbrich, Christian Bizer, and Andreas Harth. LDspider: An open-source crawling framework for the Web of Linked Data. In *Proceedings of the 2010 International Conference on Posters & Demonstrations Track*, volume 658, pages 29–32. Citeseer, 2010.

16 Arun Krishnan. Making search easier, 2018. URL: https://www.aboutamazon.com/news/innovation-at-amazon/making-search-easier.

17 Ora Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax specification, 1998.

18 Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195, 2015.

19 Marc Najork. Web crawler architecture, 2009.

20 A Gomes Raphael do Vale, Marco A Casanova, Giseli Rabello Lopes, and Luiz André P Paes Leme. CRAWLER-LD: a multilevel metadata focused crawler framework for linked data. In *International Conference on Enterprise Information Systems*, pages 302–319. Springer, 2014.

21 Michael Röder, Geraldo de Souza Jr, and Axel-Cyrille Ngonga Ngomo. Squirrel–Crawling RDF Knowledge Graphs on the Web. In *International Semantic Web Conference*, pages 34–47. Springer, 2020.

22 Amit Singhal. Introducing the knowledge graph: things, not strings. *Official google blog*, 5:16, 2012.

23 Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. Triple Pattern Fragments: a low-cost knowledge graph interface for the Web. *Journal of Web Semantics*, 37:184–206, 2016.

24 Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

25 Liyang Yu. Follow your nose: a basic semantic web agent. In *A Developer's Guide to the Semantic Web*, pages 533–557. Springer, 2011.