DR WEB: A MODERN, QUERY-BASED WEB DATA RETRIEVAL ENGINE

Ylli Prifti
Birkbeck, University of London
Email: y.prifti@bbk.ac.uk

Alessandro Provetti
Birkbeck, University of London
Email: a.provetti@bbk.ac.uk

Pasquale de Meo
DICAM, University of Messina
Email: pdemeo@unime.it

ABSTRACT

This article introduces the Data Retrieval Web Engine (also referred to as *doctor web*), a flexible and modular tool for extracting structured data from web pages using a simple query language. We discuss the engineering challenges addressed during its development, such as dynamic content handling and messy data extraction. Furthermore, we cover the steps for making the DR Web Engine public, highlighting its open source potential.

1 Introduction and Related Work

The World Wide Web reached 1 billion registered websites in 2014 and was fast approaching 2 bullion by December 2021[1]. As of the end of 2024 there are 5.5 billion users of the internet, most generating content each day[2]. Most estimations of the internet size are usually based on the number of indexed pages on the leading search engines. Counters are generally in the form of users, number of pages, number of websites, number of tweets, etc. In reality, it is a non-trivial quest to determine the memory size of the internet. The situation becomes more challenging if we consider the deep web, which is usually estimated to be much larger than the visible web.

Nevertheless, the indeterministic characteristic of the memory size of the internet, the number is bound to be large and ever-growing. The amount of data presents unprecedented opportunities for data mining and information extraction from the web. This has proven to be true given the number of scientific papers and research based on data from the web.

However, the web is unstructured. Previous tentatives to apply a machine-readable structure [1] to the web have failed to become large-scale standards. As such, in the modern days, data on the web are either made available by their owners in the form of temporal datasets or extracted using crawlers and scraper that leverage existing APIs[3] or public web pages.

Large mainstream online social networks and often well-established social media sites offer access to their data via APIs. Methods for leveraging API access for research purposes can usually be found in literature [2, 3, 4, 5].

Even though data mining via APIs is the easiest way to access structured data directly, it comes with challenges and issues. For example, Pfeffer et al. [6] showed how to tamper with twitter's sample API. Online social networks are massive, and APIs only allow for sampled or local[4] data access. The locality is determined by the point of view of a particular profile, group, tweet, or hashtag. Hence, even when APIs allow access to structured data, we often find in literature alternative approaches. For example, to build a holistic view of the data on Facebook [7] while Catanese et al. crawled 12.5M profiles on Facebook with a Breadth-First-Search crawler [8].

Web Scraping is widely used in the business world and for scientific purposes. Sirisuriya categorised the different techniques in the following groups [9]:

[1] ca. 1.92B as of December 2021 according to www.internetlivestats.com

[2] Source: www.itu.int

[3] Short for *'application programming interface.'*

[4] The reach of the starting node usually determines locality, for example, friends on Facebook, or a sampled search of tweets with a certain hashtag.

1. Traditional copy and paste;
2. Text grabbing and regular expression;
3. Hypertext Transfer Protocol (HTTP) Programming;
4. Hyper Text Markup Language (HTML) Parsing;
5. Document Object Model (DOM)Parsing;
6. Web Scraping Software;
7. Vertical aggregation platforms;
8. Semantic annotation recognising, and
9. Computer vision webpage analysers.

In a more recent state-of-the-art analysis on web scraping, Sarr et al. [10] apply a different categorisation based on the 'approach:'

1. Mimicry;
2. Weight Measurement;
3. Differential, and
4. Machine Learning.

The considerations above need also be seen from another dimension. The web tends to be divided into three categories based on its reachability. When we discuss the web, we commonly refer to the "Surface web" that tends to be reachable from traditional, mainstream web engines. However, an even bigger and more information qualitative[11] part of the web is the 'Deep Web.' The Deep Web is usually hidden behind passwords, not linked to or many links deep that are difficult to reach with the traditional approaches of web crawling. Dedicated methods are found in the literature that addresses Deep Web data extraction. Of particular interest for our research is the work of Gottlob et al. [12] on OXPath - an XPath extension for web crawling and scraping that is particularly successful in extracting data from the deep web.

OXPath uses a CLI and queries written in the extended XPath language (i.e. oxpath) to extract semi-structured data from the web. The tool was used as one of the two query engines in a large scale distributed system for data extraction discussed in Prifti's doctoral research [13] (Chapter 5 and 6).

1.1 The queryable web

The concept of *queryable* is often tightly coupled with structure. Due to the success of the 'Structured Query Language' [14], efforts to make the web queryable often took the form of SQL extensions [15]. However, that doesn't overcome the problem of "structure". The web is unstructured, with patterns emerging between sites of similar categories.

The Semantic Web [1] and the query languages build on The Semantic web standard, such as RDF [16], are viable solutions to queryability and structure on the web. On the other hand, the Semantic Web has not been adopted as a standard at speed it was initially expected[17]. Much of the web is not structured according to the Semantic Web standard.

Other approaches are derived from the markup nature of the web and the output as seen by the end users. Web users visually consume browser interpretation of HTML content incorporating other media, styling and JavaScript. Regular expressions, Document Object Model and HTML Parsing, are methods [9] that use the markup nature for searching, parsing and extracting content from web pages. In a comparison work [18] between Document Object Model (DOM), Regular Expressions (RegEx) and XPath[19], RegEx and XPath have similar performances in memory usage and speed of extraction and, unlike DOM, can be used as queries, be external, decoupled and instrumental to the web scraping implementation. Between the two, XPath is a query language working well with markup constructs whilst Regular Expressions can easily grow in complexity[20] because you have to manage the flexibility and different styles of writing HTML (with multiple spaces, double quotes, single quotes, no quotes, in one line, in multi-lines, with inner data, without inner data).

The superiority of XPath to Regular Expressions (and DOM, to the extent that DOM is an unsuitable choice) is further confirmed by the fact that XPath is supported by most modern developer tools for searching (e.g. Chrome dev tools, Firefox dev tools) and modern testing engines like Selenium and Playwright.

- **OXPath - XPath Queryability**

The efficiency of building an XPath-based query engine for extracting data from the web has already been shown with OXPath [21].

OXPath is open source, and has an advanced command line interface[5] that can be used as a web query engine.

Research on OXPath and its implementation has been carried out at Oxford University at the beginning of the last century. The OXPath implementation is also used commercially and quoting from the open-source GitHub repository[6]

> *Meltwater uses OXPath to extract millions of documents from 100'000s of sources daily.*

OXPath has fundamental characteristics that make it a preferred choice to other similar web scraping tools. These have been discussed largely in [12].

(a) The OXPath Language construct is a superset of XPath. XPath is an established query language for markup constructs such as XML or HTML;

(b) OXPath supports "Kleene Star" navigation and follows up actions with additional constructs for termination conditions;

(c) Extraction markers are embedded in the definition and transparent to the overall construct;

(d) Support for actions and user interaction simulation, and

(e) Full support of the XPath node navigation functions.

Whilst the OXPath implementation[7] has been used in automatic full-site extraction [22, 13], redundancy driven data extraction [23] and browserless web data extraction [24], it is starting to show its limitations (for example, it being tightly coupled to a specific browser version), it also provides further opportunities for improving on scalability, sustainability and performance.

Data Retrieval Web Engine - JSON Queryability During our research and building OXPath queries, we found ourselves in the following situations where OXPath wasn't the best choice.

(a) **Simpler request** - The navigability of the resulting query didn't require actions and user interaction or browser rendering, but rather simpler requests and link follows would fulfil the needs;

(b) **Pre-Actions and Batch Requests** - The need to perform pre-actions (such as logging in) and then retain user cookies for a batch of links provided in input.

(c) **Support for Modern Browser** - The latest OXPath CLI was built in 2017. It has embedded gecko drivers[8], Selenium and Firefox versions that are, at the time of writing, over five years old. More modern web technologies sometimes have unexpected or unsupported behaviour.

(d) **Closer to output format** - OXPath was built with XML in mind, and its primary output is XML format. We have seen in the last decade a shift in web technologies from mainly XML-based (SOAP, Web Services, XHTML) to JSON-based (REST APIs, JSON+LD, GraphQL). There might be a need for a query definition that is closer to the output. Additionally, JSON has been successfully used as API query definition, for example, for API queryability in GraphQL[25]

With this in mind, we set on a journey to build an open-source python package that supports JSON queries, uses the latest gecko driver, browsers and overall is build on top of more modern technologies. Our goal is to create an open source tool that can grow to make the whole web queryable.

2 The project

This is the third iteration of building the data retrieval web engine (aka *dr-web-engine* or *Doctor web*). The previous two versions were published as pre-build version in the python packages repository pypi.org[9].

[5] Please see sourceforge.net/projects/oxpath/files/oxpath-cli/1.0.1/

[6] Please see github.com/oxpath/oxpath.

[7] Please see again github.com/oxpath/oxpath.

[8] See github.com/mozilla/geckodriver.

[9] The last publication of the pre-release version published on the September 6 2020: pypi.org/project/dr-web-engine/0.3.2.2b0/.

The source code is published on the GitHub repository github.com/starlitlog/dr-web-engine and is open for community contributions. The package can be installed from the source code, as explained in the repository README file, or more conveniently, can be installed from the python package repository '*pip install dr-web-engine*'[10].

Alternatively, a docker image containing the latest version of the python package, is published in the public Docker Hub repository hub.docker.com/r/starlitlog/dr-web-engine the image can be pulled using docker pull '*docker pull starlitlog/dr-web-engine*' and used to run queries as shown in the example below.

The following listings show an example query (file name 4chan-query.json5), its execution using the installed package or the docker image, and an extract of the result.

Listing 1: A query example 4chan-query.json5

```
{
  "@url": "https://boards.4chan.org/pol/catalog",
  "@steps": [
    {
      "@xpath": "//div[contains(@class,␣'thread')]",  // Selecting each thread
      "@fields": {
        "title": ".//div[contains(@class,␣'teaser')]/text()",
// Extracting thread title
        "link": "./a/@href",  // Link to the thread
        "number_of_posts": ".//div[contains(@class,␣'meta')]/text()"
// Number of posts
      }
    }
  ]
}
```

Listing 2: Running the query using python package or docker

```
    # query execution in verbose mode with headless browser
    dr-web-engine -q 4chan-query.json5 -o 4chan-data.json -l info --xvfb

    # or alternatively by running the docker image
    docker run --rm -v ~/data:/app starlitlog/dr-web-engine -q 4chan-query.json5 \n
    -o 4chan-data.json -l info --xvfb
```

Listing 3: Extract from the result 4chan-data.json

```
[
...
  {
    "title": "Hinduphobia␣is␣about␣to␣become␣illegal␣in␣America...",
    "link": "//boards.4chan.org/pol/thread/497716745",
    "number_of_posts": "R:␣82␣/␣I:␣14\u25b6"
  },
  {
    "title": "/ptg/␣-␣PRESIDENT␣TRUMP␣GENERAL␣-␣FIXING␣THE␣FARMS␣EDITION...",
    "link": "//boards.4chan.org/pol/thread/497716315",
    "number_of_posts": "R:␣161␣/␣I:␣82\u25b6"
  },
  {
    "title": ">Freedom␣of␣speech␣caused␣the␣Holocaust␣This␣is␣actually...",
    "link": "//boards.4chan.org/pol/thread/497720014",
    "number_of_posts": "R:␣14␣/␣I:␣2\u25b6"
  },

...
]
```

[10]The head version on the pypi.org repository pypi.org/project/dr-web-engine/.

This reportintentionally avoids too much detail on the technical aspects of the package development. For a more in depth reading, we refer the reader to the GitHub repository and connected resources. With that in mind, there are some aspects of the package that are worth mentioning because of its unique and diversifying characteristics.

2.1 The Query Language for data extraction

The package support two formats for writing the queries:

1. **JSON5**: Backward compatible with JSON (i.e. JSON is always a valid JSON5), the newer version supports some additional nice features that make it a better fit as a query language. Among others, support from comments in code.
2. **YAML**: The less verbose syntax and improved readability makes YAML preferred in some contexts, and we support YAML from the outset queries from the outset.

JSON5 and YAML are a clear diversion from the OXPath syntax, which was XPath based. Additionally, we decided to diverge the semantics of the query language from OXPath with the intention to simplify writing queries and improve readability. While in some cases this might translate in more verbose queries, these are more readable and intuitive to write. In the OXPath equivalent query for the 4chan extraction in listing 4 it is not immediately clear the structure of the output. The equivalent YAML query in listing 5 supported by *doctor web* of the same example 1 is more readable. Additionally, if you remove the query syntax keywords from the JSON5 example in 1, you are left with exactly the structure of the output, creating a clear link between the query and the result.

Listing 4: OXPath query for scraping 4chan threads

```
doc("http://boards.4chan.org/pol/catalog")
  //div[contains(@class, 'thread')]:<links>[
     .//div[contains(@class, 'teaser')]:<title=normalize-space(.)>
     ./a:<link=qualify-url(@href)>
     .//div[contains(@class, 'meta')]:<number_of_posts=normalize-space(.)>
  ]
```

Listing 5: YAML equivalent for the 4chan query

```
@url: "https://boards.4chan.org/pol/catalog"
@steps:
  - @xpath: "//div[contains(@class,␣'thread')]"  # Selecting each thread
    @fields:
      title: ".//div[contains(@class,␣'teaser')]/text()"  # Extracting thread title
      link: "./a/@href"  # Link to the thread
      number_of_posts: ".//div[contains(@class,␣'meta')]/text()"
# Number of posts
```

While it is not the intention of this report to expand on extendability, we want to highlight the simplicity of the language model and the easy with which it can be expanded as shown by the keyword definitions and mapping shown in the mode file [11]

For completeness of the query language construct and semantics, listing 6 shows a more complex query that extracts child minder profiles from a search and follows the profile links to extract reviews and other profile information.

Listing 6: A more complex query example childcare-query.json5

```
{
  "@url": "https://www.childcare.co.uk/search/Babysitters/DA12+1AB",
  "@steps": [
    {
      "@xpath": "//div[contains(@class,␣'search-result')]",
// Selecting each search result
      "@fields": {
        "full_name": ".//div[contains(@class,␣'items-baseline')]/div[1]/span[1]/text()",
        "ratings": ".//div[contains(@class,␣'rating')]/span[1]/text()",
        //"distance": ".//span[contains(@class, 'distance')]/span[2]/normalize-space()",
```

[11] https://github.com/starlitlog/dr-web-engine/blob/main/engine/web_engine/models.py

```
        "image_url": ".//div[contains(@class,␣'profile-image')]//img[1]/@src"
      },
      "@follow": {  // Following the profile link
        "@xpath": ".//div[contains(@class,␣'profile-image')]//a[1]/@href",
        "@steps": [
          {
            "@xpath": "//div[contains(@class,␣'profile␣featured')]",
// Extracting profile details
            "@name": "profile",
            "@fields": {
              "bio": ".//h3[text()='About␣Me']/../p/normalize-space()",
              "experience": ".//h3[text()='My␣Experience']/../p/normalize-space()"
            }
          },
          {
            "@xpath": "//div[@id='reviews']//div[contains(@class,␣'review')]",
// Extracting reviews
            "@name": "reviews",
            "@fields": {
              "reviewer": ".//p[2]//a/text()",
              "reviewer_profile": ".//p[2]//a/@href",
              "rating": ".//div[contains(@class,␣'rating')]//img/@alt",
              "comment": ".//p[1]/normalize-space()"
            }
          }
        ]
      }
    }
  ]
}
```

2.2 Extraction engines

Data retrieval web engine uses Playwright `https://playwright.dev/python/` as its extraction engine. Playwright has its own embedded web drivers, removing the need to independently manage the drivers. In contrast, OXPath uses Selenium `https://www.selenium.dev/`, which in turn requires additional web-drivers (e.g. Chromium web drivers can be found here `https://sites.google.com/chromium.org/driver/`). OXPath uses a fixed version of Selenium, which also supports a fixed version for the web drivers. This becomes an important limitation since the web browsing technologies advance at a fast pace (e.g. Firefox had almost 50 releases in 2024 alone `https://www.mozilla.org/en-US/firefox/releases/`). It is a non-trivial task to update and support the latest browsers with OXPath while it comes out of the box with *doctor web*. We find the limitation of support for up-to-date browser version the biggest limitation of OXPath and one of the main drivers for writing *doctor web*.

While Playwright is the main engine for extraction, it is by no means intended to be the only one. In fact, other engines have been considered, including Selenium as a Playwright alternative and httpx [12] as a fast alternative when rendering the output in a browser is not needed. Doctor web was build with extendability in mind, and the extraction engine is abstracted 7 away to allow additional engines to be added by community contributions.

Listing 7: Extraction engine abstraction

```
from abc import ABC, abstractmethod
from typing import Dict, List, Any


class BrowserClient(ABC):
    """Abstract base class for browser clients."""

    @abstractmethod
    def navigate(self, url: str) -> None:
        """Navigate to a URL."""
        pass
```

[12] `https://pypi.org/project/httpx/`

```
@abstractmethod
def query_selector(self, selector: str) -> Any:
    """Query the DOM for an element matching the selector."""
    pass

@abstractmethod
def close(self) -> None:
    """Close the browser."""
    pass
```

Building doctor web has been a fascinating journey. Our approach was to create a robust solution to web scraping that could handle the complexity of modern web pages. We incorporated a branching strategy, extensive unit tests, and documentation to encourage open-source contributions.

3 Performance and Benchmarking

In this section, we are going to look at the performance comparison between doctor web and OXPath.

3.1 Experiment design

Our intention is to measure the mean of execution time, CPU usage and memory usage over 100 executions of three extraction queries, respectively designed to be of simple complexity, medium complexity and high complexity. In addition, the same experiment is repeated three times on three different machines intended to represent low performance machines, medium performance and high performance computing power and memory availability.

Table 1: Experimental Setup for Execution Time, CPU Usage, and Memory Usage Measurements

Parameter	Description	Values
Execution Queries	Type of extraction queries	Simple, Medium, High
Execution Count	Executions each query and each machine	10/query/machine
Total Execution Count	Total number of executions	90
Performance Categories	Performance categorization of machines	Low, Medium, High
Machines	Number of machines used for experiments	3
Metrics Measured	Average metrics recorded during the experiments	Mean Execution Time, CPU and Memory Usage
Environment Isolation	Measures to isolate the experimental environment	Docker Containers for OXPath and Doctor Web

While it is somehow subjective the qualitative attributes of simple, medium, high for query complexity and similarly the values of low, medium to high for computational power, we explain here some of the characteristics that drive the categorisation.

1. *Simple* A query that accesses a single page and return a limited number of properties (no more than 5), without any complex objects (single values or arrays) with the output file size no more than 0.5M
2. *Medium* A query that accesses a single page but can return multiple values (more than 5) and has complex objects (a property can be a nested object). The file size can be more than 0.5M
3. *High* A query that need to access multiple pages to build the output result utilizing Kleene star features, has multiple nested complex objects

Following a similar pattern for computational power:

1. *Low* Utilising a computer that has no more than 4 cores and no more than 8GB of virtual memory available. We utilised a MacBook Air 13-inch 2017 as low configuration. Properties: *CPU: 1.8GHz Intel i5 4 Cores, MEM: 8GB 1600MHz DDR3*
2. *Medium* Utilising a computer that has no more than 8 cores and no more than 96GB of virtual memory available. We utilised an iMac 2019 as medium configuration. Properties: *CPU: 3GHz Intel i5 6 Cores, MEM: 72GB 2667MHz DDR4*

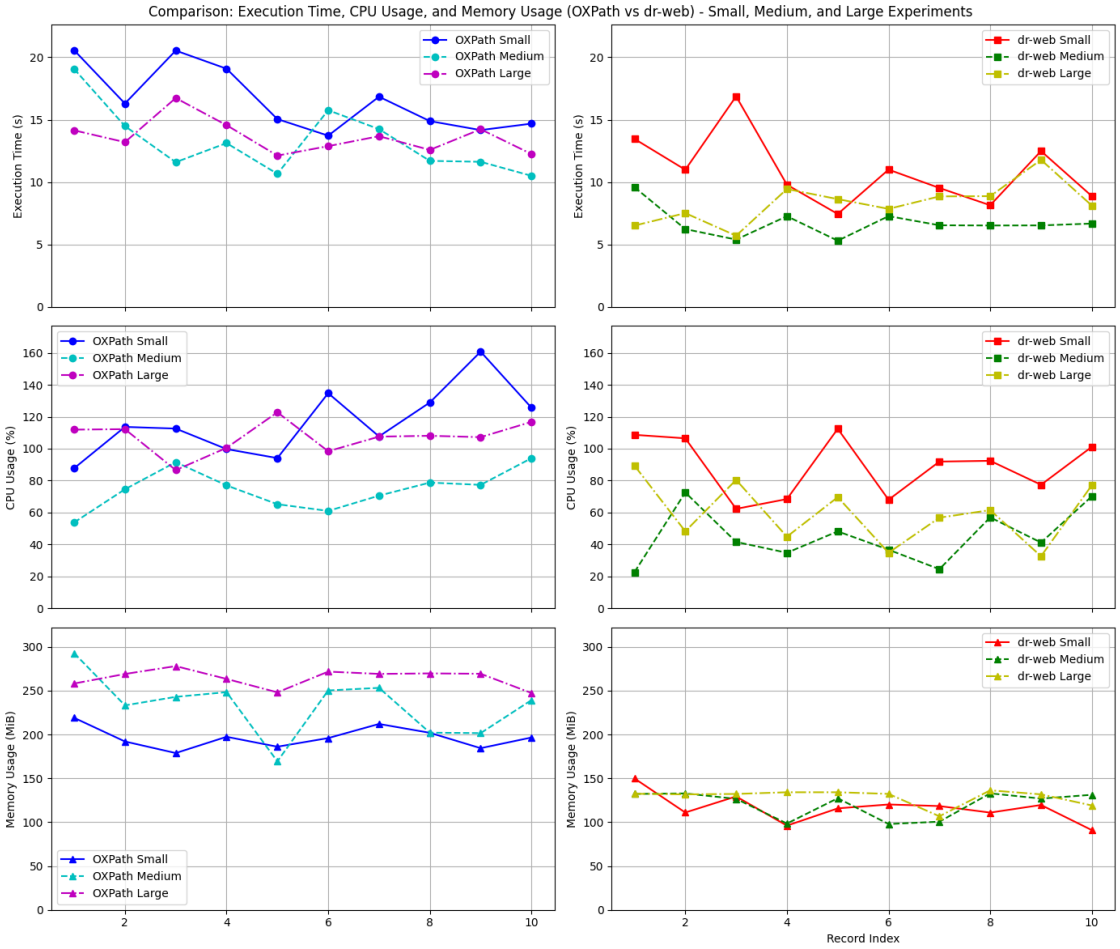

Figure 1: Comparing Doctor Web and OXPath

3. *High* Utilising a computer that highly outperforms the medium configuration. We utilised a Lenovo ThinkStation P920 as high configuration. Properties: *CPU: 2.3GHz Intel Xeon Gold 5118 48 Cores, MEM: 128GB 2133MHz DDR4*

Additionally, to partially reduce noise by the high diversity of the systems used for benchmarking, we utilized the same docker image with the same configuration to run the experiment. On all occasions, we configured docker to utilize max 2 CPU cores and 1GB of virtual memory.

3.2 Experiment results

All experiment setup, resulting data and analysis Jupyter notes can be found on GitHub [13]. We executed the experiment multiple times and consistently found the following:

1. Doctor Web is faster, reducing the execution time by half on average across all different computational specs and query complexity.
2. Doctor Web requires on average 40% less CPU usage across all different computational specs and query complexity.
3. Doctor Web uses about 60% less memory cross all different computational specs and query complexity.

In Figure 1 we show graphically show the results of running the same simple query with OXPath and Doctor Web on a small, medium and high-powered computational machine.

[13] `https://github.com/starlitlog/dr-web-engine/tree/main/benchmark`

There are some limits with the results due to the fact that we ran the experiment using docker images and the docker setup would affect the overall efficiently. While the results show a clear picture in terms of execution time, memory and CPU usage, there is a long way ahead before being conclusive due to the fact that there are clear feature gaps between OXPath and Doctor Web with the former being a feature mature engine with many years of usage.

As Doctor Web dynamically grows and the feature gab becomes smaller, the benchmarking experiments will become an important tool to measure if the improvement will resist time and increase in complexity.

4 Conclusion and Future Work

The development of the DR Web Engine is an ongoing process. We count on the tool growing dynamically by community contribution. We invite contributions and feedback to refine and enhance its capabilities.

While there are some clear feature gaps with existing established engines like OXPath, we show that Doctor Web has huge growth potential because of its modularity, build for community contribution, reduced complexity for query writing and support for different query formats. Additionally, we show that Doctor Web on average improves efficiency of memory and CPU usage by 40% and reduced the execution time by half.

In future work, we plan to tackle further challenges in web scraping and continuously improve the engine's efficiency and reliability.

Acknowledgments

We are grateful to Prof. Georg Gottlob FRS who introduced and motivated us to work on web data extraction. Thanks to all contributors and users who provided valuable feedback throughout the development process.

References

[1] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[2] Davide Marengo, Danny Azucar, Claudio Longobardi, and Michele Settanni. Mining Facebook data for Quality of Life assessment. *Behaviour & Information Technology*, 0(0):1–11, 2020.

[3] Stine Lomborg and Anja Bechmann. Using APIs for data collection on social media. *The Information Society*, 30(4):256–265, 2014.

[4] Anne Oeldorf-Hirsch and Darren Gergle. "Who Knows What": Audience Targeting for Question Asking on Facebook. *Proc. ACM Hum.-Comput. Interact.*, 4(GROUP), 1 2020.

[5] Mai Monica and Leung, Carson K. and Choi Justin M C. and Kwan Long Kei Ronnie. Big Data Analytics of Twitter Data and Its Application for Physician Assistants. In Alhajj Reda and Moshirpour, Mohammad and Far Behrouz, editor, *Data Management and Analysis: Case Studies in Education, Healthcare and Beyond*, pages 17–32. Springer International Publishing, Cham, 2020.

[6] Jürgen Pfeffer, Katja Mayer, and Fred Morstatter. Tampering with twitter's sample api. *EPJ Data Science*, 7(1):50, 2018.

[7] M Gjoka, M Kurant, C T Butts, and A Markopoulou. Walking in Facebook: A Case Study of Unbiased Sampling of OSNs. In *2010 Proceedings IEEE INFOCOM*, pages 1–9, 2010.

[8] Salvatore A Catanese, Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara, and Alessandro Provetti. Crawling Facebook for Social Network Analysis Purposes. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, WIMS '11, New York, NY, USA, 2011. Association for Computing Machinery.

[9] De S Sirisuriya et al. A comparative study on web scraping. *Empty*, 2015.

[10] Rabiyatou Diouf, Edouard Ngor Sarr, Ousmane Sall, Babiga Birregah, Mamadou Bousso, and Seny Ndiaye Mbaye. Web Scraping: State-of-the-Art and Areas of Application. In *2019 IEEE International Conference on Big Data (Big Data)*, 2019 IEEE International Conference on Big Data (Big Data), pages 6040–6042, Los Angeles, United States, 12 2019. IEEE.

[11] Michael K. Bergman. White paper: The deep web: Surfacing hidden value. *The Journal of Electronic Publishing*, 7(1), August 2001.

[12] Tim Furche, Georg Gottlob, Giovanni Grasso, Christian Schallhart, and Andrew Sellers. Oxpath: A language for scalable data extraction, automation, and crawling on the deep web. *The VLDB Journal*, 22(1):47–72, 2013.

[13] Ylli Prifti. *The emergence of interpersonal and social trust in online interactions*. PhD thesis, Birkbeck, University of London, 2023.

[14] Carolyn J Hursch, Jack L Hursch, and Carolyn J Hursch. *SQL, the Structured Query Language*. Tab Books, 1988.

[15] Ellen Spertus and Lynn Andrea Stein. Squeal: a structured query language for the web. *Computer Networks*, 33(1-6):95–103, 2000.

[16] Ora Lassila, Ralph R Swick, et al. Resource description framework (rdf) model and syntax specification. *""*, 1998.

[17] Brian McBride. Four steps towards the widespread adoption of a semantic web. In *International Semantic Web Conference*, pages 419–422. Springer, 2002.

[18] Rohmat Gunawan, Alam Rahmatulloh, Irfan Darmawan, and Firman Firdaus. Comparison of web scraping techniques : Regular expression, html dom and xpath. In *2018 International Conference on Industrial Enterprise and System Engineering (ICoIESE 2018)*. Atlantis Press, 2019/03.

[19] James Clark, Steve DeRose, et al. Xml path language (xpath), 1999.

[20] Andrzej Ehrenfeucht and Paul Zeiger. Complexity measures for regular expressions. In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, page 75–79, New York, NY, USA, 1974. Association for Computing Machinery.

[21] Andrew Jon Sellers, Tim Furche, Georg Gottlob, Giovanni Grasso, and Christian Schallhart. OXPath. In *Proceedings of the 20th international conference companion on World wide web - WWW '11*. ACM Press, 2011.

[22] Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, Christian Schallhart, and Cheng Wang. Diadem: thousands of websites to a single database. *Proceedings of the VLDB Endowment*, 7(14):1845–1856, 2014.

[23] Jinsong Guo, Valter Crescenzi, Tim Furche, Giovanni Grasso, and Georg Gottlob. Red: Redundancy-driven data extraction from result pages? In *The World Wide Web Conference*, pages 605–615, 2019.

[24] Ruslan R Fayzrakhmanov, Emanuel Sallinger, Ben Spencer, Tim Furche, and Georg Gottlob. Browserless web data extraction: challenges and opportunities. In *Proceedings of the 2018 World Wide Web Conference*, pages 1095–1104, 2018.

[25] Olaf Hartig and Jorge Pérez. Semantics and complexity of graphql. In *Proceedings of the 2018 World Wide Web Conference*, pages 1155–1164, 2018.