World Wide Web (2019) 22:1577–1610
https://doi.org/10.1007/s11280-018-0602-1

Deep Web crawling: a survey

Inma Hernández[1] · Carlos R. Rivero[2] · David Ruiz[1]

Received: 22 May 2017 / Revised: 15 May 2018 / Accepted: 25 May 2018/
Published online: 5 June 2018

Abstract Deep Web crawling refers to the problem of traversing the collection of pages in a deep Web site, which are dynamically generated in response to a particular query that is submitted using a search form. To achieve this, crawlers need to be endowed with some features that go beyond merely following links, such as the ability to automatically discover search forms that are entry points to the deep Web, fill in such forms, and follow certain paths to reach the deep Web pages with relevant information. Current surveys that analyse the state of the art in deep Web crawling do not provide a framework that allows comparing the most up-to-date proposals regarding all the different aspects involved in the deep Web crawling process. In this article, we propose a framework that analyses the main features of existing deep Web crawling-related techniques, including the most recent proposals, and provides an overall picture regarding deep Web crawling, including novel features that to the present day had not been analysed by previous surveys. Our main conclusion is that crawler evaluation is an immature research area due to the lack of a standard set of performance measures, or a benchmark or publicly available dataset to evaluate the crawlers. In addition, we conclude that the future work in this area should be focused on devising crawlers to deal with ever-evolving Web technologies and improving the crawling efficiency and scalability, in order to create effective crawlers that can operate in real-world contexts.

Keywords Deep Web · Web crawling · Form filling · Query selection · Survey

✉ Inma Hernández
inmahernandez@us.es

Carlos R. Rivero
crr@cs.rit.edu

David Ruiz
druiz@us.es

[1] Department of Languages and Computer Systems, University of Seville, Seville, Spain

[2] Department of Computer Science, Rochester Institute of Technology, Rochester, NY, USA

1 Introduction

Deep Web crawling refers to the problem of traversing the collection of pages in a deep Web site and interacting with them automatically. In the past, the term "deep Web" has been used by different authors to refer to a variety of Web pages that are not indexed by traditional search engines, i.e., pages that are behind Web forms, that are not in HTML format, that belong to private areas of a Web site (which usually implies that to access them a user/password combination needs to be provided by the user), or that are not backlinked (and therefore cannot be reached by following links from other pages) [16, 23, 25, 30, 48, 58, 59, 76, 103]. Some authors [76] even conflate the deep Web (also known as "hidden Web") and the dark Web, which refers to a collection of Web sites that hide their location and IP addresses using a particular encryption software, such as Tor [62]. Usually, dark Web sites are the scenario of illegal activities, such as drug and weapon selling, illegal pornography, unauthorised leaks of sensitive information, or money laundering, amongst others [18]. In this article, we focus on the traditional meaning of the term "deep Web", which was proposed in 2001 by Bergman [9] as the collection of pages that only exist when they are dynamically generated in response to a particular query that is submitted using a search form.

A Web crawler is a piece of software that automatically navigates the Web and retrieves Web pages [70, 75]. Initially, crawlers were designed to retrieve the so-called surface Web pages, (i.e., Web pages that can be accessed by following links, as opposed to deep Web pages) [16, 75, 76]. Later, deep Web applications that offered their data only through search forms progressively became the main source of data in the Web [103]. Therefore, an effort was made to develop crawlers that were able to retrieve pages behind such forms.

The size of the deep Web is known to be significantly large [31, 58], and it typically contains high quality and interesting data in a wide range of semantic domains. As a result, devising deep Web crawlers that automatically access such data is an interesting research challenge. Unfortunately, the actual size of the deep Web remains unknown, and it is uncertain whether it can be actually measured. The technical report by Bergman [9] is the most cited reference for an estimation of its size, which was at the time 4,000–5,000 times larger than the size of the surface Web. The deep Web was already growing exponentially in 2001, so it is reasonable to assume that the size of the deep Web has only grown since then. The most recent estimations point out that the deep Web size reached several trillion Web pages in 2012 [25].

Deep Web crawling has a number of applications. The most usual ones are: creating indexed repositories of deep Web pages for search engines [9, 25, 47] and periodically refreshing the repositories to keep them updated [19, 25, 71, 73], performing automated testing of Web sites [37], or virtual integration, that is, offering a unified view on the data offered by different Web sites [24, 87, 96].

Figure 1 summarises a typical deep Web crawling process. We use Booking.com as a motivating example to illustrate some of the vocabulary that will be used throughout the article. Booking.com is a deep Web site that offers information about accommodations in a particular location and allows making reservations for them.

First, the search forms that constitute the entry points to deep Web sites (cf. Figure 2) must be known, which in most cases involves discovering them in an automated way. Since deep Web pages are generated as a response to the submission of those forms, deep Web crawlers must be able to fill in their fields using appropriate values, i.e., values of the same type and domain that are expected by the Web application server that generates the responses [76]. Since search forms are designed for a human-computer interaction, the

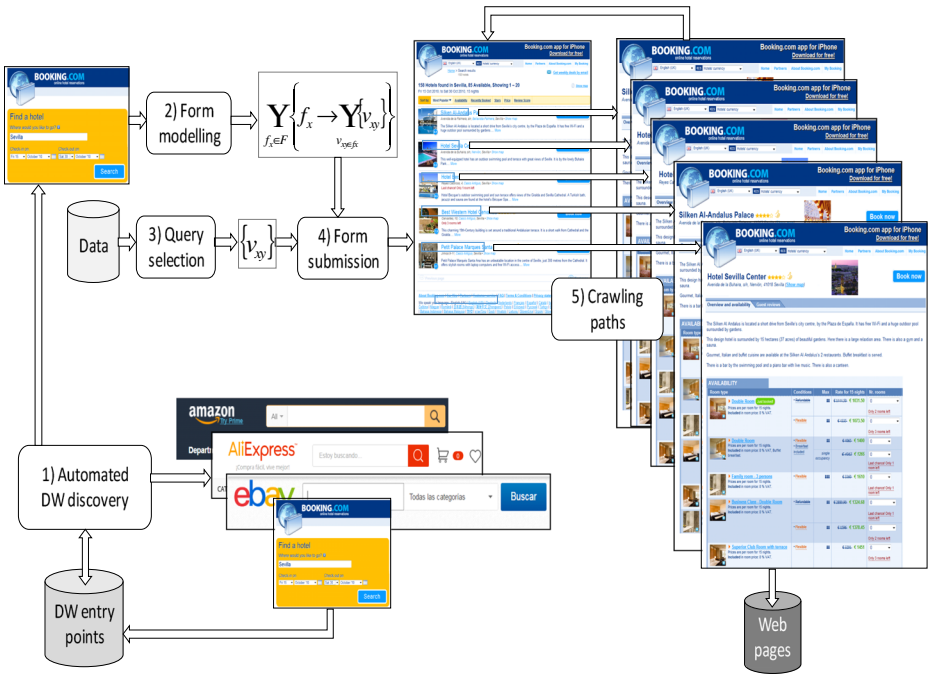

Figure 1 Deep Web crawling workflow

access to the deep Web requires simulating these interactions, hopefully without the intervention from the user. A human user that interacts with search forms needs to understand the search form to be able to discern the semantics associated to each field, the type and domain of the values that are expected in each field, and the relationship between fields

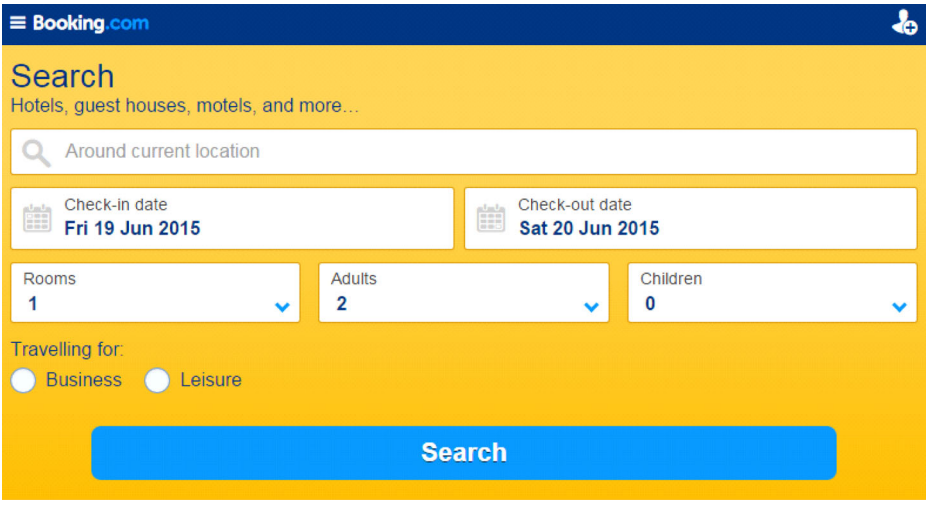

Figure 2 Sample form page in Booking.com

(form modelling), and provide a combination of values of the proper type and domain to fill in each field, so that submitting the form using those values yields as a response results that are relevant for the user (query selection) [86]. In some cases, a crawler needs to go further, navigating from those results pages into the deep Web site, by following a crawling path that allows it to reach the deep Web pages.

Usually, the pages reached by a deep Web crawler are processed by an information extractor, which extracts and gives structure to the information, allowing to integrate it in automated processes. Information extraction is a widely researched area, and many surveys can be found in the literature that provide a thorough, up-to-date vision of the state of the art [77, 82].

In summary, a typical deep Web crawling process includes the following steps:

1. Automated deep Web entry points discovery.
2. Form modelling.
3. Query selection.
4. Form submission.
5. Crawling paths learning.

To the best of our knowledge, there are four surveys on deep Web crawling [25, 61, 70, 76], but they do not provide a framework that allows comparing the proposals, and they do not analyse the measures and datasets that are used to evaluate the crawlers performance. There are two recent surveys on form filling techniques [47, 49], but they do not analyse the other steps in the deep Web crawling process. All these issues have motivated us to work on this survey, in which we propose a framework that analyses the main features of existing deep Web crawling related techniques, including the latest proposals in automated deep Web discovery, form filling, and crawling path learning, and we provide an overall picture regarding deep Web crawling. We have created a four dimensional comparison framework in which each dimension reports on several related objective features. The first dimension includes a collection of features that are related to the ability to automatically discover deep Web entry points; the second dimension includes a collection of features that are related to the ability to automatically fill in forms; the third dimension analyses features related to the process performed to automatically learn a crawling path. Finally, our fourth dimension focuses on the measures and datasets that are used to evaluate the crawler performance, which are aspects that have been neglected in the past. We include some features that have already been analysed in previous surveys, such as the ability to classify Web pages, classification features and algorithms [61, 76]. We introduce as well new features that, to the best of our knowledge, have not been analysed before, such as the ones related to the datasets used for experimentation, the features and algorithms to classify deep Web forms and locate deep Web entry points, the ability to learn crawling paths with or without user intervention, or the implementation languages and data structures behind these learning techniques.

Our main conclusion is that even though deep Web crawling is being a thoroughly researched field, there are still a number of open challenges, being one of the most compelling the measurement of a crawler's performance. This is mainly because of the lack of a standard measure to evaluate the effectiveness of crawlers, and because there does not exist a benchmark or publicly available dataset to use in the experiments, so the performance results provided by different authors are not comparable, and do not allow to draw significant conclusions regarding which crawler performs better. We provide a discussion of this and other open challenges that future proposals in this area should address, such as adapting crawlers to current trends in Web development technologies and the improvement of the efficiency and scalability of crawlers, to make them usable in a real-world context, amongst others.

The rest of this article is organised as follows: Section 2 reports on previous surveys of the literature; Sections 3 and 4 provide an overall view on the existing proposals for automated deep Web entry point discovery and form filling proposals; in Section 5 we survey the existing proposals to learn crawling paths; Section 6 analyses the measures and datasets that are used to evaluate the performance of deep Web crawlers; we describe the remaining open issues in deep Web crawling and anticipate the future work in this area in Section 7; finally, Section 8 concludes the article.

2 Related work

In this section, we focus on current surveys that analyse deep Web crawling proposals.

Olston and Najork [70] performed an exhaustive analysis of deep Web and surface Web crawlers, focusing on their goals, architecture, algorithms used to sort the queue of URLs to be visited, limitations, and problems. Their analysis covers 15 proposals from 1993 to 2010. They propose a framework to compare the proposals regarding whether they consider the following factors in their design:

- Coverage, which is the fraction of relevant pages that the crawler is able to retrieve.
- Freshness, which is the degree to which the crawled pages remain up-to-date regarding the online version of the page.
- Importance of the page or Web site with regard to other pages and Web sites.
- Relevance, which is the degree of interest that a user may have on the crawled pages.
- Dynamism, which refers to the content of a page or Web site change rate over time.

Their main conclusion is that crawling is a thoroughly studied area, but that there are some unresolved issues that should be addressed, namely:

- Optimal parameter tuning.
- Optimisation of the storage space by reducing the amount of metadata downloaded from each page and discarding low-quality and irrelevant pages.
- The integration of different sorting algorithms to improve crawling performance.
- The ability to fill in Web forms.
- How to implement vertical crawlers (i.e., crawlers that specialise in particular Web sites).
- Crawling of Web sites that use dynamic scripts to generate their content, such as JavaScript or AJAX.
- How to leverage personalised content offered by Web sites to tune the behaviour of a crawler.
- Utilisation of the information provided by some Web sites, like RSS feeds to further expand content discovery.

Their survey [70] is not entirely focused on deep Web crawling, which means that only some of the proposals that they analyse are deep Web crawlers, it does not cover the path learning ability of crawlers, and it does not analyse the crawlers regarding the measures and datasets that are used to evaluate them.

Dragut et al. [25] presented a survey on deep Web query interface understanding and integration. Their analysis is the most thorough to the present day, covering 26 proposals from 2001 to 2011. It covers the different steps of deep Web crawling, such as automated discovery of deep Web entry points, form modelling, query selection, as well as other related tasks, such as form integration and clustering, semantic typing, and information extraction.

The authors conclude that there are still some challenges remaining, such as the modelling and filling of forms that change in response to user interaction and forms that are implemented using Javascript and asynchronous technologies, the development of a proposal that covers every aspect of deep Web crawling in an integrated manner, and the evaluation of such proposals. Despite its thoroughness, this survey does not cover the path learning ability of crawlers, nor does it discuss the issue of how to measure the performance of crawlers. Finally, it does not provide a framework that allows comparing the proposals side by side with regards to their features.

Manvi et al. [61] proposed another survey on deep Web crawling proposals that covers 4 proposals from 2001 to 2010. They proposed a framework to compare the proposals that includes the following features:

- The ability to crawl structured and unstructured documents.
- Whether they provide a simple interface.
- Support for page classification and query probing.
- The use of ontologies to improve the crawling performance.
- The ability to dynamically revisit the pages to check if the crawled copies are up-to-date.

Their main conclusions are that crawlers rely on databases of words to perform form filling and, as a result, the crawling performance depends on the contents of these databases, and that there are some unresolved tasks in this area that should be addressed, namely:

- Indexing and ranking the crawled pages to allow a search engine use them appropriately.
- Analysing how the data extracted from the crawled pages can be used to answer user queries.
- Synchronising the different processes that must be executed to crawl a page (form filling and submitting, URL visiting, and indexing, amongst others).

Their survey [61] does not cover a wide range of proposals, neither does it organise the proposals into a taxonomy, or analyse the crawlers regarding the measures and datasets that are used to assess their performance.

Ru and Horowitz [76] proposed a survey on deep Web indexing, i.e., exposing deep Web site contents so that a search engine can index and return them in response to user queries. According to this survey, deep Web indexing includes methods for the automated discovery of deep Web forms, automated classification of deep Web sites, form modelling, form filling, information extraction from the resulting pages, and Web site content summarising, amongst others. Their analysis covers 10 proposals from 2000 to 2004. They distinguish between offline crawling techniques, which crawl deep Web sites and index their pages in a repository, from which they can be accessed in the future to respond to a user query; and on-the-fly crawling techniques, which access deep Web pages directly from their site to respond to a user query. They conclude that the main challenges for deep Web indexing techniques are:

- Generating combinations of values to fill in forms and yield as many relevant pages as possible.
- Developing a generalised method for automated form filling.
- Determining when a deep Web site has been thoroughly crawled and indexed.

Their survey [76] does not provide a framework that allows comparing the proposals side by side with regard to their features.

Regarding form filling surveys, Khare et al. [49] surveyed 10 proposals from 2001 to 2009. They classify the proposals according to two dimensions: the techniques used to understand the forms, which can be rule-based and model-based, and the complexity of the form model. They propose a framework to compare the proposals that includes the following features:

– Elements in the form model.
– Technique to parse the HTML page and extract the relevant elements.
– Technique to infer the model from the parsed HTML page.

Their main conclusions are that it is not clear whether future proposals should be domain-specific or domain-independent, but it is clear that to be scalable, they should be able to extract form models without supervision. They highlight that different techniques are usually devised with a particular application scenario in mind, which should be taken into consideration when evaluating its performance and potential benefits. This survey covers only form filling techniques and does not analyse the path learning ability of crawlers, or the techniques for deep Web entry points discovery, nor does it include the most updated proposals.

Finally, Kantorski et al. [47] presented the most recent survey on form filling. Their analysis covers 15 proposals from 2001 to 2013. They classify the proposals according to the technique on which they are based, and the type of forms that they are able to deal with, namely: heuristics applied to simple text forms; heuristics applied to advanced forms; machine learning applied to advanced forms; and overlapping combinations of forms types and/or techniques. They propose a framework to compare the proposals that includes the following features:

– Seed used to generate data to fill in forms.
– Algorithms used to generate data from the seeds.
– Whether the technique needs prior knowledge or not.
– Form submission method.
– User supervision.
– Number of forms.
– Performance measures.

Their main conclusions are that the future work in this area should be focused on automatically generating seeds and reducing human supervision to improve the scalability of the techniques, analysing different machine learning techniques to identify the most suitable one, and devising form filling proposals that are domain-specific. This survey covers only form filling proposals and does not analyse the path learning ability of crawlers, or the techniques for deep Web entry points discovery.

In Table 1, we summarise the coverage of the dimensions of our framework by the existing surveys. In this table, DWD stands for deep Web discovery, FF for form filling, CPL for crawling paths learning, and PM for performance measures.

3 Discovery of deep Web sites

3.1 Analysis of the proposals

Crawling the deep Web entails filling the search forms that constitute the entry point to deep Web sites and accessing the information that is behind those forms. Thus, a preliminary step is to discover where the entry points to these sites are, i.e., the pages in the Web that contain

Table 1 Coverage of the dimensions of our survey by previous proposals

Year	Survey	DWD	FF	CPL	PM
2010	Khare et al. [49]	×	✓	×	×
2010	Olston and Najork [70]	×	×	×	✓
2012	Dragut et al. [25]	✓	✓	×	×
2013	Manvi et al. [61]	×	✓	✓	×
2015	Kantorski et al. [47]	×	✓	×	×
2015	Ru and Horowitz [76]	✓	✓	✓	×

the forms that give access to the deep Web. Note that there exists a variety of forms scattered around the Web, some of which are intended for searching the deep Web, but also others whose goal is to allow the users perform other tasks, such as signing in to a site, signing up, or providing feedback to the Web site owner, amongst others. Obviously, only the first category of forms are entry points to the deep Web, which means that automated techniques have been devised to filter them out.

Some proposals are based on simple heuristics to automatically locate searchable forms. For example, Wu et al. [96] classify as potential deep Web entry points forms with at least one input textbox and the presence of keywords such as 'search', 'query', or 'find', either in the form tag or in the text immediately preceding or following it. Madhavan et al. [58] select forms that have at least one input textbox and between two and ten input elements of any kind. Bergholz and Chidlovskii [8] first discard forms with short input textbox (less than six characters) or that include a password field. Then, they fill in and submit the forms using a set of keywords and analyse the resulting pages, which are considered positive or negative whether or not they contain results. A form is considered a deep Web entry point if all negative pages have a similar length, at least 80% of the positive pages are at least three times larger than the average negative page length, and at least 80% of the positive pages have a significantly large edit distance with respect to negative pages.

Other proposals are based on training a classifier using machine learning techniques. Cope et al. [21] analyse a number of features that are extracted from the name and action attributes of each form, the names and values of its fields, and the types and number of fields, amongst others. Their proposal applies the C4.5 algorithm to learn a classification model to identify deep Web entry points using these features. Barbosa and Freire [5] propose FFC, a focused crawler that aims at locating deep Web entry points. FFC is based on four classifiers:

- A form classifier that decides whether a form is searchable or not.
- A topic classifier that predicts the class of information that each form gives access to.
- A page classifier that decides if a Web page contains information about a certain topic in a taxonomy.
- A link classifier that analyses links and decides if they eventually lead to searchable forms.

Their form classifier relies on features, such as the number of HTML input elements of each type, the form submission method, and the presence of keywords like 'search', amongst others. Their crawler relies on a multilevel URL queue, such that links that are found in

pages that are nearer to the Web site home page are assigned a higher priority, and therefore are visited earlier. The crawler stops when a number of searchable forms have been found, or a maximum number of pages have been visited. The different classifiers are built on well-known machine learning techniques, namely Naïve-Bayes and C4.5 decision trees. Since one of the main limitations of FFC is that selecting the features to train the link classifier is a time-consuming manual process, the authors propose ACHE [6], an evolution of FFC that is based on learning agents that deal with the problem of selecting a collection of good features for link classification. Zhao et al. [105] propose SmartCrawler, which follows the same approach to locate deep Web forms for performing deep focused crawling. SmartCrawler is based on a two-stage architecture: a first stage in which the crawler locates deep Web forms, and a second stage in which the crawler navigates the deep Web sites behind those forms.

Hicks et al. [36] propose a supervised approach to the problem of discovering and documenting deep Web sites of interest, i.e., sites that belong to a particular information domain. Their proposal is based on the Service Class Description concept (SCD), which aims to define a catalog of deep Web sites and how to interact with them. Their technique relies on a number of configuration files that must be provided by the user, namely:

- The SCD file: a handmade XML file with a description of a given deep Web site, including example queries, the expected data types that can be found in its pages and the control flow graph (interaction patterns).
- A crawler configuration file: it includes an ordered list of domain search terms, which are used to issue queries to the Bing search engine, a set of positive keywords that represent form labels that are most frequently found on deep Web entry pages of the domain, and the minimal set of form fields that must be present in the entry point form.
- A query configuration file: it includes the result data types from the SCD that must match the result page after a successful query submission, the number of result data types required to be a valid result page, a list of positive keywords, a list of negative keywords that are known not to be representative of the domain, possible names of the submit button, and the example query to be used.

The authors highlight that their proposal relies heavily on these files provided by the user, which makes the crawler error-prone and can be rendered useless if significant changes are made to the sites.

Finally, Li et al. [54] propose another crawling technique to automatically and efficiently locate the entry points to interesting deep Web sites. Their crawler is based on four different classifiers:

- A term-based page classifier, which determines if a page contains information about a certain topic.
- A link-based link classifier, which determines which links on a page eventually lead to a search form page.
- A search form classifier, which filters out non-searchable forms such as login forms.
- A domain specific form classifier, which selects only search forms from Web sites that offer relevant information about a given topic.

The crawler keeps a number of multi-level queues with promising URLs that are likely to lead to relevant searchable form pages in each Web site. To initialise these queues, the crawler uses as seeds the links to the first two results after performing a search on the Google search engine using descriptive words for each topic of interest.

3.2 Comparison framework

We summarise the existing proposals in Table 2, in which a dash ('-') in a cell means that a proposal does not have a value for the corresponding feature. We have included some features that are related to the ability to discern search forms that are entry points to deep Web sites from other types of HTML forms, namely:

- **PT**: Type of proposal, either based on heuristics or on machine learning classification techniques.
- **AP**: Approach that is used to discern deep Web entry points from other types of forms, either pre-query (in which the form is classified prior to its submission), or post-query (in which the form is submitted and the response pages are a source of features for the form classification).
- **CF**: Features of each form that are analysed to determine if the form is a deep Web entry point.
- **FO**: Whether the proposal is focused towards deep Web sites of a particular domain or not.
- **AL**: Classification algorithm used.
- **SU**: Whether the technique is supervised or not.

Although the earliest proposals were based on simple heuristics that allowed to perform this task without supervision, the current trend is to discover deep Web entry points based on a supervised classifier. Naive-Bayes and decision trees are the most common choices.

Almost all of the existing proposals follow a pre-query approach, which is more efficient since it avoids having to submit forms to classify them. The quest for crawling efficiency is one of the main conclusions of this survey, along with the scalability. In this case, heuristic-based proposals rely on a human expert that devises such heuristics, which hinders their scalability. Finally, the most recent proposals are focused on finding deep Web sites of a particular domain [5, 6, 36, 54, 105], instead of performing a generic search of deep Web sites of any domain.

4 Form filling

Since search forms are designed for human-computer interactions [27], the access to the deep Web requires simulating such interactions with an automated process. Any agent that interacts with search forms needs to fulfil two tasks [86]: form modelling and query selection. Form modelling involves understanding the search form to be able to discern the semantics associated to each field, the type and domain of the values that are expected in each field, and the relationship between fields. Query selection involves producing a combination of values of the proper types and domains to fill in each field. The goal of these two tasks is to submit the form using those values and retrieving as a response as many relevant results as possible. Both tasks can be automated using unsupervised or semi-supervised techniques.

In the following subsections, we provide some insight into the state of the art in both areas. Finally, we present a comparison framework to compare the existing proposals.

4.1 Form modelling

One of the main steps of automated form filling is to transform a search form designed for humans into a computational model that can be automatically processed [1, 17, 24,

Table 2 Automated discovery capabilities: PT (Proposal type), AP (Approach), CF (Classification features), FO (Focused), AL (Classification algorithm), SU (Supervision)

Year	Proposal	PT	AP	CF	FO	AL	SU
2003	Wu et al. [96]	Heuristics	Pre	Presence of input textbox, certain keywords	No	–	No
2003	Bergholz and Chidlovskii [8]	Heuristics	Pre / Post	Presence of input textbox with less than 6 characters, password fields	No	–	Semi
2003	Cope et al. [21]	Classifier	Pre	Form name, form action, field names field values, field types, number of fields	No	C4.5	Yes
2005	Barbosa and Freire [5]	Classifier	Pre	Number of fields of each type, submission method, presence of keywords	Yes	C4.5	Yes
2007	Barbosa and Freire [6]	Classifier	Pre	Automatically selected	Yes	C4.5	Yes
2007	Madhavan et al. [58]	Heuristics	Pre	Presence of input textbox, between 2 and 10 fields	No	–	No
2012	Hicks et al. [36]	Classifier	Post	Handmade	No	Ad-hoc	Yes
2013	Li et al. [54]	Classifier	Pre	Automatically selected	Yes	Naive-Bayes	Yes
2016	Zhao et al. [105]	Classifier	Pre	Terms	Yes	Naive-Bayes	Yes

32, 52, 53, 63, 64, 67, 75, 81, 86, 103]. Form modelling is not a trivial task, since every search form is created specifically for a particular Web site and there do not exist standards, guidelines or recommendations to guide form designers [47, 86]. However, there are some common patterns that designers follow implicitly when they design forms, so that they are user-friendly and easy to use, for instance: display the elements in order from top to bottom and from left to right (in western Web sites), visually group fields that are semantically related, place field labels visually near to their associated fields, use graphical elements to separate semantically different fields [98], or use a single label to describe each field, amongst others [16, 17, 24, 25, 86].

Forms can be either simple, i.e., composed of a single text field that allows to perform a keyword-based search, or advanced, i.e., composed of several fields of different types that allow to introduce combinations of values for creating refined search criteria.

Several proposals have been made to deal with the problem of understanding forms: Liddle et al. [56] do not create a form model. Instead, they submit search forms leaving each of their fields to their default values. He et al. [33] follow a similar approach in which only one text field from each form is considered in the model, and the rest of the fields are left to their default values.

Other proposals do create a model to represent each form in a machine-processable way. The most basic model is the flat compilation of form fields and the labels that semantically describe them [98]. Labels can be automatically extracted, with or without user supervision. Supervised label extraction techniques train on a number of annotated forms to infer correct associations between form fields and labels [51, 67, 86]. Unsupervised label extraction can be fulfilled by analysing the HTML code to identify each label in the text [29, 32, 53], by inspecting the DOM tree [29, 75], or by using visual techniques that analyse the proximity and relative position of fields and labels when the page is rendered in a browser [1, 24, 29, 45, 98, 103]. Note that the usual western convention is to place a label above its associated field or at its left side.

The most complex models consider not only the relationship between form fields and their semantic labels, but also some advanced features from each form, such as the existence of groups of semantically or logically related fields [24, 29, 31, 45, 81, 86, 98], mandatory fields [13, 22, 29, 59, 81], the domain of each field [13, 29, 44, 75, 80, 98], the measurement unit of a field [24], the relative order between fields [64], the distinction between fields that represent attributes from a given domain and fields that define how to display the search results [81], or an estimator of the probability of the form offering information about a particular domain [1], amongst others. The extraction of these models is usually performed using similar techniques as in the label extraction problem, with the help of other HTML features, such as separator lines, font sizes, or CSS styles [98].

Diadem [29] is the latest approach in this context. It creates the most complete model, which is a hierarchical representation of the fields in the form, considering the relationship between them and their associated constraints. Their proposal is based on the OPAL form filling technique, which is focused towards deep Web crawling and surfacing systems [27]. OPAL analyses visual, structural and textual features from each form field, such as the existence of a HTML label with a "for" attribute, if the field is a single child in the DOM tree, the visual similarity between fields, their "class" attribute, or the relative position between fields. This analysis provides a preliminary domain-independent form model, which can be later linked to an ontology to create a domain-specific form model.

Example 4.1 As an example, filling the Booking.com form (Figure 2) without a model entails leaving the Location field empty, radio buttons Business and Leisure unmarked, and the other fields to their default values.

A flat model would include the relation between each form field and its associated label. In most cases, Booking.com follows the aforementioned labelling convention; as an example, the field that allows selecting the number of rooms has an associated label "Rooms" just above it. However, note that this convention is not always followed. e.g., the radio fields that allow selecting the purpose of the travel have their associated label at their right side, instead of their left side.

Finally, a complex model should consider the dependencies between fields. For example, when filling the Check-in date field in with a particular day, the Check-out date field is automatically set to the following day, since both fields are semantically related and the check-out date must be always later than the check-in date. Mandatory fields should be considered as well. As an example, field Destination is mandatory, which means that submitting the form without filling the destination makes the Web site return an error page with the same search form and a descriptive message for the user.

4.2 Query selection

When a form has been modelled, it can be automatically filled in by instantiating the model and selecting a valid combination of values for its fields [75]. Each combination of values involves a new form submission, which is received by the Web site server. The server uses the form values to compose a query that is executed on its database, and yields a response page with the results of that query. The number of results for each query depends on the availability of information that is related to that query in the Web server, which means that in order to retrieve all the relevant information, usually more than one query is needed. However, every submission increases the amount of resources consumed by the crawler, which hinders its scalability. Furthermore, some of the queries are useless, either because they do not retrieve any response data, or because they retrieve exactly the same data as other previous queries [46, 80]. In addition, increasing the amount of submissions puts a load on the Web site server, which may have a negative effect on its performance [59]. Therefore, the goal is to select a number of queries such that a maximum coverage of the relevant information is achieved with a minimum number of form submissions [47]. Sheng et al. [80] propose a model to estimate the minimum number of form submissions that were necessary to retrieve the entire content of a particular deep Web site.

Query selection for a simple form that includes a single text field involves choosing a number of keywords to fill it in. There are a number of unsupervised or semi-supervised techniques that retrieve those keywords from the Web site itself, which avoids having to supply a repository of sample data. In that case, an information extraction technique is used to automatically retrieve the keywords from the Web site without the intervention of the user [29, 42, 55, 59, 83].

Barbosa and Freire [4] proposed one of the first unsupervised proposals for keyword selection that starts with a number of initial keywords extracted from the form page. After each form submission, they retrieve the most frequent keywords in the response page, repeating this process until a maximum number of submissions has been reached. Ntoulas et al. [69] propose a semi-supervised statistical-based technique that starts submitting the

form with a user-provided keyword, and extracts further keywords from the response pages, selecting those keywords with higher informativeness, which is calculated as their accumulated frequency. A similar proposal is made by Wu et al. [97] using a different criteria to calculate the informativeness of each keyword. Wang et al. [91] initially extract a pool of keywords from a random sample of pages from the site. Each page in the sample is assigned the inverse of the number of keywords in the pool that occurs in the page, and each keyword is assigned the sum of the page weights of all pages containing that keyword in the sample. Keywords whose frequency in the initial sample divided by their weight is as small as possible are chosen as queries. Finally, He et al. [33] propose an unsupervised technique in which the keywords are not extracted from the Web site itself, but from Google's query logs and Freebase [12], a knowledge base that stores tuples about general human knowledge, such as people and places.

Query selection for advanced forms poses additional research challenges, specially when dealing with generic text boxes. A number of supervised techniques have been proposed that rely on a repository of sample data to fill in the fields, usually attribute-value pairs [1, 53, 75]. A correspondence can be established between the form model and the data in the repository, by means of matching each field in the model with an attribute in the repository, and assigning the field one of the values associated to the attribute. These correspondences can be automatically created by using machine learning techniques [41, 88, 106], or visual and text-based heuristics [1, 53]. There are some unsupervised proposals to deal with this problem, namely: Madhavan et al. [59] follow the same approach for query selection as Barbosa and Freire [4], using the TF-IDF measure instead of the absolute frequency of each candidate keyword, and Kantorski et al. [46] apply the informativeness model that has already been applied to simple forms, assigning an estimator of informativeness to each combination of values.

A different approach is followed by Wu and Zhong [99] and Wu et al. [100], who propose Deep2Q, a technique to translate search forms from a number of deep Web sites into phrase queries, i.e., queries that are expressed as sentences in simplified natural language. This translation is made offline, based on rules that are inferred from the observation of the most frequent query patterns and values in search engine query logs. To improve the crawling efficiency, Deep2Q executes offline queries using those values against the search forms in the repository, and the result pages are cached. During crawling time (online), Deep2Q allows the user to introduce a number of keywords; the most similar phrase query in the repository is selected and used to return a number of result pages, either from the cache (if the query is one of the most frequent ones), or by filling the form using the corresponding rule.

Finally, note that selecting values for radio buttons, drop-down lists, or checkboxes does not generally pose a challenge, since the possible values can be retrieved from HTML code of the search form page. However, some techniques are focused on forms that include these elements, specially in order to improve the crawling efficiency [102].

Example 4.2 As an example, query selection for Booking.com search form may rely on a repository of toponyms to fill in the location text field and valid date values for the check-in and check-out text fields, with the restriction that the combinations of check-in and check-out values should be valid. The other fields are radio buttons and drop-down lists, which means that it is not necessary to provide sample values for them.

4.3 Comparison framework

We summarise the form filling capabilities of the existing crawling proposals in Table 3, in which a dash symbol ('-') in a cell means that a proposal does not provide information about the corresponding feature. We have included some features that are related to the form model of each proposal, and the data used to fill in the forms, namely:

- **FT**: Type of form that each proposal is able to deal with; we distinguish between simple forms with a single textbox, and advanced forms with more than one field, possibly of different types.
- **MO**: Type of form model, either flat models that represent each form as a list of field-value pairs, or complex models that represent each form as a collection of hierarchically structured objects subject to constraints.
- **SU**: Whether the extraction of the form model requires supervision.
- **MI**: Information that each model includes, such as which form fields are mandatory, the domain of each field, or the semantic relationship between fields.
- **RE**: Repository of data from which a technique extracts values that can be used to fill in the form fields; we distinguish between user-provided values (user), values automatically extracted from the Web site that is been crawled (site), values that are automatically extracted from other sources (off-site), or all of the former (all).

The main conclusions that we draw from this analysis are the following: regarding the form model, most proposals use a flat model that consists of a list of fields and their associated labels; these models discard semantic information that would be very useful to process these forms and the result pages that they lead to. We are living in the era of the Web of data, in which the main focus is on the data, represented by means of entities, attributes and the relationship between those, and the deep Web is not an exception. Therefore, the latest proposals are tending towards providing a complex representation of forms that allows to model them as a collection of entities, attributes and relationships between them, and their associated constraints.

Regarding the repository of data, we observe that most techniques are supervised, which means that the user has to provide the values to create the repository; the proposal by Lage et al. [53] offers the possibility of extracting the repository values from an external data source; Google's crawler [59] requires only a reduced set of seed words to start crawling, and it automatically extracts new words from the crawled pages; Raghavan and Garcia-Molina [75] combine the three approaches: it requires an initial set of seed words provided by the user, it is also able to query Web repositories through an API to retrieve more words, and it automatically extracts further words from crawled pages. Relying on the user to provide the repository of data hinders the scalability of these techniques.

We believe that future proposals should be focused on reducing user intervention to improve scalability, and on relying on automated techniques to extract the data. Furthermore, extracting data from the site that is being crawled means that this data is domain-specific, i.e., belonging to a particular information domain, and submitting the forms using this data would yield relevant results faster, improving the crawling efficiency. This means that such a proposal would be able to adapt to specific domains of interest without having to resort to domain-specific repositories or sample data provided by the user.

Table 3 Form filling capabilities: FT (Form types), MO (Form model), SU (Model extraction supervision), MI (Model information), RE (Repository of data)

Year	Proposal	FT	MO	SU	MI	RE
1998	Chakrabarti et al. [15]	Simple	–	–	–	User
1999	Davulcu et al. [22]	Advanced	Flat	Yes	Mandatory fields, optional fields, field labels	User
2000	Anupam et al. [2]	Advanced	Flat	Yes	Field labels	User
2001	Modica et al. [64]	Advanced	Complex	No	Fields order	User
2001	Raghavan and Garcia-Molina [75]	Advanced	Flat	No	Field labels, field domains	All
2002	Liddle et al. [56]	Advanced	–	–	–	–
2002	Pan et al. [72]	Advanced	Flat	Yes	Field labels	User
2003	Kushmerick. [51]	Advanced	Flat	Yes	Field labels, field domains	–
2004	Caverlee et al. [14]	Simple	Flat	–	Field labels	User
2004	Lage et al. [53]	Advanced	Flat	No	Field labels	User/off.
2004	Zhang et al. [103]	Advanced	Flat	No	Field labels	–
2005	Baumgartner et al. [7]	Advanced	Flat	Yes	Field labels	User
2006	Holmes and Kellogg [37]	Advanced	Flat	Yes	Field labels	User
2007	Álvarez et al. [1]	Advanced	Flat	No	Field labels, relevance, domain probability	User
2007	He et al. [32]	Advanced	Complex	No	Semantically related fields, logical relationships between fields	–
2007	Shu et al. [81]	Advanced	Complex	No	Semantically related fields, mandatory fields, attribute fields vs. display fields	User
2008	Blythe et al. [11]	Advanced	Flat	Yes	Field labels	User
2008	Madhavan et al. [59]	Advanced	Complex	No	Field types, field labels, mandatory fields	Site
2008	Nguyen et al. [67]	Advanced	Flat	Yes	Field labels	–
2008	Vidal et al. [89]	Advanced	Complex	No	Field labels, field types	User
2009	Dragut et al. [24]	Advanced	Complex	No	Semantically related fields, field domains, measurement units	–
2009	Montoto et al. [65]	Advanced	Flat	Yes	Field labels	User
2011	Jin et al. [44]	Advanced	Complex	No	Field domains	–

Table 3 (continued)

Year	Proposal	FT	MO	SU	MI	RE
2011	Sheng et al. [80]	Advanced	Complex	No	Field domains	–
2012	Liakos and Ntoulas [55]	Simple	Flat	–	Field labels	User/site
2012	Zhao and Wang [104]	Advanced	Flat	Yes	Field labels	User
2013	Furche et al. [28]	Advanced	Flat	Yes	Field labels	User
2013	He et al. [33]	Simple	Flat	No	–	Off-site
2013	Su et al. [86]	Advanced	Complex	Yes	Semantically related fields, logical relationships between fields	–
2014	Losada et al. [57]	Advanced	Flat	Yes	Field labels	User
2014	Xu et al. [101]	Simple	Flat	–	–	User
2014	Furche et al. [29]	Advanced	Complex	Yes	Field labels, mandatory fields, field domains, semantically related fields	Site
2015	Jamil and Jagadish [40]	Advanced	Complex	Yes	Logical relationships between fields	User

5 Crawling paths learning

Deep Web crawlers are not confined to filling forms and returning the result pages. In some cases, they need to go further into the deep Web, by navigating sites and finding the subset of pages in those sites that are relevant for the user or process that is making use of the crawler.

Deep Web crawling techniques build on crawling paths, i.e., sequences of pages that must be visited to reach relevant pages, as well as the interactions needed in each page (e.g., the forms to be filled in, the values to fill in each form, the user events to simulate, or the links to be followed). Note that some techniques include the form filling and submissions as part of the crawling paths, whereas for others, the crawling paths start from the result pages that the forms return after a submission. We would follow the first definition, since it is more general. In some cases, every page in the site that is reachable from the Web site home page by following links and/or filling in forms is relevant, which means that all of them should be downloaded by the crawler. In other cases, only a subset of the reachable pages are relevant, which means that an efficient crawler should apply a classification technique to select the subset of relevant pages that should be downloaded. According to this distinction, in the literature we can find blind crawlers, focused crawlers, and ad-hoc crawlers.

Blind crawlers [14, 59, 75] aim to download as many pages as possible from a Web site; they start with a seed page and follow every link that it provides iteratively until every page that is reachable from the seed has been downloaded. Therefore, the crawling paths that they follow are composed of every reachable URL in the Web site. Focused crawlers [15, 29, 55, 101] aim to follow only those links that are likely to lead to pages that contain information about a certain topic; they usually classify the pages that are downloaded to check if they belong to the topic, and, in that case, they follow their links. Finally, ad-hoc crawlers [1, 2, 7, 11, 22, 28, 33, 37, 53, 57, 65, 72, 89, 104] aim to follow only those links that are likely to lead to pages that contain information that is relevant for a particular user, not necessarily belonging to the same topic.

Blind crawlers are the most straightforward approach, and they usually do not apply a complex algorithm to generate the crawling paths; in most cases, their paths consist on queues in which they arrange the URLs to be visited in a particular order, usually for efficiency reasons. Focused and ad-hoc crawlers, instead, need to discriminate which pages are included in their crawling paths, and therefore the procedures that need to be performed in order to reach them. Depending on the technique used to generate the crawling paths and the level of supervision that they require, we distinguish between recorders, supervised learners, and unsupervised learners.

5.1 Recorders

Recorders allow the users to define crawling paths by hand, usually providing them with some kind of visual support to ease this task [1, 2, 7, 22, 37, 57, 65, 72]. The simplest proposals offer the user a recorder-like interface, and they require her to perform the crawling steps, such as selecting a search form, providing the keywords to fill it in, selecting the links that must be followed from the resulting hubs, and so on. These steps compose the paths that are recorded and later replayed.

Davulcu et al. [22] propose a technique to extract information from the Web, which includes a recorder. In this technique, paths are expressed using two declarative languages: F-logic to model objects, such as Web pages, forms and links, and transactional logic to model sequences of actions, such as submitting forms or following links. Their navigator

includes a Prolog-based module that learns the paths from user examples, i.e., as the user navigates the site, the system models the objects in the pages that she visits, and the actions that she performs. In some cases, the user has to provide some additional information, e.g., which fields are mandatory in each form. Then, the paths are executed by a transaction F-logic interpreter module. This system is able to automatically update the paths when minor changes are made to Web sites, such as the addition of new options in a select field of a search form.

Anupam et al. [2] propose WebVCR, a recorder with a VCR-like interface to record and replay a user's navigation steps through a Web site. Paths are XML-like documents in which the steps can be either following a link or submitting a form with user-provided values. For each step, the recorder stores the values of some DOM attributes (name, href, target, and locator, if the step involves following a link, or name, method, and action, if the step involves submitting a form), as well as the list of form fields and values that must be used to fill in the form. The recorder can be implemented as a client Java applet to be loaded in a Web browser, or as a Web server that acts as a proxy for the user requests. In the server version, the replay of a path is invisible to the user, who only receives the final page. The authors propose some heuristics to update the path automatically after minor changes in Web pages occur, and to optimise the replay of paths by not loading the images in the intermediate pages.

Pan et al. [72] introduce the WARGO system to generate Web wrappers. WARGO uses a browser-based recorder that captures the user actions and creates a path that is written in a declarative language called Navigation Sequence Language (NSEQL). NSEQL is based on the WebBrowser Control, which is a module in Microsoft Internet Explorer that supplies the functionalities associated to navigation. Therefore, NSEQL is composed of commands that allow, for example, loading a Web page, finding a form in a Web page by name, filling form fields, or clicking on links and buttons. Often, it is necessary to leave some commands undefined until runtime, e.g., the number of clicks on a "next" link, which depends on the results returned. The authors also propose DeepBot [1], which is another deep Web crawler based on NSEQL, and which is able to automatically fill in forms using an advanced form filling model supplied by the user. DeepBot discards forms that do not offer information about the domain of interest.

Baumgartner et al. [7] propose Lixto Visual Wrapper, a Web wrapper generation tool. It provides an Eclipse-based interface which embeds a Mozilla browser that captures user actions on a Web page, stores them in an XML-based script, and replays them when necessary. Paths include actions that involve mouse or keyboard user events, such as following a link, filling in a text field in a form, or selecting an option in a select field. The HTML elements that are involved in each action are identified by means of XPath expressions. More complex actions can be also modelled using Elog, a logic-based programming language. Their proposal includes some handlers to deal with unexpected events, such as popup dialogues that were not displayed while the path was being recorded.

Montoto et al. [65] propose a recorder that captures user actions building on DOM events (e.g., clicking on a button, or filling in a text field). In contrast to the previous proposals, they consider every possible DOM event that involves any HTML element (e.g., moving the mouse over an element or entering a character in a text field), and they deal with AJAX requests. Another difference is that the user has to explicitly select each element and choose on a menu which of the events available she wishes to execute; this helps discern irrelevant events easily. The recorder implementation is based on the Internet Explorer programming interface to record user actions, but the resulting paths can be executed using either Internet

Explorer or Firefox. The authors propose an algorithm to build robust XPath expressions to identify HTML elements in the path. Their technique is an extension of the WARGO system [72]. The same authors also propose Exp-IE [57], which is an implementation of a custom browser that is able to record user navigation steps. Exp-IE allows to analyse the DOM tree of the navigated pages to build XPath-like expressions that point to the nodes in the tree that are not necessary to correctly replay the user actions. These nodes can be safely discarded when the page is loaded, saving time and resources when the path is replayed. The same principle is applied to detect which DOM events can be safely ignored when the page is loaded, since they do not affect the recorded user action.

Selenium [37] is a widely-used open-source project for testing automation. It provides Selenium IDE, a recorder that can be integrated in Web browsers to capture user actions and translate them into a path that can be later replayed. Paths can also be handcrafted building on a programming interface of predefined commands that is provided by the recorder. These commands implement usual interactions with Web pages, such as opening a Web page, typing text in a form field, or clicking on buttons and links. Scripts created in the recorder are stored in HTML format. Furthermore, Selenium provides libraries for different programming languages, such as Perl, PHP, C#, Java, Python, or Ruby, which allow defining paths that take advantage of the functionalities offered by those languages. These paths are executed on the Selenium Remote Control Server, which is responsible for launching and shutting down browsers and acts as a proxy for the requests made to the browsers.

Similar to Selenium, other testing automation tools allow as well to automatise crawling tasks in a particular Web site, and they provide an API that allows defining these tasks in a given programming language. Therefore, the different crawling tasks are actually defined by the user using the API provided by the automation tool in a particular programming language. In some cases, these tools include a GUI that allows the user to perform these tasks on a browser-like interface and they automatically generate the program that implements them using their API. Some examples of automation tools are Watij, Watir and Watin [92–94] (for Java, Ruby or .Net language, respectively), iMacros [39] (for a custom programming language named Macros), HTTPUnit (for Java) [38], Scrapy [78] (for Python), and PhantomJS, ZombieJS, Chromeless, or Nightwatch (for Javascript) [20, 68, 74, 108].

Example 5.1 In our motivating example, a recording path could include the following steps: {move mouse to field Destination in form, type ‘New York City’, move mouse to check in date calendar, wait until the calendar is displayed, click on date ‘19/06/2015’, move mouse to check out date calendar, wait until the calendar is displayed, click on date ‘20/06/2015’, move mouse to Guests combobox, select ‘2 adults’, click Search button, wait until hub page is loaded, click on link ‘New York Marriott Marquis’}.

5.2 Supervised learners

Other techniques do not exactly repeat the steps in the user sample navigation path; instead, they generalise the user steps to learn a model, that allows them to crawl and reach not only the same pages navigated by the user, but also other similar pages from the same Web site.

Blythe et al. [11] propose a supervised technique called EzBuilder. To create a crawling path for a Web site, the user has to provide some examples, submit forms, and navigate to several relevant pages. The number of examples depends on the regularity of the site. User actions, such as clicking buttons or filling forms, are captured by the system. Then, EzBuilder generalises the user’s behaviour to create a path, i.e., the path is not a repetition of the user steps, but a general sequence of steps that must be performed to reach pages that

are similar to the ones that were provided as examples. The resulting paths may require the user to provide additional data to fill in forms; the information extracted from the relevant pages is returned in XML format.

Zhao and Wang [104] propose Nautilus, a framework that supports the design and implementation of deep Web crawling tasks. Their framework includes some common functionalities that are useful for Web crawling, such as error page detection, and allows users to define their particular crawling tasks using a XML-based language named Nemo Description. Nautilus is supported by a fault-tolerant distributed computing system that allows running different crawling tasks in parallel, making the crawling process more efficient.

Furche et al. [28] propose OXPath, an extension for the XPath language that allows defining a sequence of interactions with the pages of a Web site. While XPath expressions point to a particular collection of nodes inside the DOM tree of a Web page, OXPath expressions allow to interact with those nodes as well, i.e., by clicking on a link in an anchor node, filling in a form field in an input text node with a value provided by the user, or extracting the information contained in a text node, amongst others. The evaluation of the OXPath expression involves performing the defined actions over the corresponding nodes to reach the ultimate nodes that have to be extracted, and the result of the evaluation is the collection of extracted nodes, organised as a tree. Therefore, OXPath can be used to generate supervised crawling paths that perform user-centric crawling and information extraction tasks.

Example 5.2 In our motivating example, a supervised-learnt crawler could perform the following steps: {fill in form, click Search button, wait until hub page is loaded, click link whose URL is similar to http://www.booking.com/hotel/en/*}.

5.3 Unsupervised learners

Some of the proposed techniques are unsupervised; they analyse a Web site and learn crawling paths with no user intervention. These proposals are based on some sample data or Web page and automatically learn a crawling path that reaches Web pages in the site that contain some of the sample data, similar data, or Web pages that are similar to the sample page.

Lage et al. [53] propose a blind unsupervised technique to learn crawling paths in Web sites that follow a common path template, e.g., paths start by filling a search form, which after submission returns a hub page that contains links to detail pages. This technique is supported by a repository of sample data, which consists of objects specified by a set of attribute-value pairs. To create a path, their technique first performs a blind search to find a search form; then, it tries to find a correspondence between the form labels and the attributes in the repository to fill in and submit the form. The response page is analysed: if it contains any of the sample data, it is a detail page; otherwise, if it contains links with anchors similar to "Next", or links with non-alphabetical anchor text (e.g., "≫"), it is a hub, its links are followed and the target pages are characterised similarly. Finally, the previous steps are encoded as a Java program.

Caverlee et al. [14] propose an unsupervised approach that relies on a structure-based Web page classifier. This classifier is based on the fact that Web pages are usually generated by means of server-side templates that provide the structure of the pages and have placeholders that must be filled in with data by means of server-side scripts. As a consequence, Web pages that are generated by the same template are likely to belong to the same class. Therefore, if the templates associated to each class are known beforehand, a Web page can be classified by assigning it to the class whose template is more similar.

Vidal et al. [89] propose another structure-based unsupervised technique that receives a sample page as input, which represents the class of pages that are considered relevant, and it returns a crawling path, which is composed of the sequence of regular expressions that represent the URLs that lead to relevant pages in a Web site. In this context, the structural similarity is measured in terms of the tree-edit distance between DOM trees. Their technique involves two steps: site mapping and pattern generation. Site mapping consists of building a map of the Web site, which requires to crawl the entire site. They keep record of the paths in the map that lead (directly or indirectly) to pages that are similar to the sample page (target pages). Then, pattern generation consists of generalising the URLs of the pages in the former paths using regular expressions, and then selecting the best path, i.e., the one that leads to the largest number of target pages. They propose some heuristics to deal with form submission and the processing of the response pages, which are classified as either relevant (added to the site map), error page (discarded), hub page (some of its links will be followed), or form page (its form is submitted and the same heuristics are applied to classify the resulting page).

A different approach consists of classifying the pages according to the terms that they contain [1, 33, 53, 55, 59, 101]. In these techniques, an ordered dictionary of terms is used as a reference, and each page is represented as a vector of term frequencies, i.e., each position stores the frequency of appearance in the page of the corresponding term from the dictionary. Web pages that belong to the same class usually include similar terms. Therefore, a Web page can be classified by assigning it to the class whose characteristic vector of terms is more similar.

Furche et al. [29] propose a system to extract information from deep Web sites in different domains, with no user intervention. Their system includes a crawler among its components, which is responsible for reaching pages that contain relevant information inside the site. Their crawler is focused, which means that it classifies pages according to the terms that they contain and is thus able to select pages that contain terms that are related to one or various topics.

URL-based classifiers [10, 34, 35] build on features of the Web page URL to classify it. Therefore, it can be classified without actually downloading it, which has a positive impact on performance. He et al. [33] propose a semi-supervised entity-oriented technique to crawl deep Web sites, i.e., a technique in which relevant pages are exclusively detail pages that offer information about a type of entities in a given domain. They analyse the search forms of a particular Web site and the URLs that are requested to the server when those forms are submitted. Then, they generate a URL pattern that matches the former URLs. Finally, the URL patterns are filled in with values to generate a path of URLs, which allows them to crawl the Web sites without having to explicitly submit the forms. They propose an algorithm to filter no-result pages that is based on the idea that no-result pages returned by a given Web site have very similar contents. In a training phase, they submit a number of queries that are designed to not retrieve any results (e.g., by using meaningless terms), and they therefore retrieve a set of potential no-result pages. In the crawling phase, each crawled page is compared against the former set; if it is considered similar to the pages in the set, it is classified as a no-result page and ignored, otherwise it is further processed.

Example 5.3 As an example, the user may provide a collection of sample Booking.com hotel detail pages, and the crawler should automatically learn which actions lead to those pages (and other pages similar to them), which includes inferring the values to fill in the search form.

5.4 Comparison framework

We summarise the existing path learning proposals in Table 4, in which a dash symbol ('-') in a cell indicates that a proposal does not provide information about the corresponding feature. We have included some features that are related to the learning process, namely:

- **DS**: Supporting data structure that is used to store the path of each crawler.
- **GU**: Whether the proposal provides a graphical user interface that supports path definition or not.
- **IM**: Implementation technology.
- **CF**: Type of features that are used to classify the pages, i.e., page terms, their structure, or their URLs.
- **AL**: Classification algorithm used.
- **AJ**: Whether the technique is able to learn paths that deal with Web sites that use asynchronous JavaScript functions.
- **SU**: Degree of supervision of the user that is required to learn the path.

Recorders and supervised learners usually offer a browser-based interface, i.e., an interface that is either based on or that imitates a well-known Web browser, such as Internet Explorer or Mozilla Firefox. Since users are accustomed to browser interfaces, it is quite straightforward for them to learn how to use these techniques. However, it is still burdensome for the users to record the paths by hand. Furthermore, this type of techniques is not scalable due to the increasingly large number of deep Web sites.

Most proposals use a term-based classifier, i.e., one that extracts classification features from the page text. These proposals represent Web pages either as HTML text (including HTML tags) or as a bag or vector of terms (excluding HTML tags). We distinguish between two types of techniques regarding the classification algorithm used: techniques in the first group [1, 14, 33, 55, 59, 89] are based on the use of a distance or similarity measure that allows to establish relationships between similar elements. Techniques in the second group [2, 7, 11, 22, 28, 37, 57, 65, 72, 104] are not based on a well-known classification algorithm but on an ad-hoc technique, which only classifies as relevant those pages that the user has selected.

The main disadvantage of devising ad-hoc techniques is the effort that is required to devise those techniques. We believe that crawling proposals should benefit from the existing classification machine-learning algorithms, specially with the latest advances in big data processing that have led to a number of efficient machine learning platforms. Furthermore, these techniques should focus on URL-based classification, since it significantly reduces the number of Web pages downloaded, and improves the crawling efficiency.

Most of the proposals are not able to deal with asynchronous JavaScript functions, although the latest proposals are focusing on dealing with this problem. As we will discuss in Section 7, this is one of the most urgent concerns in this area, and should be addressed by future proposals due to the rapid pace of change that we are witnessing in Web development technologies.

6 Performance measures comparison framework

Deep Web crawling proposals need to be able to measure their effectiveness and efficiency by means of experimental empirical results on a number of datasets [28].

Table 4 Crawling paths learning capabilities: DS (Data structure), GU (Graphical user interface), IM (Implementation technology), CF (Type of classification features), AL (Classification algorithm), AJ (Asynchronous JavaScript support), SU (Supervision)

Year	Proposal	DS	GU	IM	CF	AL	AJ	SU
1998	Chakrabarti et al. [15]	URL queue	Tcl/Tk API	C, Tcl/Tk	Link	Based on HITS	No	No
1999	Davulcu et al. [22]	F-logic / transactional logic	Browser	Transactional F-logic	–	Ad-hoc	No	Yes
2000	Anupam et al. [2]	XML	Applet	Applet / Web server	–	Ad-hoc	No	Yes
2001	Raghavan and Garcia-Molina [75]	URL queue	–	–	–	–	No	No
2002	Pan et al. [72]	NSEQL	Browser	Browser	No	Ad-hoc	No	Yes
2004	Caverlee et al. [14]	Java program	No	Java	Structure	K-Means / Cosine similarity	No	No
2004	Lage et al. [53]	Java program	No	Java	Terms	Heuristics	No	No
2005	Baumgartner et al. [7]	XML + Elog	Browser	Browser	–	Ad-hoc	No	Yes
2006	Holmes and Kellogg [37]	HTML	Browser / API	Browser	–	Ad-hoc	Yes	Yes
2007	Álvarez et al. [1]	NSEQL	Browser	Browser API	Terms	TF-IDF / Jaro-Winkler edit distance	Yes	Yes
2008	Blythe et al. [11]	SPARK procedure	Browser	–	–	Ad-hoc	No	Yes
2008	Madhavan et al. [59]	URL queue	–	–	Terms	Clustering	No	No
2008	Vidal et al. [89]	URL regular expressions	No	Java	Structure	Tree-edit distance	No	No
2009	Montoto et al. [65]	HTML events list	Browser	Browser	–	Ad-hoc	Yes	Yes
2012	Liakos and Ntoulas [55]	HTML queue	–	–	Terms	Naive-Bayes / Cosine similarity	–	Yes
2012	Zhao and Wang [104]	Nemo Description	No	Python	–	Ad-hoc	–	Yes
2013	Furche et al. [28]	OXPath	Eclipse RC Platform	Java	–	Ad-hoc	Yes	Yes
2013	He et al. [33]	URL patterns	–	–	URL / Terms	Jaccard Similarity	No	Yes
2014	Losada et al. [57]	HTML DOM, events list, XPath	Browser	Java	–	Ad-hoc	Yes	Yes
2014	Xu et al. [101]	URL queue	–	Weka	Terms / link	TF-IDF	No	Yes
2014	Furche et al. [29]	OXpath	–	Hadoop, WebDriver, DLV	Terms	–	Yes	No

6.1 Comparison framework

We have analysed some features of the existing proposals that refer to the measures and experimental datasets that are used to evaluate the crawler performance, namely:

- **PE**: Measures that are used to assess the goodness of the crawler.
- **TS**: Type of dataset on which the experiments are performed, either composed by real-world Web pages or by synthetic pages created ad-hoc for the experiment.
- **DS**: Size of each dataset. Each size represents the number of elements that compose each dataset, and it is expressed as X/Y, where X refers to the number of Web sites and Y to the number of pages from those sites that have been retrieved.
- **SD**: Semantic domain that defines the pages in each dataset. Datasets that include pages from different domains are labelled as "Multi".
- **AV**: Public availability of the dataset (e.g., as a Web resource).

We summarise the results of our analysis regarding the methodology used by each proposal to measure its performance in Table 5, in which a dash symbol ('-') in a cell indicates that a proposal does not provide any details about the corresponding feature.

Regarding performance measures, most of the proposals use measures that are similar to the well-known precision and recall. Note that precision and recall are measures that have traditionally been proposed in the information retrieval research field, but their use has extended to other areas, such as Web crawling.

The precision of a crawler refers to the proportion of the number of crawled relevant pages to the total number of crawled pages. In this context, it is also referred to as harvest rate in the proposal by Li et al. [54]. In the proposal by Chakrabarti et al. [15], the precision is not automatically calculated, but assigned by a human user on a scale from 1 to 10, and referred to as accuracy.

The recall of a crawler refers to the proportion of the number of pages crawled to the total number of pages in the Web that are relevant. In the crawling context, the actual set of pages that are relevant in the Web is unknown. Therefore, the recall is usually calculated as the proportion of the number of pages crawled to the number of pages in a target relevant set, given by the user, where the target relevant set is a subset of the actual relevant set. In this context, it is also referred to as coverage [54]. In [15], the recall is not automatically calculated, but assigned by a human user on a scale from 1 to 10, and referred to as comprehensiveness.

However, other measures are used as well to evaluate the goodness of crawlers: Álvarez et al. [1] use F_1, which is the harmonic mean of precision and recall; Chakrabarti et al. [15] use robustness besides precision and recall, where robustness is calculated as the number of common URLs crawled starting with different seeds; Montoto et al. [65] use effectiveness, which is calculated as the number of crawling scripts that are correctly executed; other proposals use the crawler training and execution times [11, 22, 57, 72].

As we observed earlier, the recall measure makes little sense in the deep Web crawling context, since it is usually impossible to compute the set of relevant pages or its size (note that that is actually a different challenge). Computing the target recall is not the best alternative, either, since it imposes on the user the burden of having to compile the target set. Therefore, a conclusion is that a new measure should be defined to represent the amount of relevant information that a crawler has retrieved and how close is that amount to the desirable amount. Note that retrieving a large number of relevant results implies in most cases increase significantly the execution time, as seen previously in this survey. Since efficiency

Table 5 Performance measures capabilities: PE (Performance measures), TS (Type of test set), DS (Dataset size), SD (Semantic domain), AV (Availability)

Year	Proposal	PE	TS	DS	SD	AV
1998	Chakrabarti et al. [15]	Human-based, accuracy/precision, comprehesiveness/recall	Real	–	Multi	No
1999	Davulcu et al. [22]	CPU execution time	Real	10/33	Cars	No
2000	Anupam et al. [2]	No measure provided	–	–	–	No
2001	Raghavan and Garcia-Molina [75]	Submission efficiency (%succesful submissions), accuracy	Real	50/–	Semiconductors, movies, computer sciences	No
2002	Pan et al. [72]	Training time	Real	400/–	Finantial, shopping, e-mail, news	No
2004	Caverlee et al. [14]	Execution time, average entropy, precision, recall	Real / Synthetic	50/5.5M	Multi	No
2004	Lage et al. [53]	Precision, recall	Real	30/–	Books, CDs, software	No
2005	Baumgartner et al. [7]	–	–	–	–	No
2006	Holmes and Kellogg [37]	–	–	–	–	No
2007	Álvarez et al [1]	Precision, recall, F_1	Real	10/–	Books, music, movies	No
2008	Blythe et al. [11]	Wrapper generation time, number of steps	–	11	Office, travel	No
2008	Madhavan et al. [59]	Forms/Keywords extracted, %records retrieved, URLs generated/form	Real	10/5M	Multi	No
2008	Vidal et al. [89]	Precision, recall	Real	17/3106	Multi	Yes
2009	Montoto et al. [65]	Effectiveness	Real	75/–	Multi	No
2012	Liakos and Ntoulas [55]	Precision, recall	Real	2/5M	Multi	No
2012	Zhao and Wang [104]	Execution time	Real	12/7M	Universities	No
2013	Furche et al. [28]	Memory usage, execution timings	Real	–/140M	Multi	No
2013	He et al. [33]	Precision, recall	Real	10/–	Multi	No
2014	Losada et al. [57]	Efficiency	Real	28/246	Multi	No
2014	Xu et al. [101]	Volume of downloaded pages, precision, cumulative precision	Real	–/1781	Health	No
2014	Furche et al. [29]	Wrapper generation time, % sites with effective wrapper, precision, form labelling accuracy, extraction accuracy, extraction time	–	–/10602	Real state, used cars	Yes

is a main concern for most of the authors in the literature, this measure should consider as well the time invested by the crawler to retrieve the relevant pages.

Regarding the datasets used for calculating the performance measures, all the analysed proposals rely on datasets composed of real-world Web pages. This is a sensible approach, since it allows the authors to draw conclusions about the applicability of these crawlers in a real-world context. Only one proposal [14] includes a supplementary synthetic dataset. The use of synthetic datasets is only advisable when the goal is to test a crawler against a Web site with some features, that are not shared by any other site in a real-world dataset. The reason behind this might be that the existing real-world datasets are insufficient or out-of-date, which would be another reason for investing time in compiling improved and up-to-date datasets. However, another reason might be that there do not exist in the deep Web actual Websites with that features. In the latter case, the authors should reflect on whether testing their crawler against such fictitious site is actually worthwhile.

Most of the techniques are evaluated on datasets composed of pages from different domains, instead of focusing on a single domain. Therefore, the current trend is to devise doman-independent proposals, despite the observation of some authors that it is better to devise domain-specific form filling proposals [47]. Note that this is not necessarily a contradiction, but it means that domain-independent crawling techniques should be devised as frameworks in which different form filling proposals may be used.

The sizes of the datasets vary from around 30 sample pages to 140M pages. Logically, the tendency is towards larger datasets, since the purpose of the experimentation is to test the performance of crawlers in a realistic context, and the size of the deep Web is still growing. Obviously, retrieving a large dataset requires some costly effort, which increases if the dataset has to be annotated by a user. The problem of generating large annotated datasets with a minimum effort from the user has already been addressed in other research areas, e.g., with the application of active learning [79] and other semi-supervised techniques [107].

Finally, all of the analysed proposals are evaluated using completely different datasets, which, in most cases, are not publicly available. Only the proposal by Vidal et al. [89] uses some data from a previously published available repository, and Furche et al. [29] have made their datasets available online. Note that to make a dataset publicly available, it does not suffice to provide the URLs or names of the Web pages and Web sites that compose the dataset, since Web sites change frequently, which means that the current version of a page that is currently available in a Web site is probably quite different from the version of the same page that was available when it was retrieved for the experiment. Moreover, these Web pages and Web sites might not even exist anymore. Therefore, to make a dataset available, it is mandatory to create a snapshot of its Web pages at the moment that they are retrieved, and make this snapshot publicly available.

7 Future directions

After the analysis of the proposals in the former sections, we can conclude that deep Web crawling is a thoroughly researched area. However, there are still a number of open issues that should be addressed in the near future, as shown by the latest works in this area. Next, we discuss these issues, grouped according to the framework dimension that they refer to:

1. Discovery of deep Web sites:
   - Estimate the size of the deep Web [31].

- Index the deep Web [31].

2. Form filling:
   - Improve the form filling capabilities of the crawler, in order to reach a larger number of relevant pages [22, 31, 60, 75], and minimise the number of form submissions [3, 46, 66].
   - Automatise some tasks related to form filling, such as retrieving the values to fill in the forms from the site that is being crawled, instead of relying on the user or other external sources, or inferring the semantics associated to each form field, in order to create complex models that allow to fill them in more effectively and efficiently [29].
3. Crawling paths learning:
   - Devise crawlers that are able to deal with common-use Web technologies [53, 60]. In the last few years, we have witnessed a significant evolution in the way that Web pages and sites are conceived and implemented, mainly with the adoption of scripting languages like EcmaScript or TypeScript, and the use of complex CSS frameworks to create increasingly dynamic and user-friendly Web interfaces. Most crawling techniques are based on the analysis of the HTML code of the Web pages, disregarding many aspects that have an impact on the resulting page, such as client side scripts, asynchronous functions, or CSS styles, some of which may even alter the DOM tree and contents of the HTML page [95]. Furthermore, crawlers are usually neglecting elements that may contribute to a more efficient and effective performance, such as HTML5 semantic tags or microformats, amongst others. Therefore, this issue, which was already observed by Olston and Najork [70] in 2010, is currently still a challenge. Future crawling proposals should take into account the latest trends in Web development to result in more effective and efficient crawlers able to deal with current Web interfaces.
   - Improve the access to the deep Web from mobile devices [43, 66], which impose some restrictions on the amount of computing resources available for the crawler. The use of mobile devices to search information in the Web has grown in the latest years, to the point that in 2016, for most information domains, more than 50% of the queries made in US to search engines were issued from mobile devices, which is confirmed by recent reports [26]. Mobile devices accounted for 49.74% of Web page views worldwide in 2017 [85], and their use is currently on the rise.
   - Automatically update the crawling definitions when major changes are made to the Web site that invalidate the existing definitions [57, 72].
4. Performance measures:
   - Improve the crawler effectiveness, e.g., by using different types of features to classify [89], integrating information extraction techniques [2, 11, 14], or using feedback information [54].
   - Improve the crawler efficiency, e.g., by choosing an optimal set of seed pages [90], or creating distributed (also known as mobile or migrating crawlers) [50]. Note that this also results in better scalability, which is another main concern [70].
   - Automatically update the crawled pages to improve their freshness [19, 31, 71, 73, 95, 104].

- Devise a comprehensive methodology to measure the efficiency and effectiveness of crawling techniques. As we mentioned in Section 6, the performance of the existing proposals is measured using different measures, performing experiments over different datasets. Therefore, the performance results cannot be compared to one another, and no conclusion can be drawn regarding which proposal performs better (more effectively or efficiently) than the others in each case. Furthermore, the recall measure, which is commonly used in this and other areas, is of little use for crawling proposals. Our conclusion is that new measures are needed to assess how much relevant information has a crawler retrieved considering the amount of time invested in doing it.
- To the best of our knowledge, there does not exist a repository of standard datasets of deep Web pages, nor a general benchmark for evaluating deep Web crawling performance, which would allow the side-by-side comparison of crawlers from an empirical point of view [106].

  In the past, authors relied on the TEL-8 repository, specially for evaluating form filling proposals. However, this repository is out-of-date nowadays. The nearest approach to an up-to-date dataset repository are the datasets made available by some authors in recent proposals like Furche et al. [29], with snapshots of the pages that they have used in their experiments. However, a consensus about using these particular datasets for the experiments does not exist as in other areas of research, such as machine learning.
- We have only found a proposal for a focused crawler evaluation framework [84], which has some limitations, namely: it only applies to focused crawlers, and it requires a hierarchical topic directory such as The Open Directory. We believe that more efforts should be made to create more testing environments and benchmarks that allow to empirically compare crawling proposals side-by-side, based on statistically significant data. In turn, these results will allow drawing conclusions and making informed decisions regarding which crawler should be chosen to perform a given crawling task.

8 Conclusions

Nowadays, most of the Web sites that offer useful information belong to the deep Web, i.e., they are based on forms that must be filled in with the values provided by a user query. As a result, a deep Web crawler is needed to perform a number of steps that include automatically locating the forms, filling them in, and crawling through the results to discern relevant pages.

There are many proposals to crawl deep Web pages in the literature, but unfortunately, there is not a survey that allows comparing them extensively. In this article, we have surveyed deep Web crawling proposals and compared them regarding different aspects, namely: automated discovery capabilities, form filling capabilities, path learning capabilities, and the measures and datasets used to evaluate their performance. From the analysis of the proposals in our survey, we can conclude that even though deep Web crawling is a thoroughly researched area, a number of research challenges remain to be explored by future proposals.

In the present day, it is essential to devise crawlers that are able to deal with ever-evolving Web technologies. The latest trends in Web design technologies are giving an ever increasing role to elements external to the HTML code of Web pages, such as client-side scripts, CSS styles and asynchronous functions; as a consequence, crawlers should not rely on the

mere analysis of the HTML code, but on the analysis of rendered pages, probably with the help of headless browsers such as Puppeteer or Chromeless.

It is important to take into account the crawlers' efficiency and effectiveness on their design, and focus on improving them by using different techniques. Efficiency and scalability are two main concerns that should be addressed by developers of Web systems in general, and deep Web crawlers in particular.

Authors should be able to measure the performance of their crawlers using standard measures and datasets, which make their performance results comparable to those of other proposals. To the best of our knowledge, there does not exist a testing environment that allows performing experiments on datasets composed of up-to-date Web pages and measuring the crawlers performance using a number of standardised measures, which in turn would allow the side-by-side comparison of crawlers. These datasets should be large enough as to provide a realistic experimentation environment, which means that the process of annotating them should require a minimal supervision from the user in order to make it scalable. Furthermore, we have detected the lack of a good measure that replaces recall as an standard measure due to its limitations in the deep Web crawling context.

Finally, we can conclude that even though the different types of proposals analysed in this survey have the same ultimate goal (to crawl deep Web sites), some of them are better suited than others depending on the particular context in which the crawling is being performed. Therefore, the application for which a crawler is devised should be taken into consideration when assessing its performance, as has already been noted by other authors regarding form filling techniques. For example, a distinction should be made between crawling to find general information that can be found in a large number of pages in the Web site, versus crawling to find very specific information that can only be found in a few pages. In the first case, blind and focused crawlers would be a good choice, since they are less complex and require less supervision from the user. However, in the second case, blind and focused crawlers would not be as efficient, since a large number of pages would be downloaded, classified and discarded. In this case, crawlers that automatically learn the crawling path would be a better choice, although they sometimes require a higher level of supervision.

Acknowledgements The authors would like to thank Dr. Rafael Corchuelo for his support and assistance throughout the entire research process that led to this article, and for his helpful and constructive comments that greatly contributed to improving the article. They would also like to thank the anonymous reviewers of this and past submissions, since their comments have contributed to give shape to this current version. Supported by the European Commission (FEDER), the Spanish and the Andalusian R &D & I programmes (grants TIN2016-75394-R, and TIN2013-40848-R).

References

1. Álvarez, M., Raposo, J., Pan, A., Cacheda, F., Bellas, F., Carneiro, V.: Crawling the content hidden behind Web forms. In: ICCSA, pp. 322–333 (2007). https://doi.org/10.1007/978-3-540-74477-1_31
2. Anupam, V., Freire, J., Kumar, B., Lieuwen, D.F.: Automating Web navigation with the WebVCR. Comput. Netw. **33**(1-6), 503–517 (2000). https://doi.org/10.1016/S1389-1286(00)00073-6
3. Asudeh, A., Thirumuruganathan, S., Zhang, N., Das, G.: Discovering the skyline of Web databases. PVLDB **9**(7), 600–611 (2016). https://doi.org/10.14778/2904483.2904491
4. Barbosa, L., Freire, J.: Siphoning hidden-Web data through keyword-based interfaces. In: SBBD, pp. 309–321. (2004).
5. Barbosa, L., Freire, J.: Searching for hidden-Web databases. In: WebDB, pp. 1–6 (2005)

6. Barbosa, L., Freire, J.: An adaptive crawler for locating hidden-Web entry points. In: WWW, pp. 441–450 (2007). https://doi.org/10.1145/1242572.1242632
7. Baumgartner, R., Ceresna, M., Ledermuller, G.: Deep Web navigation in Web data extraction. In: CIMCA/IAWTIC, pp. 698–703 (2005). https://doi.org/10.1109/CIMCA.2005.1631550
8. Bergholz, A., Chidlovskii, B.: Crawling for domain-specific hidden Web resources. In: WISE, pp. 125–133 (2003). https://doi.org/10.1109/WISE.2003.1254476
9. Bergman, M.K.: The deep Web: Surfacing hidden value. J. Electron. Publ. **7**, 1 (2001).
10. Blanco, L., Dalvi, N., Machanavajjhala, A.: Highly efficient algorithms for structural clustering of large Webs ites. In: WWW, pp. 437–446 (2011). https://doi.org/10.1145/1963405.1963468
11. Blythe, J., Kapoor, D., Knoblock, C.A., Lerman, K., Minton, S.: Information integration for the masses. J UCS **14**(11), 1811–1837 (2008). https://doi.org/10.3217/jucs-014-11-1811
12. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: A collaboratively created graph database for structuring human knowledge. In: SIGMOD, pp. 1247–1250 (2008). https://doi.org/10.1145/1376616.1376746
13. Calì, A., Martinenghi, D.: Querying the deep Web. In: EDBT, pp. 724–727 (2010). https://doi.org/10.1145/1739041.1739138
14. Caverlee, J., Liu, L., Buttler, D.: Probe, cluster, and discover: Focused extraction of qa-pagelets from the deep Web. In: ICDE, pp. 103–114 (2004). https://doi.org/10.1109/ICDE.2004.1319988
15. Chakrabarti, S., Dom, B., Raghavan, P., Rajagopalan, S., Gibson, D., Kleinberg, J.M.: Automatic resource compilation by analyzing hyperlink structure and associated text. Comput. Netw. **30**(1-7), 65–74 (1998). https://doi.org/10.1016/S0169-7552(98)00087-7
16. Chang, K.C.C., He, B., Li, C., Patel, M., Zhang, Z.: Structured databases on the Web: Observations and implications. SIGMOD Record **33**(3), 61–70 (2004). https://doi.org/10.1145/1031570.1031584
17. Chang, K., He, B., Zhang, Z.: Toward large scale integration: Building a metaquerier over databases on the Web. In: CIDR, pp. 44–55. (2005).
18. Chen, H.: Dark Web: Exploring and data mining the dark side of the Web. Online Inf. Rev. **36**(6), 932–933 (2012). https://doi.org/10.1108/14684521211287981
19. Cho, J., Garcia-Molina, H.: Effective page refresh policies for Web crawlers. ACM Trans. Database Syst **28**(4), 390–426 (2003). https://doi.org/10.1145/958942.958945
20. chromeless: https://github.com/graphcool/chromeless (2018)
21. Cope, J., Craswell, N., Hawking, D.: Automated discovery of search interfaces on the Web. In: ADC, CRPIT, vol. 17, pp. 181–189 (2003)
22. Davulcu, H., Freire, J., Kifer, M., Ramakrishnan, I.V.: A layered architecture for querying dynamic Web content. In: SIGMOD, pp. 491–502 (1999). https://doi.org/10.1145/304182.304225
23. Devine, J., Egger-Sider, F.: Beyond google: The invisible Web in the academic library. J. Acad. Librarianship **30**(4), 265–269 (2004). https://doi.org/10.1016/j.acalib.2004.04.010
24. Dragut, E.C., Kabisch, T., Yu, C., Leser, U.: A hierarchical approach to model Web query interfaces for Web source integration. PVLDB **2**(1), 325–336 (2009). https://doi.org/10.14778/1687627.1687665
25. Dragut, E.C., Meng, W., Yu, C.T.: Deep Web Query Interface Understanding and Integration. Synthesis Lectures on Data Management. Morgan & Claypool (2012). https://doi.org/10.2200/S00419ED1V01Y201205DTM026
26. Fetto, J.: Mobile search: Topics and themes. report, Hitwise (2017)
27. Furche, T., Gottlob, G., Grasso, G., Guo, X., Orsi, G., Schallhart, C.: The ontological key: Automatically understanding and integrating forms to access the deep Web. VLDBJ **22**(5), 615–640 (2013). https://doi.org/10.1007/s00778-013-0323-0
28. Furche, T., Gottlob, G., Grasso, G., Schallhart, C., Sellers, A.J.: OXPath: A language for scalable data extraction, automation, and crawling on the Deep Web. VLDB J **22**(1), 47–72 (2013). https://doi.org/10.1007/s00778-012-0286-6
29. Furche, T., Gottlob, G., Grasso, G., Guo, X., Orsi, G., Schallhart, C., Wang, C.: DIADEM: Thousands of Websites to a single database. PVLDB **7**(14), 1845–1856 (2014). https://doi.org/10.14778/2733085.2733091
30. Green, D.: The evolution of Web searching. Online Inf. Rev. **24**(2), 124–137 (2000). https://doi.org/10.1108/14684520010330283
31. He, B., Patel, M., Zhang, Z., Chang, K.C.C.: Accessing the deep Web: A survey. Commun ACM **50**(5), 94–101 (2007). https://doi.org/10.1145/1230819.1241670
32. He, H., Meng, W., Lu, Y., Yu, C.T., Wu, Z.: Towards deeper understanding of the search interfaces of the Deep Web. In: WWW, pp. 133–155 (2007). https://doi.org/10.1007/s11280-006-0010-9
33. He, Y., Xin, D., Ganti, V., Rajaraman, S., Shah, N.: Crawling deep Web entity pages. In: WSDM, pp. 355–364 (2013). https://doi.org/10.1145/2433396.2433442

34. Hernández, I., Rivero, C.R., Ruiz, D., Corchuelo, R.: Towards discovering conceptual models behind Web sites. In: ER, pp. 166–175 (2012). https://doi.org/10.1007/978-3-642-34002-4_13
35. Hernández, I., Rivero, C.R., Ruiz, D., Corchuelo, R.: CALA: An unsupervised URL-based Web page classification system. Knowl.-Based Syst. **57**(0), 168–180 (2014). https://doi.org/10.1016/j.knosys.2013.12.019
36. Hicks, C., Scheffer, M., Ngu, A., Sheng, Q.Z.: Discovery and cataloging of deep Web sources. In: IRI, pp. 224–230 (2012). https://doi.org/10.1109/IRI.2012.6303014
37. Holmes, A., Kellogg, M.: Automating functional tests using selenium. In: AGILE, pp. 270–275 (2006). https://doi.org/10.1109/AGILE.2006.19
38. HTTPUnit: http://httpunit.sourceforge.net/ (2016)
39. iMacros: http://imacros.net/ (2016)
40. Jamil, H.M., Jagadish, H.V.: A structured query model for the deep relational Web. In: CIKM, pp. 1679–1682 (2015). https://doi.org/10.1145/2806416.2806589
41. Jiang, L., Wu, Z., Feng, Q., Liu, J., Zheng, Q.: Efficient deep Web crawling using reinforcement learning. In: PAKDD, pp. 428–439 (2010). https://doi.org/10.1007/978-3-642-13657-3_46
42. Jiménez, P., Corchuelo, R.: Roller: A novel approach to Web information extraction. Knowl. Inf. Syst., 1–45 (2016). https://doi.org/10.1007/s10115-016-0921-4
43. Jin, X., Mone, A., Zhang, N., Das, G.: Mobies: Mobile-interface enhancement service for hidden Web database. In: SIGMOD, pp. 1263–1266 (2011). https://doi.org/10.1145/1989323.1989471
44. Jin, X., Zhang, N., Das, G.: Attribute domain discovery for hidden Web databases. In: SIGMOD, pp. 553–564 (2011). https://doi.org/10.1145/1989323.1989381
45. Kabisch, T., Dragut, E.C., Yu, C.T., Leser, U.: Deep Web integration with visQI. PVLDB **3**(2), 1613–1616 (2010). https://doi.org/10.14778/1920841.1921053
46. Kantorski, G.Z., Moraes, T.G., Moreira, V.P., Heuser, C.A.: Advances in Databases and Information Systems, pp. 125–136. Springer, Berlin (2013). Chap Choosing Values for Text Fields in Web Forms
47. Kantorski, G.Z., Moreira, V.P., Heuser, C.A.: Automatic filling of hidden Web forms: A survey. SIGMOD Rec **44**(1), 24–35 (2015). https://doi.org/10.1145/2783888.2783898
48. Kautz, H.A., Selman, B., Shah, M.A.: The hidden Web. AI Mag **18**(2), 27–36 (1997). https://doi.org/10.1609/aimag.v18i2.1291
49. Khare, R., An, Y., Song, I.Y.: Understanding deep Web search interfaces: A survey. SIGMOD Rec. **39**(1), 33–40 (2010). https://doi.org/10.1145/1860702.1860708
50. Kumar, M., Bhatia, R.: Design of a mobile Web crawler for hidden Web. In: RAIT, pp. 186–190 (2016)
51. Kushmerick, N.: Learning to invoke Web forms. In: CoopIS, pp. 997–1013 (2003). https://doi.org/10.1007/978-3-540-39964-3_63
52. Kushmerick, N., Thomas, B.: Adaptive information extraction: Core technologies for information agents. In: Intelligent Information Agents - The AgentLink Perspective, pp. 79–103 (2003). https://doi.org/10.1007/3-540-36561-3_4
53. Lage, J.P., da Silva, A.S., Golgher, P.B., Laender, A.H.F.: Automatic generation of agents for collecting hidden Web pages for data extraction. Data Knowl Eng **49**(2), 177–196 (2004). https://doi.org/10.1016/j.datak.2003.10.003
54. Li, Y., Wang, Y., Du, J.: E-FFC: An enhanced form-focused crawler for domain-specific deep Web databases. J Intell Inf Syst **40**(1), 159–184 (2013). https://doi.org/10.1007/s10844-012-0221-8
55. Liakos, P., Ntoulas, A.: Topic-sensitive hidden-Web crawling. In: WISE, pp. 538–551 (2012). https://doi.org/10.1007/978-3-642-35063-4_39
56. Liddle, S.W., Embley, D.W., Scott, D.T., Yau, S.H.: Extracting data behind Web forms. In: Workshop on Conceptual Modeling Approaches for e-Business, pp. 402–413 (2002). https://doi.org/10.1007/b12013
57. Losada, J., Raposo, J., Pan, A., Montoto, P.: Efficient execution of Web navigation sequences. WWWJ **17**(5), 921–947 (2014). https://doi.org/10.1007/s11280-013-0259-8
58. Madhavan, J., Jeffery, S.R., Cohen, S., Dong, X.L., Ko, D., Yu, C., Halevy, A.: Web-scale data integration: You can only afford to pay as you go. In: CIDR, pp. 342–350 (2007)
59. Madhavan, J., Ko, D., Kot, L., Ganapathy, V., Rasmussen, A., Halevy, A.Y.: Google's deep Web crawl. PVLDB **1**(2), 1241–1252 (2008). https://doi.org/10.14778/1454159.1454163
60. Madhavan, J., Afanasiev, L., Antova, L., Halevy, A.Y.: Harnessing the deep Web: present and future. Syst. Res. **2**(2), 50–54 (2009).
61. Manvi, Dixit, A., Bhatia, K.K.: Design of an ontology based adaptive crawler for hidden Web. In: CSNT, pp. 659–663 (2013). https://doi.org/10.1109/CSNT.2013.140
62. Mccoy, D., Bauer, K., Grunwald, D., Kohno, T., Sicker, D.: Shining light in dark places: Understanding the tor network. In: PETS, pp. 63–76 (2008). https://doi.org/10.1007/978-3-540-70630-4_5
63. Meng, X., Hu, D., Li, C.: Schema-guided wrapper maintenance for Web-data extraction. In: WIDM, pp. 1–8 (2003). https://doi.org/10.1145/956699.956701

64. Modica, G.A., Gal, A., Jamil, H.M.: The use of machine-generated ontologies in dynamic information seeking. In: CoopIS, pp. 433–448 (2001). https://doi.org/10.1007/3-540-44751-2_32
65. Montoto, P., Pan, A., Raposo, J., Bellas, F., Lopez, J.: Web navigation sequences automation in modern Websites. In: DEXA, pp. 302–316 (2009). https://doi.org/10.1007/978-3-642-03573-9_25
66. Nazi, A., Asudeh, A., Das, G., Zhang, N., Jaoua, A.: Mobiface: A mobile application for faceted search over hidden Web databases. In: ICCA, pp. 13–17 (2017). https://doi.org/10.1109/COMAPP.2017.8079749
67. Nguyen, H., Nguyen, T., Freire, J.: Learning to extract form labels. PVLDB **1**(1), 684–694 (2008). https://doi.org/10.14778/1453856.1453931
68. nightwatch: http://nightwatchjs.org/ (2018)
69. Ntoulas, A., Zerfos, P., Cho, J.: Downloading textual hidden Web content through keyword queries. In: JCDL, pp. 100–109 (2005). https://doi.org/10.1145/1065385.1065407
70. Olston, C., Najork, M.: Web crawling. Found. Trends Inf. Retriev. **4**(3), 175–246 (2010). https://doi.org/10.1561/1500000017
71. Olston, C., Pandey, S.: Recrawl scheduling based on information longevity. In: WWW, pp. 437–446 (2008). https://doi.org/10.1145/1367497.1367557
72. Pan, A., Raposo, J., Álvarez, M., Hidalgo, J., Viña, Á.: Semi-automatic wrapper generation for commercial Web sources. In: EISIC, pp. 265–283 (2002). https://doi.org/10.1007/978-0-387-35614-3_16
73. Pandey, S., Olston, C.: User-centric Web crawling. In: WWW, pp. 401–411. https://doi.org/10.1145/1060745.1060805 (2005)
74. phantomjs.org: http://phantomjs.org/ (2018)
75. Raghavan, S., Garcia-Molina, H.: Crawling the hidden Web. In: VLDB, pp. 129–138 (2001)
76. Ru, Y., Horowitz, E.: Indexing the invisible Web: a survey. Online Inf. Rev. **29**(3), 249–265 (2005). https://doi.org/10.1108/14684520510607579
77. Schulz, A., Lässig, J., Gaedke, M.: Practical Web data extraction: are we there yet? - a short survey. In: WI, pp. 562–567 (2016). https://doi.org/10.1109/WI.2016.0096
78. Scrapy: http://scrapy.org/ (2016)
79. Settles, B.: Active learning. Synthesis Lect. Artif. Intell. Mach. Learn. **6**(1), 1–114 (2012). https://doi.org/10.2200/S00429ED1V01Y201207AIM018
80. Sheng, C., Zhang, N., Tao, Y., Jin, X.: Optimal algorithms for crawling a hidden database in the Web. PVLDB **5**(11), 1112–1123 (2012). https://doi.org/10.14778/2350229.2350232
81. Shu, L., Meng, W., He, H., Yu, C.T.: Querying capability modeling and construction of deep Web sources. In: WISE, pp. 13–25 (2007). https://doi.org/10.1007/978-3-540-76993-4_2
82. Sleiman, H.A., Corchuelo, R.: A survey on region extractors from Web documents. TKDE **25**(9), 1960–1981 (2013). https://doi.org/10.1109/TKDE.2012.135
83. Sleiman, H.A., Corchuelo, R.: Trinity: On using trinary trees for unsupervised Web data extraction. IEEE Trans Knowl Data Eng **26**(6), 1544–1556 (2014). https://doi.org/10.1109/TKDE.2013.161
84. Srinivasan, P., Menczer, F., Pant, G.: A general evaluation framework for topical crawlers. Inf. Retr. **8**(3), 417–447 (2005). https://doi.org/10.1007/s10791-005-6993-5
85. Statista: Mobile internet usage worldwide. Report (2018)
86. Su, W., Wu, H., Li, Y., Zhao, J., Lochovsky, F.H., Cai, H., Huang, T.: Understanding query interfaces by statistical parsing. ACM Trans Web **7**(2), 8,1–8,22 (2013). https://doi.org/10.1145/2460383.2460387
87. Su, W., Li, Y., Lochovsky, F.H.: Query interfaces understanding by statistical parsing. In: WWW, pp. 1291–1294 (2014). https://doi.org/10.1145/2567948.2579702
88. Toda, G.A., Cortez, E., da Silva, A.S., de Moura, E.: A probabilistic approach for automatically filling form-based Web interfaces. PVLDB **4**(3), 151–160 (2010). https://doi.org/10.14778/1929861.1929862
89. Vidal, M.L.A., da Silva, A.S., de Moura, E.S., Cavalcanti, J.M.B.: Structure-based crawling in the Hidden Web. J UCS **14**(11), 1857–1876 (2008)
90. Vieira, K., Barbosa, L., Silva, A.S., Freire, J., Moura, E.: Finding seeds to bootstrap focused crawlers. World Wide Web, 1–26 (2015). https://doi.org/10.1007/s11280-015-0331-7
91. Wang, Y., Lu, J., Chen, J.: Crawling deep Web using a new set covering algorithm. In: ADMA, pp. 326–337 (2009). https://doi.org/10.1007/978-3-642-03348-3_32
92. Watij.com: http://watij.com/ (2016)
93. Watin.org: http://watin.org/ (2016)
94. Watir.com: http://watir.com/ (2016)
95. Weninger, T., Palácios, R., Crescenzi, V., Gottron, T., Merialdo, P.: Web content extraction: A metaanalysis of its past and thoughts on its future. SIGKDD Explorations **17**(2), 17–23 (2015). https://doi.org/10.1145/2897350.2897353

96. Wu, Z., Raghavan, V., Qian, H., Rama, K.V., Meng, W., He, H., Yu, C.: Towards automatic incorporation of search engines into a large-scale metasearch engine. In: WI, pp. 658–661 (2003). https://doi.org/10.1109/WI.2003.1241290
97. Wu, P., Wen, J.R., Liu, H., Ma, W.Y.: Query selection techniques for efficient crawling of structured Web sources. In: ICDE, pp. 47–56 (2006). https://doi.org/10.1109/ICDE.2006.124
98. Wu, W., Doan, A., Yu, C., Meng, W.: Modeling and extracting deep-Web query interfaces, pp. 65–90 (2009). https://doi.org/10.1007/978-3-642-04141-9_4
99. Wu, W., Zhong, T.: Searching the deep Web using proactive phrase queries. In: WWW Companion, pp. 137–138 (2013). https://doi.org/10.1145/2487788.2487854
100. Wu, W., Meng, W., Su, W., Zhou, G., Chiang, Y.Y.: Q2p: discovering query templates via autocompletion. ACM Trans Web **10**(2), 10,1–10,29 (2016). https://doi.org/10.1145/2873061
101. Xu, S., Yoon, H.J., Tourassi, G.: A user-oriented Web crawler for selectively acquiring online content in e-health research. Bioinformatics **30**(1), 104–114 (2014). https://doi.org/10.1093/bioinformatics/btt571
102. Yan, H., Gong, Z., Zhang, N., Huang, T., Zhong, H., Wei, J.: Aggregate estimation in hidden databases with checkbox interfaces. TKDE **27**(5), 1192–1204 (2015). https://doi.org/10.1109/TKDE.2014.2365800
103. Zhang, Z., He, B., Chang, K.: Understanding Web query interfaces: Best-effort parsing with hidden syntax. In: SIGMOD, pp. 107–118 (2004). https://doi.org/10.1145/1007568.1007583
104. Zhao, J., Wang, P.: Nautilus: a generic framework for crawling Deep Web. In: ICDKE, pp. 141–151 (2012). https://doi.org/10.1007/978-3-642-34679-8_14
105. Zhao, F., Zhou, J., Nie, C., Huang, H., Jin, H.: Smartcrawler: a two-stage crawler for efficiently harvesting deep-Web interfaces. IEEE Trans Serv. Comput. **9**(4), 608–620 (2016). https://doi.org/10.1109/TSC.2015.2414931
106. Zheng, Q., Wu, Z., Cheng, X., Jiang, L., Liu, J.: Learning to crawl deep Web. Inf. Syst. **38**(6), 801–819 (2013). https://doi.org/10.1016/j.is.2013.02.001
107. Zhou, X., Belkin, M.: Chapter 22 - semi-supervised learning. In: Academic Press Library in Signal Processing: Volume 1, Academic Press Library in Signal Processing, vol 1, pp. 1239–1269. Elsevier (2014). https://doi.org/10.1016/B978-0-12-396502-8.00022-X
108. zombiejs.org: http://zombie.js.org/ (2018)