Automated Ontology Instantiation from Tabular Web Sources – The AllRight System

Dietmar Jannach[a,*], Kostyantyn Shchekotykhin[b], Gerhard Friedrich[b]

[a] *Technische Universität Dortmund, 44221 Dortmund, Germany*

[b] *University of Klagenfurt, 9020 Klagenfurt, Austria*

Abstract

The process of populating an ontology-based system with high-quality and up-to-date instance information can be both time consuming and prone to error. In many domains, however, one possible solution to this problem is to automate the instantiation process for a given ontology by searching (*mining*) the web for the required instance information.

The primary challenges facing such a system include: (a) efficiently locating web pages that most probably contain the desired instance information, (b) extracting the instance information from a page, and (c) clustering documents that describe the same instance in order to exploit data redundancy on the web and thus improve the overall quality of the harvested data. In addition, these steps should require as little *seed knowledge* as possible.

In this paper, the AllRight ontology instantiation system is presented, which supports the full instantiation life-cycle and addresses the above-mentioned challenges through a combination of new and existing techniques. In particular the system was designed to deal with situations where the instance information is given in tabular form. The main innovative pillars of the system are a new high-recall *focused crawling* technique (xCrawl), a novel *table recognition* algorithm, innovative methods for document clustering and instance name recognition, as well as techniques for fact extraction, instance generation and query-based fact validation.

The successful evaluation of the system in different real-world application scenarios shows that the ontology instantiation process can be successfully automated using only a very limited amount of seed knowledge.

Key words: Ontology instantiation, Populating the Web of Data

* Corresponding author, dietmar.jannach@udo.edu, Tel: +49 231 755 7272

[1] This paper combines and significantly extends the work presented in [33–35].

1 Introduction

The vision of the World Wide Web as a huge repository of *machine-processible* information may be realized in different ways. The first option is to rely on the large-scale use of semantic annotations that refer to entities defined in formal ontologies. Examples of such resource annotation systems include emerging new technologies such as RDF [2] or microformats that allow the augmentation of human-readable content (in HTML) with machine-processible information. However, even with appropriate tool support, it is unlikely that web documents will be sufficiently (semantically) annotated on a broad scale in the near future. Techniques for *automated tagging* have been recently proposed as a possible solution for overcoming this limitation as outlined for instance in [8] or [13]. These approaches have shown that they can partially solve the annotation problem.

The other option to exploit the Web of Data is to try to automatically "re-construct" the knowledge presented in (unstructured) web documents. Several *Web Mining* and *Information Extraction* (IE) techniques have been proposed to automate this task. Some rely on the redundancy of information on the web and use statistical methods [10,15], while others use natural language processing (NLP) techniques [29,30] to extract the required knowledge from unstructured documents. It has also been shown that domain *ontologies* can support knowledge extraction, which in turn means that the information extraction problem can be generalized to the problem of finding and inserting information that matches a given ontology into the system's knowledge base, a process also referred to as *ontology instantiation* or *ontology population.* The instantiated ontology can then in turn serve as a basis for the development and provision of so-called *knowledge services* in the Semantic Web [4] .

The AllRight ontology instantiation system and its innovations presented in this work were inspired by the requirements that arose throughout the development of real-world recommendation and product-comparison services that we have implemented using the Advisor Suite framework [16,21]. These recommendation services were developed to support online customers' decision-making and purchase processes and are based on in-depth and accurate knowledge about the items being recommended. The central piece of information required in such interactive recommenders is thus all possible instances (facts) describing detailed product specifications. Keeping such specialized instances of an ontology-based application up-to-date can be a time consuming and error-prone task, particularly in the fast-paced domain of electronic consumer goods (MP3 players, laptop computers, or digital cameras). The possibility of automating this knowledge-acquisition process however soon became obvious,

[2] http://www.w3.org/TR/rdfa-syntax/

as in these domains the required item information is already available on the web, e.g., on manufacturer homepages or community portals.[3]

The goal was therefore to develop a Web Mining System (WMS) that is capable of finding and extracting this knowledge automatically. What was soon recognized, however, is that existing web mining techniques are not directly applicable, because, e.g., the instance information is in many cases presented in *tabular form.*

Although tables are generally well-suited to the human reader as they are a compact and precise form for representing information, the above-mentioned information extraction techniques do not work well with them: methods that are based on Natural Language Processing (NLP) are for instance not applicable as tables in web documents typically do not contain full sentences but rather simple attribute-value pairs describing the features of an item; clustering methods that aim to identify several documents that describe the same instance (e.g. digital camera) based on term co-occurrences do not work either, because pages with *the same sets of keywords/tokens* (from the same portal) describe *different items.* Furthermore, these tables are not necessarily constructed from underlying databases, meaning that neither SPARQL[4] nor "Hidden-Web" techniques can be used. The AllRight system takes these particularities into account and presents several new techniques for dealing with tabular information.

The contributions of the AllRight to the state of the art can be summarized as follows. Overall, we show how automatic ontology instantiation can also be applied to domains in which data is given in tabular form (such as personal information, geographical data and so forth). In terms of technical innovation, the AllRight system includes a new crawling method for the fast location of tabular descriptions, a "visual" table identification technique, a novel way of applying clustering algorithms to deal with tabular descriptions, as well as a query-based fact validation method. Given the promising evaluation results with an accuracy of about eighty percent in different domains, we finally show how Semantic Web technology, ontologies and a combination of new and existing extraction techniques can help us to automatically *mine* the web for ontology instances in the context of real-world, industrial problem settings.

Note that although the design of the AllRight system was originally motivated by the requirements of a particular application scenario, it was devised as a more general, domain independent ontology instantiation system in the sense of Alani et al. [4]: The input to the extraction system is thus a domain ontology that describes the structure of and the relations between the items

[3] Our experiences showed that the quality and level of detail of commercially available data sets is not sufficient for our purposes.
[4] http://www.w3.org/TR/rdf-sparql-query/

to be searched for. The extraction process itself is domain-independent and is structured into a series of generic tasks such as web crawling, name extraction, validation, document clustering and fact extraction.

The paper is organized as follows. Based on a motivating example from the domain of digital cameras, we first sketch the overall ontology instantiation process flow, give then technical details of the individual steps required for information extraction, and finally present evaluation results for the individual subtasks. Throughout the paper, we compare the techniques presented with those outlined in related work and discuss commonalities and differences. The paper ends with a summary and an outlook on future work.

2 System overview

Process flow. Figure 1 provides a high-level overview of the ontology instantiation process in the AllRight system in the context of the "recommender system" application scenario, where the problem consists of extracting detailed product data for a content-based recommender in an online store.

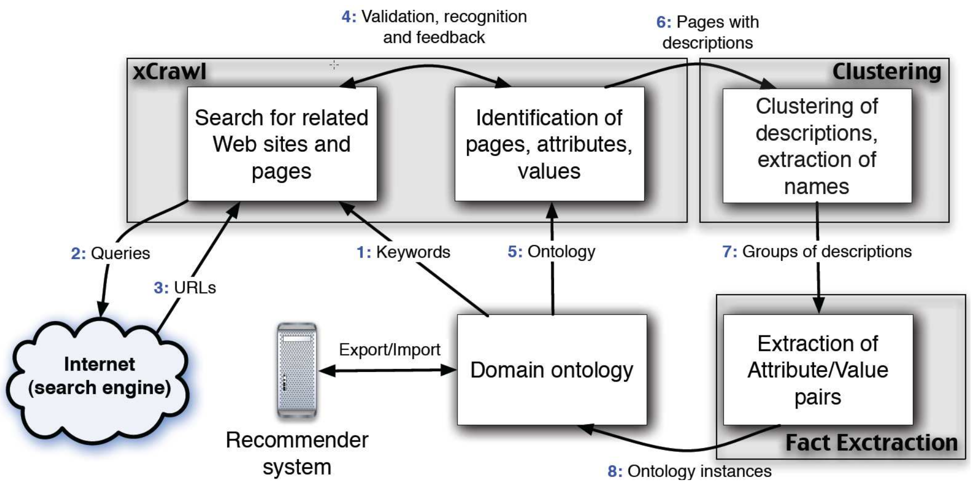

4: Validation, recognition and feedback
6: Pages with descriptions
xCrawl
Clustering
Search for related Web sites and pages
Identification of pages, attributes, values
Clustering of descriptions, extraction of names
2: Queries
3: URLs
1: Keywords
5: Ontology
7: Groups of descriptions
Internet (search engine)
Export/Import
Domain ontology
Extraction of Attribute/Value pairs
Recommender system
Fact Exctraction
8: Ontology instances

Fig. 1. AllRight ontology instantiation process in application scenario

The first step is to develop a domain ontology that describes the basic characteristics of the items being searched for. In our example configuration, some parts of the knowledge are imported from the Advisor Suite system, which can, for instance, be used to model a list of interesting item characteristics. The system then generates keywords from this domain ontology (1) that are used for crawling the web for relevant web pages (2,3) with the new xCrawl method. The downloaded pages are then analyzed by the Identification Component (validator) (4) to determine if they contain the desired item descriptions. Again, this analysis is done on the basis of the knowledge from the ontology (5). Next, the validated set of documents is forwarded (6) to a module which generates clusters of pages describing the same product, extracts the specific facts for each product and feeds the new knowledge back to the ontology (7). Note that in principle fact extraction could be performed prior to duplicate removal [27]. However, such systems require more detailed seed knowledge and specialized and complex fact recognition techniques.

Core ontology and domain ontology. In order to remain applicable to a broad spectrum of application scenarios, the AllRight system differentiates between three "levels" of ontologies, see Figure 2 for sample fragments. In the product data extraction scenario, the predefined *core ontology* forces the system to search for entities that have attributes of given types and certain units of measurement and that both the attributes and the units can be annotated with string-typed keywords. Note that in principle also other concepts and/or roles can be used (or re-used) in the core ontology. Changing the core ontology is however not easily possible in such a system because, e.g., all data access methods depend on the design of the core ontology. Still, our practical experiences show that the concepts and roles defined in the core ontology are sufficient to cover a wide range of applications areas in which data can be represented in the fom of attributes and values.

As a part of the configuration process of the AllRight system for a specific domain, e.g. digital cameras or other consumer electronics, a *domain ontology* has to be modeled based on the core ontology. In the example, we thus state that "resolution" is a property of interest for entities of type "digital camera". The resolution is given as a real-valued number and is measured in megapixels. Appropriate keywords could be "effective pixels" and "million", respectively. Note that the task of defining the domain ontology corresponds to the definition of *seed knowledge*, which is required in every Web Mining System (WMS). Technically, the domain ontology is stored as a Web Ontology Language (OWL) document. The (graphical) definition and maintenance of the model can therefore be accomplished with the help of any general-purpose ontology editing tool. In our application scenario, however, we used the built-in graphical knowledge acquisition tools from the Advisor Suite system [16].

Ultimately, the final goal of the web mining step is to find a set of ontology instances (e.g. a description of a particular *Canon* camera model) as shown on the right hand side of Figure 2.

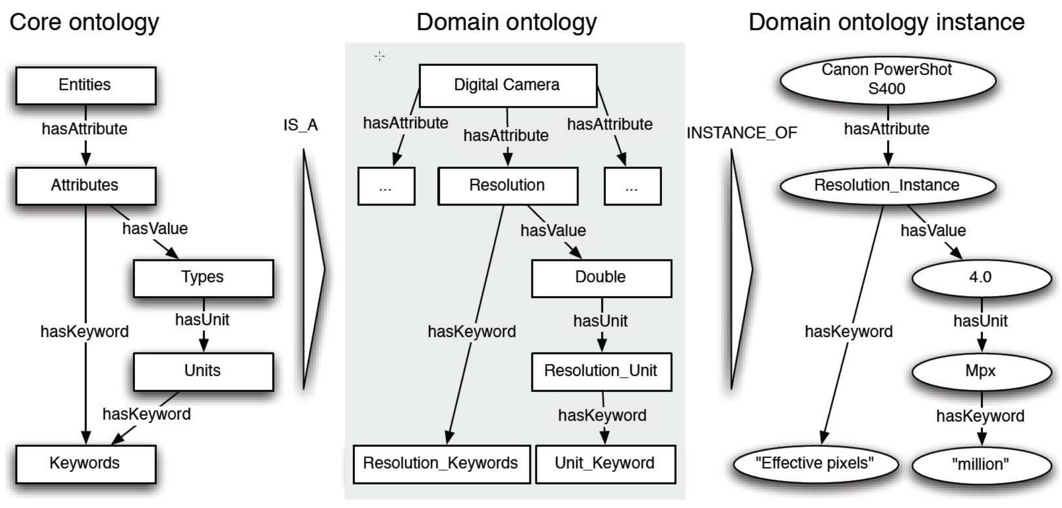


Fig. 2. Fragments of ontologies used in the AllRight system.

3 Focused crawling with xCrawl

In this section, we will describe the first phase of the ontology instantiation process (i.e. ontology-based data collection using a new *focused crawling* method) in more detail.

3.1 Background

The first goal of Web Mining Systems (WMSs) that exploit the information redundancy on the web (such as SemTag [13], Deadliner [25] or WEB→KB [10]) is to retrieve as many *relevant* documents as possible as, in general, the quality of the subsequent data extraction process strongly depends on the results of this document retrieval phase. The goal is thus not only to download many documents (in order to increase information coverage and redundancy) but also to minimize the number of documents that are irrelevant for the extraction goal.

Modern WMSs, like the ones mentioned above, are good at determining whether or not a given document contains the desired instance information, i.e. they are optimized for validating the obtained documents with a very high "precision". A higher "recall" value, which is achieved when more information is retrieved, can however also be valuable in a web mining context as the additional information can for instance be used to measure the plausibility of the extracted facts or to resolve inconsistencies. There are two general techniques for information retrieval, namely *crawling* and *searching*.

Web Crawling. "Crawling", is today's standard method for retrieving and refreshing document collections [11] in WMSs. Crawlers explore the web by automatically downloading web documents and following the links in these documents. A range of different crawling *strategies* are currently available. A simple strategy could be to start from the root of a website (homepage) and to traverse all links in the order in which they are found. This procedure can then be applied to all links that are extracted from the newly downloaded pages. More sophisticated crawlers, see for instance [32], minimize the effort of crawling, for example by identifying and ignoring different links that point to the same page.

While such strategies might be good for crawlers that do not search for specific pieces of information, in web mining scenarios such an "uninformed" and broad search is often not the best choice as in many cases the desired documents are located deep in the navigational hierarchy. Since validation of documents is time consuming and computationally expensive due to the complexity of modern validation algorithms, the speed of the WMS's retrieval process is dramatically reduced because lots of irrelevant documents have to be checked.

Focused crawlers, in contrast to breadth-first crawlers used by search engines, therefore use an informed-search strategy and try to retrieve only those parts of the web relevant to a particular topic [1,7,12,31]. Focused crawlers automatically navigate through the web (using a web browser component) similar to general-purpose crawlers and construct an internal "Web Graph" that represents the part of the web that is known to the crawler; see Figure 3 for a general architecture. The relevant documents determined by a *WMS validator* are stored in a repository. In contrast to general search engine crawlers, link (URL) selection is based on specific algorithms, which exploit the results of previous actions and aim to direct to crawler to those pages that will be accepted by the validator with the highest probability.

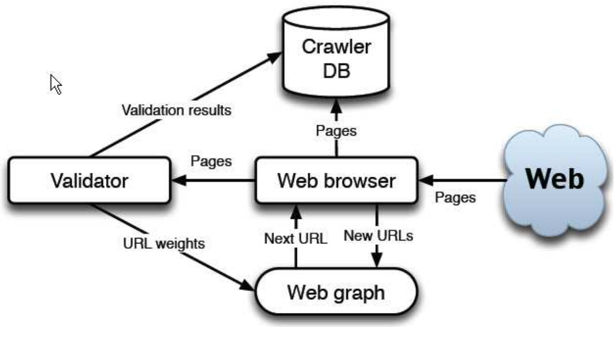


Fig. 3. A general focused crawler architecture.

Note that depending on the domain and the mining goals, different validation and extraction techniques can be used in the *WMS validator* component. Examples include learning-based methods using Support Vector Machines [39] or Bayesian Networks [10], as well as table detection techniques such as those used in the AllRight system. However, in order to ensure the broad applicability of the crawling component in different scenarios, it is desirable that the crawling algorithm of a WMS is not dependent on specific extra information from the validator component.

Searching. The other approach to automating the document collection process in WMSs is based on (public) search engines and *Automatic Query Generation* (AQG), see, e.g., [20]. Instead of developing a special crawling component for the WMS, the idea is to send queries to existing large-scale search engines, thus enabling quick access to documents that are relevant to the current web mining task. The input to an AQG-based system is a manually determined set of seed tokens, e.g. based on an initial set of relevant documents. With the help of these tokens, different search queries are constructed and sent to a search engine. The returned documents (or typically a defined fraction thereof) are downloaded and analyzed, allowing more tokens to be extracted which are added to the *seed* tokens. This procedure is repeated until a defined stop criterion is met. Whether or not a document is relevant with respect to the web mining goals can again be determined with the help of a validation algorithm.

While AQG-based methods allow for the quick retrieval of documents, the relevance function used by the search engine for document ranking is usually unknown. Thus, high recall values are often not possible in cases where analysis is limited to a set of top-ranking documents returned by a search engine. Slight recall improvements can theoretically be achieved by exploiting special features of modern search engines such as the ability to search in particular areas of target websites. Such complex queries, however, create an enormous load on the search engines and many search service providers therefore limit the number of queries that can be executed, which in turn hampers the practical implementation of such approaches.

3.2 Basic ideas of xCrawl

The xCrawl document collection method is based on the assumption that websites are organized in a way that allows information to be easily accessed by users, for example *navigational structures* (index pages) such as hierarchies, site maps, or lists.

The idea of xCrawl is therefore to try to identify these index pages (called *hubs* [23]) as they hopefully contain many links leading to the actual relevant pages (called *authorities*) on a website.

In the context of our problem, a website (graph) therefore consists of a set of authorities, a set of hubs as well as some irrelevant pages. Figure 4 shows an example of such a web graph. Nodes correspond to web pages, while edges represent the links between the pages. The goal is to automatically discriminate between the hubs and authorities in order to direct a focused crawler in the right direction, i.e. towards the hubs, as they most probably contain links to further authorities.

Technically, this problem can be mapped to finding a subgraph $G' = (V', E')$ of the web graph $G = (V, E)$, which is bipartite, i.e. in which the nodes of the set V' can be partitioned into two disjoint sets V'_1 (authorities) and V'_2 (hubs) such that edges $e \in E'$ only connect nodes from V'_1 with nodes of V'_2 and vice versa.

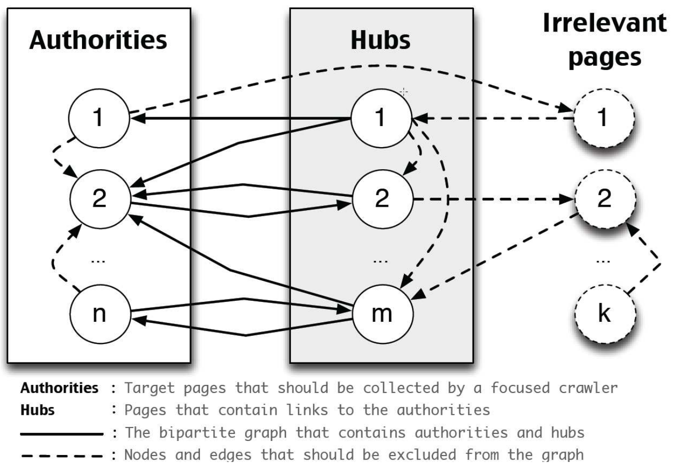

Authorities
Hubs
Irrelevant pages
1
2
...
n
1
2
...
m
1
2
...
k
Authorities : Target pages that should be collected by a focused crawler
Hubs : Pages that contain links to the authorities
: The bipartite graph that contains authorities and hubs
: Nodes and edges that should be excluded from the graph

Fig. 4. Example of the web graph partition.

Unfortunately, such partitioning within many websites is impossible because the authorities or hubs are interlinked, as shown in Figure 4. Consider, for instance the popular digital camera review site `www.dpreview.com`. On this site, reviews are grouped by the product announcement year in a “Timeline” section. Each page of this section corresponds to a year and contains links pointing to camera specifications announced during this year, i.e. it functions as a hub within our problem. When a crawler finds such a hub, it will – by following all the outgoing links of the page – easily find all camera specifications thus lifting the recall value almost to 1. However, as each such page also contains links to the index pages of previous years (i.e. to other hubs), the property of bipartite graphs is violated. Therefore, the partitioning algorithm must detect such links and remove or ignore them; the same holds for links between authorities In Figure 4, the nodes and edges which should be removed or ignored are depicted using dashed lines. In the AllRight system, the HITS algorithm [23] is used in combination with the PageRank metric[6] to compute a measure that guides the search towards new hubs (see the next section for more details).

The problem of where to start the crawling process within xCrawl is addressed by the automatic identification of an initial set of authorities. This step is based on *Automatic Query Generation*; the set of keywords to be used can be automatically determined by means of TF-IDF (Term Frequency – Inverse Document Frequency) weights of terms occurring in sample documents. Alternatively, such a set of keywords can be manually provided by the user.

Based on this initial set of authorities, xCrawl explores the web by following their outgoing links. During the crawling process, the link weights (that guide the search) are continually updated in order to guide the crawler to hub pages. In order to prevent the system getting lost deep within a web graph, xCrawl implements a "Random Restart" strategy, resetting the crawling process with a given probability.

3.3 xCrawl *algorithm details*

The complete xCrawl algorithm is sketched in pseudo-code in Figure 5. In addition to the set of example documents (target instances) for a domain, the algorithm accepts two parameters for fine-tuning the search behavior (described later in this section). The algorithm can also be parameterized with a *validator* component that is capable of determining whether or not a retrieved page is relevant. Thus, in comparison to other algorithms, the algorithm does not depend on specific outcomes of the validation component. The algorithm works as follows.

Tokenization. First, a token vector with TF-IDF weights is extracted from the sample documents (GetTokens). Standard techniques such as stop-word removal are applied. In principle, such a token vector can also be manually provided by the user, in which case all tokens have an initial weight of "1".

AQG-phase. Next, an initial set of authorities is retrieved by constructing queries based on the initial token set and sending them to a public search engine (GetAuthoritiesAQG). In order to determine the importance of individual tokens not only in the context of the sample documents but also on a more general scale, xCrawl first calculates a *search weight* for each token by sending singleton queries to the search engine and counting the results. The search weight corresponds to the $[0, 1]$-normalized number of hits returned by the engine: values close to 1 correspond to very general terms, values close to 0 are very specific. The harmonic mean is then used to combine the two weight values (TF-IDF weight and *search weight)* into a single measure of term importance. The harmonic mean was chosen as a metric in order to avoid a bias toward one of the two weights, which could be introduced by using, e.g., the average of both weights.

The query generation algorithm implemented in the GetAuthoritiesAQG procedure starts from the middle of the weight-ordered list and proceeds in

```
function xCrawl returns a collection of retrieved documents
  inputs: Documents, a set of WMS target documents
      restartProbability, (number) a restart threshold of the crawler
      Validator, a component that determines if a page describes
          an instance of the WMS domain
      maxIterations, (number) a convergency threshold of the crawler
  local variables (initially empty):
      WebGraph, an object that contains the graph for the crawled part
          of the Web
      Hubs, a collection of pages that are linked with many authorities
      Authorities, a collection of WMS target documents
  Tokens := GetTokens( Documents );
  Authorities := GetAuthoritiesAQG( Tokens, Validator, maxIterations )
  WebGraph := WebGraph ∪ Authorities
  Sites := GetWebSites( Authorities );
  for each ( site ∈ Sites ) {
    do {
      Page := SelectRandom( Authorities, site )
      repeat {
        Weights := CalculateWeights( WebGraph, Authorities, Hubs );
        Link := SelectLink( Page, WebGraph, Weights );
        Page := GetPage( Link );
        WebGraph := WebGraph ∪ Page;
        Authorities := Authorities ∪ Validator.AnalyzePage( Page );
      } until (Restart ( restartProbability ))
      Hubs := Hubs ∪ FindHubs( WebGraph, Authorities, Hubs )
    } while (noNewHubsFound( Hubs, maxIterations ))
  }
  for each ( hub ∈ Hubs ) {
    Authorities := Authorities ∪ DownloadAllAuthorities( Validator, hub )
  }
  return Authorities
```

Fig. 5. Pseudo-code of the xCrawl algorithm

both directions simultaneously, thus avoiding the inclusion of very specific or very general terms in the queries. All possible combinations of the tokens are used to construct the queries; for practical reasons, however, the size of the queries is limited depending on the restrictions of the search engine used.

Each generated query is sent to the search engine and the first n result entries are downloaded. The downloaded documents are then checked by the validator component. If they are recognized as target documents they are added – along with their outgoing links – to the web graph. In order to retrieve more authorities from sites on which authorities were already found, in each iteration the base website URLs of the authorities are determined and the queries are re-executed. This time, however, the search scope is limited to these base URLs, which can be done by adding the "site:" constraint to the query string.

Note that we did not exploit potentially existing local search facilities of the target sites. The reason is that we only considered sites on which at least one authority was found by the public search engine. If however one authority exists that was indexed by the search engine, then also the other authorities will be indexed. Thus, using the public search engine is sufficient for our purposes. Web sites, whose product databases are not accessible for public search engines, were therefore not considered in the AQG phase.

The AQG phase continues until no new authorities are found within a maximum number of iterations as defined by the *maxIterations* input parameter.

Finally, the websites hosting the authorities are extracted by GetWebSites. Note that in some scenarios it could in theory be advantageous not to start the subsequent crawling phase at the root ("/") of a site but at some subdirectory for which it is supposed that it is shared by all authorities, e.g. "/home/cameras". The GetWebSites method currently does not implement heuristics to detect such directory structures. However, due to the nature of the HITS algorithm used in the next phase, the crawler will be implicitly guided to the relevant parts of the web site very quickly.

Crawling phase. The For-loop implements the main crawling procedure of xCrawl and includes the two main tasks, *focused website traversal* and *hub identification* (implemented in the FindHubs method).

For each of the sites found in the AQG-phase, xCrawl repeatedly selects one of its known authorities at random (SelectRandom) and starts exploring the web graph. In order to prevent the system from getting lost deep in irrelevant branches of the graph, the algorithm occasionally restarts from another randomly-selected authority with some pre-defined probability, i.e. when the method Restart(probability) returns *true*.

The traversal of the web graph is guided by *link weights* which are calculated in the CalculateWeights method: Given a link l pointing to a page v, the link weight of l is computed as

$$linkWeight(l) = weight(v)/outdegree(v) \tag{1}$$

where $weight(v)$ is a weight of a node v and $outdegree(v)$ is the number of outgoing links of v. In the xCrawl implementation, the node weight $weight(v)$ corresponds to the average of three different weights: PageRank [6], as well as authority weight and hub weight (details of their calculation are given later in this section). Non-hub pages – as identified by the FindHubs method –

initially receive a small weight of $1/n$, where n is the number of pages in the web graph. Hub nodes are assigned the average of $1/n$ and their original hub weight.

The weights are then propagated over the web graph using the PageRank algorithm. In the initialization phase, the highest weight of "1" is assigned to all known authorities, i.e. to the pages that were accepted by the validator. Later on, the weight of a page u with respect to a set of links L_u pointing to u and a damping factor c is defined as:

$$weight(u) = c \sum_{l \in L_u} linkWeight(l)$$

The SELECTLINK method simply returns the outgoing link (URL) of the current page with the highest weight. The page behind this URL is downloaded (GETPAGE) and added to the web graph together with all its outgoing links. If the newly added page contains information about a desired target instance (determined through function $Validator$.ANALYZEPAGE), it is added to the list of known authorities.

Hub identification / hub weights. During the expansion of the web graph, xCRAWL tries to identify additional hubs which, as described above, are likely to guide the crawler to further authorities. The hub identification method FINDHUBS is executed whenever a restart event disrupts the web traversal process.

Technically, the method implements the HITS algorithm for the identification of the bipartite cores in the web graph and calculates a so-called hub weight and an authority weight for each node of the graph. The main assumption in our context is that a "good" authority in our context is linked with "good" hubs and a "good" hub points to many authorities, thus mutually reinforcing one another. Thus, if a web page is linked with a page which has already been identified as an authority, it receives a higher weight as it is most probably a part of a navigational structure (e.g. an index page).

The HITS algorithm in our implementation starts by assigning an initial weight of $1/n$ to every node, where n is the number of nodes in the graph. Upon each subsequent execution of the algorithm, the weights are updated using the hub weight (H) and authority weight (A) operators, which are defined for a node u and a set of nodes L_u that contain links pointing to u as follows:

$$H : hubWeight(u) = \sum_{v \in L_u} authorityWeight(v)$$

$$A : authorityWeight(u) = \sum_{v \in L_u} hubWeight(v)$$

Nodes that are identified as authorities by the validator in the crawling cycle are ultimately assigned an authority weight of 1 and a hub weight of 0. Hub nodes (i.e. nodes with a non-zero hub weight) that have been found by FindHubs maintain their existing hub weights; their authority weights are, however, set to 0. After each iteration, the obtained weights are normalized and the algorithm continues the reinforcement of node weights until a predefined number of iterations has elapsed or no weights are changed during an iteration. Finally, the hub weights are stored in the web graph. The actual authority weights produced by the HITS algorithm are irrelevant in our context and are overwritten during the execution of the validation component.

The crawling process stops when it is unable to identify any new hubs for a website after a predefined number of iterations (NoNewHubsFound).

Authority harvesting. The focused crawling phase also helps the system to identify additional important hubs in the web graph. In the final step, xCrawl analyzes the Document Object Model (DOM) of links pointing to the known authorities and retrieves their DOM parent. Based on this DOM navigation, the algorithm is capable of downloading and validating all the pages published on the same website and which are referenced from the identified subtree of each hub (DownloadAllAuthorities). Newly identified authorities are added to the list of known authorities and finally the list of authorities (containing target instance information) is returned as the result of xCrawl.

3.4 Evaluation of crawling performance

3.4.1 Experimental setup

In order to evaluate our new crawling technique and compare it with existing approaches, we measured its performance in two typical web mining scenarios. In one scenario the goal was to extract information from highly-structured web sources, which is the standard situation in our ontology instantiation problem domain. The specific test domains were *digital cameras* and *laptop computers*. This scenario utilized the "standard" table recognition validator of the AllRight system; see [26,18] for more details.

In the second scenario, the goal was to find *text reviews* for certain products written in natural language. This scenario was chosen in order to evaluate how xCrawl performs with different validators and to demonstrate that our method is generally not limited to a certain type of (domain-specific) validator. The application domains of the second scenario were *MP3 players* and *cell phones*. Here, we used a Bayesian Network (BN) classifier implemented in the WEKA framework [38] which was trained to recognize *text reviews* of the products.

In order to make our approach comparable with existing methods for finding plain-text documents, we did not use a predefined user ontology as described in Section 2 in the context of this second scenario but rather provided twenty valid sample documents for each domain as seed data for the crawler. These seed documents were required to train the baseline approach BN classifier that was employed before the crawling process. The resulting relevance score distributions were strongly bimodal and similar to those reported in [7]; the training documents for the were manually chosen in a way that no bias was introduced to the BN classifier by taking sample documents from the six web sites chosen for the recall measurements later on.

The table recognition algorithm used for the first scenario does not need any training samples but rather uses a number of heuristics to locate and extract a table from a page and relies on the correct rendering of the page by the crawler's web browser. Consequently, a modern browser that can correctly render any web page was used in the experiments. More details of the table recognition algorithm are given in Section 5.1 and in [18]; a deeper discussion of this technique is beyond the scope of this paper.

As a baseline for our comparative evaluation, we chose a crawler that uses a combination of two BN classifiers as proposed by Chakrabarti et al. [7]. Note that the "Context Focused Crawler" (CFC) proposed in [12] is very similar our approach as it takes the website structure into account to guide the search process. However, in our experiments, CFC was unable to construct the required "context graphs" because the used search engines (Google and Yahoo!) did not return a sufficient number of pages that were linked to the initial authorities. Google's search engine, for instance, did not return any of the (at least) three pages published on `www.dpreview.com` that contained a link to the *Canon A650is* page, which was identified by the AQG method as an authority. Therefore, we compare our results only to those obtained using the BN-based crawling technique from [7].

In order to be able to measure the *recall* value for each crawler and to provide equal test conditions, we limited the crawlers to six predefined websites. These websites were determined by running the AQG phase of xCrawl once and picking the top six entries. Note that the baseline BN crawler is not able to identify such webpages efficiently. Instead, such a BN crawler could a use directory such as `dmoz.org` as a starting point. If, however, an important site is not listed, it will not be considered. The website `imaging-resource.com` for example contains more than 790 specifications for digital cameras but is not listed in the corresponding `dmoz.org` category [5]. Still, as this site is important, it is highly ranked by Google and will most probably be found by xCrawl's AQG component.

3.4.2 Measurement methods and results.

To measure achievable recall values, we manually determined the number of target instances on the selected websites. On some sites, this information was published explicitly; if this was not the case, we estimated the number of instances by browsing the relevant website sections manually. Depending on the domain and the website, the number of instances varied between 100 and 900.

In order to determine whether or not our new crawling method is capable of finding more domain instances over a shorter period of time than existing approaches, we used a running time limit of six hours in all our experiments [6]. After executing both the baseline crawler and xCrawl in all domains, we measured the recall values by comparing the number of found instances with the number of actual ones. Detailed measurements for the individual websites are shown in Figures 6 to 9. In Figure 10, the overall results of each indidivual experiment are given and it can be seen that xCrawl performed significantly better across all domains, with an average recall improvement of 42%.

Website	Found by xCrawl	Actually existing	Found by baseline
reviews.cnet.com	643	670	6
imaging-resource.com	794	894	72
dcresource.com	372	432	175
parkcameras.com	179	191	181
steves-digicams.com	634	686	479
dpreview.com	825	845	625

Fig. 6. Digital camera domain (Tabular sources)

Furthermore, a manual analysis of the individual website results showed that the baseline crawler performed particularly poorly on websites that have many different subsections, i.e. sites on which a range of products can be found. The main problem of the BN-based crawler on these sites was that the pages of these subsections were very similar but contained different product types. There are, for example, many websites that contain information about both digital cameras and photo printers. Many printer manufacturers are, however,

[5] Home/Consumer_Information/Electronics/Photography/Digital_Cameras (Date of access: 20 January 2009).

[6] This value was determined by running xCrawl once for the digital camera domain and observing its convergence behavior.

Website	Found by xCrawl	Actually existing	Found by baseline
digitaltrends.com	217	255	46
www.mp3.com	134	150	82
www.pixmania.co.uk	195	227	112
www.pcmag.com	186	214	149
www.reviewcentre.com	183	196	195
reviews.cnet.com	648	713	350

Fig. 7. MP3 player domain (Text sources)

Website	Found by xCrawl	Actually existing	Found by baseline
dabs.com	289	302	57
newegg.com	493	507	58
datavis.com	96	108	84
europc.co.uk	86	89	88
ww2.inoax.com	612	626	142
tabletpc.alege.net	793	851	148

Fig. 8. Laptop computer domain (Tabular sources)

Website	Found by xCrawl	Actually existing	Found by baseline
laptopmag.com	82	86	46
www.mobiledia.com	132	132	75
mobile-phones-uk.org.uk	74	88	80
pcmag.com	365	389	153
reviews.cnet.com	503	554	158
www.mobile-review.com	524	524	345

Fig. 9. Cell phone domain (Text sources)

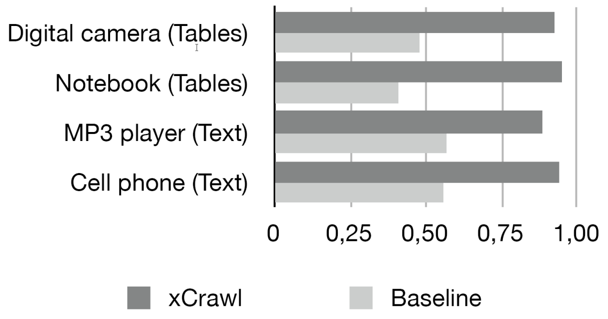


Fig. 10. Comparison of average recall

also producers of digital cameras. The apprentice classifiers of the BN approach often assign high probability scores to manufacturer names because of the manufacturer-oriented structures of many websites. Thus, if the BN crawler successfully retrieves some target instances in the beginning, it might also mislead the crawler (based on the importance of the manufacturer name) to the wrong sections of the website. Such a behavior is avoided in xCrawl because of its “restart” strategy and the “bottom-up” crawling process [7].

Note that we do not report detailed numbers on crawling *precision* for the individual experiments, because the achievable precision values strongly depend on the validation component used by the crawler; therefore, these numbers are no indicators for the performance of the xCrawl method, which – by design – can work with different validator components. In general, for all tested domains both validation methods (the BN-based classifier and the table recognition method) were capable of differentiating relevant and irrelevant documents with high precision as reported for the two-classifier approach already in [7]. More precision results for the table recognition method are reported in [19].

3.5 Comparison with previous approaches

Previous systems that use a focused crawler for web mining problems include [1,7,12] and [31]. Most systems follow a learning-based approach to guide the crawling process. The method presented in [7], for instance, uses two classifiers for learning: a primary classifier that decides on the relevance of documents and an apprentice classifier that predicts the relevance of unseen pages. Basically, the apprentice acts as a simple reinforcement learner that is trained using features derived from the Document Object Model (DOM) of the retrieved documents and the output of the main classifier. The crawler presented in [31] also relies on reinforcement learning. In contrast to [7], however, this approach analyzes the text fragments in the neighborhood of outgoing links or within the title and header tags with the goal of incrementally learning whether or not a link is likely to lead to a target page and should be followed by the crawler.

The main difference in the xCrawl method is that it does not rely on a specific validation technique or special types of feedback from the validator component (such as certain probability estimates). Thus, different validation algorithms can be used depending on specifics of the application domain and the mining goal. In addition, links and their environments within modern web applications are often created through scripts and used in menus or sidebars. Exploiting only the DOM as well as the surrounding text may thus be insufficient and not provide an effective method of crawling. xCrawl can cope with such dynamically generated links by using a browsing client which also executes scripts. This issue is discussed in more detail in [34].

In the approach proposed by Diligenti et al. [12] a so called *Context Focused Crawler* (CFC) is used that discovers typical link structures (Context Graphs)

[7] For details of the parametrization of xCrawl (e.g. finding a suitable value for the restart probability), see [34].

in which relevant documents appear. Existing search engines are exploited to implement a "reverse crawling" strategy in order to calculate expected link distances between a target document and other documents. A set of classifiers is trained to assign categories to documents according to their links. In a similar fashion to the AQG step of our xCrawl method, the Content-Focused Crawler of [12] relies on public search engines. Their method however places a significant load on the underlying search engines in the training phase, in particular when the crawled websites are not homogeneous and include a variety of sections with different content. In comparison, the number of requests forwarded to the search engine in the initial phase of xCrawl is significantly lower, allowing the approach to also be used when the number of search engine queries is more restricted. In addition, the AllRight system may be manually pre-configured with (additional) links to target pages within the domain, if for some reason a website cannot be indexed by the search engine and would otherwise remain undetected.

Another query-based system is *QXTract* [2], which extracts features from relevant documents based on user-defined tuples. In addition, the system implements a hybrid technique combining queries that are generated with the Okapi query expansion method and two text classification methods, *Ripper* and *SVM*. The method handles the advantages and disadvantages of the individual query-based approaches, such as the fast retrieval of target documents but comparably low recall values [20]. Our method, on the other hand, overcomes these limitations through its mix of query-based retrieval, i.e. searching, and focused crawling.

Some crawling systems do not rely on search engines for locating (additional) websites on which their targets are published as they assume either that the pages of one relevant website include links to other websites from the same domain or that directories such as `dmoz.org` contain links to other target websites. [14], for instance, makes this assumption and the crawling technique described is actually a combination of two crawlers: one to discover relevant websites and another to crawl them. When the assumption of interlinked websites holds in a domain, the combination of two crawlers appears to be efficient in terms of *coverage*. The running times for crawling are, however, rather long when compared with AQG-based approaches [20].

When directories are used as a starting point, crawlers may face the problem that these directories are incomplete since they are developed in a manual process by their users. Again, the mixture of searching and crawling in the xCrawl and the use of up-to-date search engine method helps to overcome this problem and leads to much better coverage whilst locating relevant websites more quickly.

Finally, instead of only exploiting navigational structures such as site maps or other index pages, one further approach to find and extract the information from the "Hidden Web" is to locate and use the search interfaces of the website. "Query probing" is the a central and critical task in such approaches and in many cases the query generation process is based on the usage of a lexical database such as WordNet, see [5]. While such a technique could in general be seen as a complementary technique for the AllRight system, the knowledge to be extracted in our target domain is often rather specific or technical. Such specific vocabulary is often not contained in general-purpose lexical databases and is also hard to learn in an automated way. The existence of such a vocabulary is however a mandatory prerequisite for achieving high recall values. The xCrawl method therefore follows an approach that exploits navigational structures of websites instead of search interfaces and does not rely on manually-engineered or learned domain vocabularies.

Having completed the discussion of the xCrawl algorithm, the next section describes how the AllRight ontology instantiation system further processes the downloaded document collection.

4 Instance name recognition and document clustering

Similar to other WMSs, the AllRight ontology instantiation system exploits the redundancy of the information on the web in particular to check the plausibility of the extracted facts or to combine different pieces of information. Examples of previous systems that follow this approach are statistics-based approaches such as KnowItAll [15], Web→KB [10], C-Pankow [8] or SemTag [13].

The main assumption of such approaches is that instance descriptions of a class, such as the specifications of a digital camera model, are published on different websites and that the information from different sites can be combined to improve the effectiveness of the mining process. The Web→KB [10] system, for example, uses a combination of three naive Bayes classifiers trained to group (cluster) the available documents based on full text, headers/titles and hyperlinks (anchor tags). An additional technique is implemented in the KnowItAll [15] system, which uses web statistics acquired from search engines in order to determine the plausibility of extracted facts.

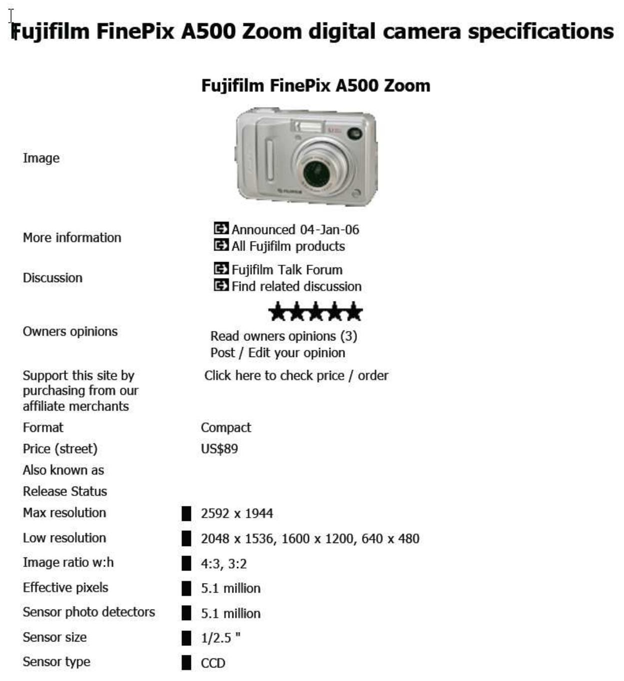
Note that in the context of the domain of electronic consumer goods such as digital cameras, we face the additional problem that webpages that describe the target instances can differ from each other in the following dimensions. Consider, for example, Figures 11 and 12:

- *Presentation.* The two tables are different with respect to the way the information is structured. Some distinct product attributes are for instance merged into a single table cell in one description while the other places them in separate cells.

- *Detail coverage.* Product descriptions can be different with respect to the level of technical detail. On manufacturer homepages, the information about memory card capacities is often not given. Such information can, however, be found on many consumer review sites.
- *Content.* Not only can the attribute *names* vary (e.g., “Total pixels” and “Sensor photo detectors” in the examples) but there may also be a mismatch between the values of attributes that describe the same property of a camera.

Most existing approaches for document clustering such as the ones mentioned above are designed to work with web sources written in natural language. If documents are however given in tabular form, existing document clustering methods do not work well because tables that describe *different* instances typically contain very common sets of words (like attribute names) and almost no grammar. As can be seen in the examples above, it is on the other hand also possible that there are documents that describe the same instance but use different sets of tokens (attribute identifiers). Again, this can lead to imprecise clustering results.

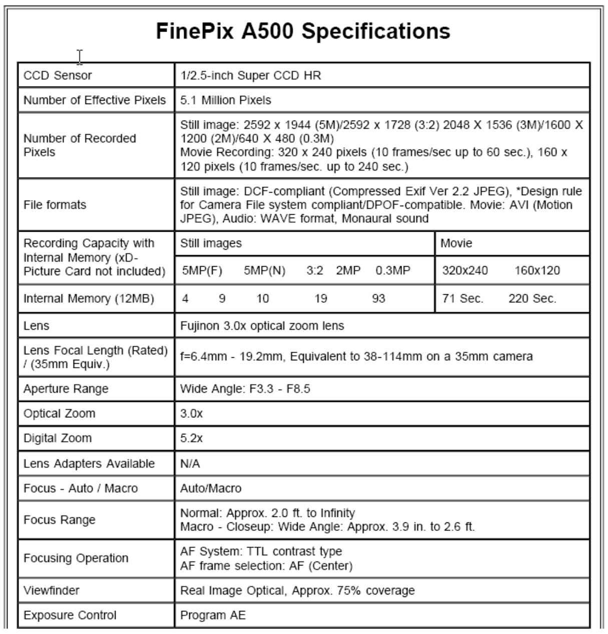
Since standard clustering techniques do not work well in the described domains, a different and more indirect approach was developed in the AllRight system, based on the extraction of unambiguous instance identifiers for the documents that were retrieved in the crawling phase. The idea is that once an



Fujifilm FinePix A500 Zoom digital camera specifications

	Fujifilm FinePix A500 Zoom
Image	
More information	Announced 04-Jan-06 All Fujifilm products
Discussion	Fujifilm Talk Forum Find related discussion
Owners opinions	★★★★★ Read owners opinions (3) Post / Edit your opinion
Support this site by purchasing from our affiliate merchants	Click here to check price / order
Format	Compact
Price (street)	US$89
Also known as	
Release Status	
Max resolution	2592 x 1944
Low resolution	2048 x 1536, 1600 x 1200, 640 x 480
Image ratio w:h	4:3, 3:2
Effective pixels	5.1 million
Sensor photo detectors	5.1 million
Sensor size	1/2.5 "
Sensor type	CCD

Fig. 11. Example camera specification



FinePix A500 Specifications

CCD Sensor	1/2.5-inch Super CCD HR						
Number of Effective Pixels	5.1 Million Pixels						
Number of Recorded Pixels	Still image: 2592 x 1944 (5M)/2592 x 1728 (3:2) 2048 X 1536 (3M)/1600 X 1200 (2M)/640 X 480 (0.3M) Movie Recording: 320 x 240 pixels (10 frames/sec up to 60 sec.), 160 x 120 pixels (10 frames/sec. up to 240 sec.)						
File formats	Still image: DCF-compliant (Compressed Exif Ver 2.2 JPEG), *Design rule for Camera File system compliant/DPOF-compatible. Movie: AVI (Motion JPEG), Audio: WAVE format, Monaural sound						
Recording Capacity with Internal Memory (xD-Picture Card not included)	Still images					Movie	
	5MP(F)	5MP(N)	3:2	2MP	0.3MP	320x240	160x120
Internal Memory (12MB)	4	9	10	19	93	71 Sec.	220 Sec.
Lens	Fujinon 3.0x optical zoom lens						
Lens Focal Length (Rated) / (35mm Equiv.)	f=6.4mm - 19.2mm, Equivalent to 38-114mm on a 35mm camera						
Aperture Range	Wide Angle: F3.3 - F8.5						
Optical Zoom	3.0x						
Digital Zoom	5.2x						
Lens Adapters Available	N/A						
Focus - Auto / Macro	Auto/Macro						
Focus Range	Normal: Approx. 2.0 ft. to Infinity Macro - Closeup: Wide Angle: Approx. 3.9 in. to 2.6 ft.						
Focusing Operation	AF System: TTL contrast type AF frame selection: AF (Center)						
Viewfinder	Real Image Optical, Approx. 75% coverage						
Exposure Control	Program AE						

Fig. 12. Different specification of same camera

instance name (such as *Canon A650is*) can be determined, it will be relatively easy to retrieve additional documents from the web to increase information redundancy. Unfortunately, such names or identifiers are usually not *explicitly* given as key-value pairs in document specifications. Also, there is no general rule where such identifiers should appear on a webpage. The AllRight approach therefore aims to exploit structural and meta-information of target webpages and, for example, to use the information contained in the `<title>` tag as well as *hyperlinks* that point to that page.

4.1 Name recognition example

For humans, it is relatively easy to see that the documents shown in Figure 11 and Figure 12 contain specifications for the same camera model, i.e., the Fujifilm FinePix A500. However, automated clustering techniques which are, for example, based on document similarity and TF-IDF weights, will most likely place them in different clusters. Instead standard techniques will rather put all the camera specifications of one website into the same cluster because the pages of one website are very similar in terms of the terms or tokens they use to describe camera features.

As mentioned above, the AllRight system therefore relies on an indirect approach clustering documents according to their instance names, determined by exploiting hyperlinks and other document meta-information. Note that while technical features of electronic consumer goods are often organized in tables, the names of the manufacturer and the model name are often placed somewhere else on the page. On review sites in particular, the model information is encoded in the page title or in the name of the (physical) HTML page and, correspondingly, in the links pointing to this page. When trying to extract the information from these sources, the problem remains that in many cases these sources also contain additional irrelevant information. Typical examples include document titles such as "ZDNet's Canon EOS Digital Rebel XT with 18mm-to-55mm lens (silver) Specifications & Features" or "Steves Digicams - Fujifilm FinePix A500 - User Review".

Table 1 shows an example of the strings that were found in the title tag of a page and in its incoming links. In order to extract the most plausible instance identifier from these strings, AllRight's name recognition and clustering technique first splits the observed strings into tokens. [8] We use a representation in which each occurrence of a token in an observation string of a website is a positive *token observation*. For the clustering technique, each observation is associated with a weight factor that depends on the observation string and thus helps us to better discriminate between different observations of a token. These weights are particularly important in our approach as the number of observations is rather small. The table resulting from this transformation step (Table 2) is subsequently used as an *observation table*, i.e. as an input for the clustering algorithm.

Title	Steves Digicams - Fujifilm FinePix A500 - User Review
Link #1	Fujifilm Finepix A500
Link #2	[A500 review]
Link #3	FinePix A500
Link #4	FinePix A500
Link #5	FinePix A500
Link #6	Fujifilm FinePix A500
Link #7	FinePix A500 5.1 megapixel camera

Table 1
Observations from a product description web page

Reconsidering the example in Table 1, we can observe that some tokens appear in several observation strings. The *observation table* as shown in Table 2 is constructed as follows. First, standard preparatory steps such as the removal of stop-words or punctuation symbols are executed. Then the table layout is determined: in the example, the table has 8 columns, which are – according to statistics terminology– called *variables*. For each of the 10 tokens appearing in the observation strings, a row (called observation vector or *profile*) is added to the table.

The token weights in Table 2 are computed as the ratio of the overall tokens to the number of tokens found in an observation vector. Consider, for example,

[8] The tokenized lists are also called *feature vectors* [24].

the observation “A500” in *Link #2*. The overall number of tokens is 10 and the number of tokens found in this observation is 2 (as also token “Review” has been found in this link). The weighted value for this observation is – according to our scheme – therefore 5. In principle, other weighting schemes are possible for discriminating between observations. Our experiments have however shown that this particular metric worked sufficiently well within the domains we considered.

	Title	Link♯1	Link♯2	Link♯3	Link♯4	Link♯5	Link♯6	Link♯7
Steves	1.43	0	0	0	0	0	0	0
Digicams	1.43	0	0	0	0	0	0	0
User	1.43	0	0	0	0	0	0	0
Review	1.43	0	5	0	0	0	0	0
Fujifilm	1.43	3.33	0	0	0	0	3.33	0
FinePix	1.43	3.33	0	5	5	5	3.33	2
A500	1.43	3.33	5	5	5	5	3.33	2
5.1	0	0	0	0	0	0	0	2
megapixel	0	0	0	0	0	0	0	2
camera	0	0	0	0	0	0	0	2

Table 2
Observations table with possible clusters

Looking at the resulting weighted profiles of “Fujifilm”, “FinePix”, and “A500”, we can observe that they differ strongly from those of the others. The clustering algorithm we subsequently describe however (correctly) places them in one cluster and thus separates them from the others. Technically, clustering works by grouping the tokens in a way that minimizes some predefined measure (the Euclidian distance in our case) for each group. The names of these groups are generated from the union of their tokens. In our example, the cluster name (corresponding to the instance name) would be “Fujifilm FinePix A500”. This procedure is applied to all downloaded documents which can subsequently be grouped by their instance names.

In this particular scenario, we only used information contained in the title and in the anchor tags. In principle, other tags could also be considered. An additional validation step could check whether the identified instance name appears somewhere else in the current or referring document.

In the next section, we will describe the algorithm formally before reporting on the accuracy of the name recognition and document clustering phase.

The input to AllRight's clustering phase is the set of *candidates*, i.e. the set of *annotated* web documents retrieved by the crawler component, and which hopefully contain the desired instance information. The annotations include the contents of the tags that are exploited for clustering as described in the example above. The final goal is to group these documents into clusters which most probably describe the same instance.

AllRight's clustering technique has several phases. First, an identifier for each candidate is determined by analyzing the observed tokens as described in the example. Next, the documents are grouped based on their names. Finally, the clusters are validated and fine-tuned by evaluating the results of queries to public search engines. Figure 13 shows the algorithms pseudo-code.

```
Clustering ( in:candidates out:clusters)
  input: candidates, a set of annotated documents
  output: clusters, a set of clusters of documents (describing the same instance)
{ /* Main clustering phase */
  Prepare( candidates )
  for each (candidate ∈ candidates)
  {
    nameClusters := X-Means( candidate )
    cluster := Select( nameClusters )
    candidate.identifier := Reconstruct-Order( cluster )
  }
  clusters := Partition( candidates )
  while (existIncoherentClusters( clusters ) )
  {
    cluster := removeMostIncoherentCluster( clusters )
    removeCoreComponent(cluster)
    improvedClusters := Partition( cluster )
    clusters := clusters ∪ improvedClusters
  }
  /* Validation and improvement phase */
  Validate-and-Improve( clusters )
  return candidateClusters
}
```

Fig. 13. AllRight Clustering algorithm

Identifier recognition. The name recognition algorithm starts with a preparatory step (method Prepare), in which the data is pre-processed, removing punctuation, stop words etc.

Next, in the first For-Loop of the algorithm, for each candidate (document), the most promising *identifier* is determined by building clusters of observed tokens as described in the example. Technically, we use the X-Means method [28] for this purpose. Clustering algorithms can in general be applied to either create hierarchies or partitions of objects. In our case, we need to partition a set of tokens into groups of highly coherent tokens. The *K-Means* algorithm is one of the most popular partitioning algorithms in this context, both due its simplicity and its tendency to converge to a local extremum very quickly in most cases. The simplest K-Means algorithm can be summarized as follows [22]:

(1) Generate an initial partition of k clusters. Usually, some k random points of a data set are selected as cluster centers (centroids). Note that a bad initial partition can greatly influence the result.
(2) Associate each point of the data set with the closest centroid.
(3) Revise the centroids of the created groups until the centroid values stabilize.

While the algorithm is simple, it has two drawbacks: poor computational scalability and the need for a predefined number of clusters k. The newer *X-Means* algorithm improves the speed by "centroid blacklisting" and by storing sufficient information in the nodes of a "KD-Tree". In our algorithm, the X-Means method takes a range of possible cluster numbers and iteratively applies the improved K-Means method to the data set starting from the lower bound until the upper bound is reached. In each iteration, it analyzes the results of K-Means to determine a better set of initial centroids. Finally the algorithm evaluates each partition set created by K-Means and returns the one with the best score as the result (*nameClusters*) along with the corresponding number of clusters k.

The subsequent selection (method Select) of the most promising cluster for the determination of the identifier is done by the analysis of unused data like the contents of other tags that were not examined by X-Means, such as the captions of tables that can also contain the identifier. The selection method first computes weights for each of the clusters. The weights are calculated as the ratio of the number of documents in which *all* tokens of an examined cluster can be found (tags already used for X-Means analysis are ignored) to the number of documents in the complete data set. Then, the algorithm calculates a centroid for all cluster scores and returns the cluster nearest to the centroid in the terms of Euclidian distance.

Since the resulting cluster only consists of tokens, all ordering information is lost. In order to restore a valid identifier the (method Reconstruct-Order) method uses X-Means input data to correctly reorder the tokens: simple string comparison is used to determine which place each token occupies in the source strings. The most frequent order is used to create the candidate identifier which is stored with the candidate document.

Document clustering. After each candidate is associated with an identifier, a partitioning algorithm (Partition) is applied to produce clusters of candidates. The algorithm uses the *string metric* [36], which is defined on the interval $[0, 1]$ to measure the similarity of candidate identifiers. The partition algorithm sequentially examines all candidates and creates groups of similar (measure = 1) documents. If the algorithm detects that one group is a subset of another group, the first group will be removed.

The clusters are then checked for coherency (While-Loop in the algorithm), i.e. that all elements in a cluster are similar to each other. If a cluster is incoherent, a core candidate is removed (Remove-Core-Component) and the remaining candidates are reanalyzed by the partitioning algorithm. We define a candidate as a *core candidate* if its identifier is similar (measure = 1) to all identifiers in a cluster that, in turn, are "non-similar" with each other (measure < 1).

Consider the incoherent cluster that consists of 5 candidate identifiers in Table 3. Identifier 1 is similar to identifiers 2-5 (cases 1-4). Identifiers 2 and 3, however, are not similar to identifiers 4 and 5 (cases 6, 7, 9 and 10), which are similar to each other (cases 5 and 8). Therefore, identifier 1 is a core candidate that links the two non-similar subgroups of identifiers 2, 3 and 4, 5 respectively. If a core candidate is removed from a cluster, the application of the partitioning algorithm to the remaining candidate identifiers will split the cluster into two new coherent clusters.

Case	Id	Name	Id	Name	Similarity
1	1	Canon Powershot	2	Canon Powershot A630	1
2	1	Canon Powershot	3	Canon Powershot A630 silver	1
3	1	Canon Powershot	4	Canon Powershot A70	1
4	1	Canon Powershot	5	Canon Powershot A70 Review	1
5	3	Canon Powershot A630 silver	2	Canon Powershot A630	1
6	4	Canon Powershot A70	2	Canon Powershot A630	0.94
7	4	Canon Powershot A70	3	Canon Powershot A630 silver	0.94
8	4	Canon Powershot A70	5	Canon Powershot A70 Review	1
9	5	Canon Powershot A70 Review	2	Canon Powershot A630	0.91
10	5	Canon Powershot A70 Review	3	Canon Powershot A630 silver	0.79

Table 3
Example of comparison of different names with string metric.

Query-based validation and improvement of clusters. The described clustering scheme based on the pure *string metric* has some practical limitations. On the one hand, if two identifiers are not completely identical (and there is no substring relationship), they will be considered different. The name "Olympus Camedia C-700 Ultra Zoom", for instance, can also be found as

“Olympus Camedia C-700 UZ” on some pages. Although both names refer to the same camera model, the algorithm will place the documents in different groups.

On the other hand, since the string metric is based on the maximum length of identical substrings, it evaluates a substring of a string as similar to the later one. Hence, names like “Canon EOS 1D Mark II N” and “Canon EOS 1D Mark II” will be put in one group although they represent two different products. In order to improve the *recall* measure, a group improvement algorithm (VALIDATE-AND-IMPROVE) is subsequently used to identify groups that should be merged or split.

1	“Canon EOS-1D Mark II” -“Canon EOS-1D Mark II N”	38	$1st\_string\_query$
2	“Canon EOS-1D Mark II N”	35	$2nd\_string\_query$
3	“Canon EOS-1D Mark II”	73	$common\_string\_query$
4	”Olympus Camedia C-700” +”Ultra Zoom” -UZ	980	$1st\_string\_query$
5	”Olympus Camedia C-700” -”Ultra Zoom” +UZ	286	$2nd\_string\_query$
6	”Olympus Camedia C-700”	1734	$common\_string\_query$

Table 4
Examples of queries written using Google syntax and number of results returned.

Technically, the refinement step is based on checking – through additional search engine queries – how often an identifier (group name) can be found together on the web. Thus, we extract the common parts of the names and construct search queries for them. The algorithm creates three queries for each combination of names (see for example queries 1 to 3 in Table 4) and calculates the number of pages on which both strings were found by combining the partial results as follows:

$$\begin{aligned} &difference = \\ &\quad count\_common\_string\_query - \\ &\quad (count\_1st\_string\_query + count\_2nd\_string\_query) \end{aligned}$$

Queries 1, 2, 4 and 5 in Table 4 are more restrictive queries which are used to identify the number of pages on which only one of the two strings was found. In order improve the clusters according to these query results, we define the *context similarity coefficient* as follows:

$$cs = \frac{difference}{count\_common\_string\_query}$$

For the pair of names “Canon EOS 1D Mark II” and “Canon EOS 1D Mark II N”, cs is 0 (see Table 4). This indicates that the two names correspond to different cameras. In the second case shown in Table 4, cs is 0.27, suggesting that the same camera model is meant by the two names.

Finally, to further improve the clusters, we calculate context similarity coefficients for all possible combinations of groups in order to test whether they can be merged or split. We use the so called "elbow rule" to detect the appropriate significance level by searching the maximum difference of the coefficient values between each pair of neighboring values in the ordered list of similarity coefficients. The groups are only modified if their context similarity coefficient is greater than the significance level.

4.3 Evaluation results

In the following, we report on selected results of the experiments we have conducted in different domains in order to evaluate the accuracy of the document clustering and instance name recognition techniques.

The application domains are again "digital cameras" and "laptop computers". In the digital camera domain, the tests were based on an automatically downloaded set of 3135 observations (candidate pages) from 18 websites; for the evaluation in the domain of laptop computers, 2930 observations from 12 websites were used. The digital camera observations contained 85 false positives, e.g. pages that describe photo printers but share similar attribute names with digital cameras, and the laptop observations 34, which were later removed in the clustering step.

The results of both clustering and fact validation approaches were evaluated using the standard information retrieval metrics of precision, recall and f-measure.

To evaluate the accuracy of the clustering approach, the inputs and outputs were analyzed by students who compared manually defined groups with automatically created ones. First, all correct results were manually identified from the input data, thus providing the number of *existing values*. Clusters were considered valid if they did not contain false positives and there were no other clusters that contained tabular descriptions of the same instance. The number of correctly created clusters defines the number of *correct found values*. The number of *found values* corresponds to the size of the output.

The evaluation initially applied the X-Means algorithm implemented in the WEKA framework [38] to the data sets using the TF-IDF measure, which corresponds to the technique used by existing approaches. The clustering approach extracted all tokens (terms) from the analyzed documents and calculated the components of each document vector. The clustering algorithm was then applied to the set of all vectors. The results, which are presented in Table 5 under X-Means (Direct), show that the algorithm tends to place documents from one website into one cluster. This tendency is particularly strong for websites whose pages are generated from a database. Almost the

Algorithm	Domain	Precision	Recall	F-Measure
X-Means (Direct)	Digital Camera	0,340	0,234	0,277
	Laptop	0,289	0,303	0,296
X-Means (Metadata)	Digital Camera	0,453	0,342	0,390
	Laptop	0,392	0,389	0,390
X-Means (Identifier)	Digital Camera	0,964	0,965	0,964
	Laptop	0,943	0,918	0,930
Association Rules	Digital Camera	0,935	0,974	0,954
	Laptop	0,913	0,934	0,923

Table 5
Evaluation results for different algorithms

same results were obtained when applying X-Means to a vector space created from metadata like anchor text and titles which were extracted by xCrawl, see X-Means (Metadata).

When using the AllRight clustering approach in the domain of digital cameras, the system created 498 groups, i.e. 498 camera models with multiple descriptions were identified. 234 groups were generated for the domain of laptop computers. The final results are presented in row X-Means (Identifier) of Table 5. As one can see, the F-Measure is significantly better compared to previous approaches. Note also that the value for the digital camera domain is higher than for laptops, since more observations could be exploited. In general, these measures will improve even more if more observations are available. These results are very promising as very precise partitions could be achieved although the clustering method used only a limited number of observations, which is a common situation in many web mining scenarios.

Finally, a variant of the suggested clustering method was evaluated in which the X-Means clustering technique was replaced with the *Apriori* association rule mining algorithm [3]. The latter is often used in text document clustering techniques to extract identifiers from text sources. The obtained results are comparable with the results of our new algorithm. However, in order to extract association rules efficiently, one has to set an appropriate *minimum support value*. This value in our case indicates the minimal acceptable fraction of metadata strings in which an token or a union of tokens can be identified. The value of the minimal support was set to 0.37 in our experiment and was calculated manually using a fraction of the candidate set. The proposed X-Means-based clustering technique, however, does not require such a training and calibration phase making it more suitable for real world applications.

5 Fact extraction & instance generation

In the last step of the ontology instantiation process, the goal is to use the information extracted from the retrieved documents to populate the ontology. This final phase is based on three techniques: automated table recognition, ontology-based text extraction and query-based validation and conflict resolution.

5.1 Visually-guided table extraction

Similar to the validation techniques described in Section 3.3, the identification of facts within the AllRight system is based on a table recognition approach. This method was specifically developed to address the problems classical table extraction methods experience when processing modern webpages, where tables can be created using the various presentation description techniques provided by HTML and Cascading Style Sheets (CSS). These languages provide a number of possibilities to position content of HTML tags so that the text surrounded by these tags will appear as a table in a browser whilst remaining undetectable in, for example, a straightforward analysis of the DOM structure.

The table detection approach used in the Allright project [26] is based on technologies developed in the context of Optical Character Recognition (OCR). The method aims to identify a table on a webpage by considering its *rendered presentation*. Consequently, the table extraction approach is based on a web browser that supports all modern standards and can thus render the majority of webpages correctly.

Fact extraction from tables is implemented using two components, a table interpretation module and a text extractor. Figure 14 sketches the table recognition approach. In order to extract facts from a web document, the table interpretation component first determines the coordinates of the text blocks that would usually be rendered in a browser. The algorithm then uses these coordinates to partition the page and isolate the target table.

More specifically, the algorithm uses a bottom-up approach that includes four main steps (see Figure 14):

(1) Apply predefined heuristics to group text blocks that are most likely positioned in one cell of a table.
(2) Identify columns by finding vertical lines that specify the alignment of text boxes.

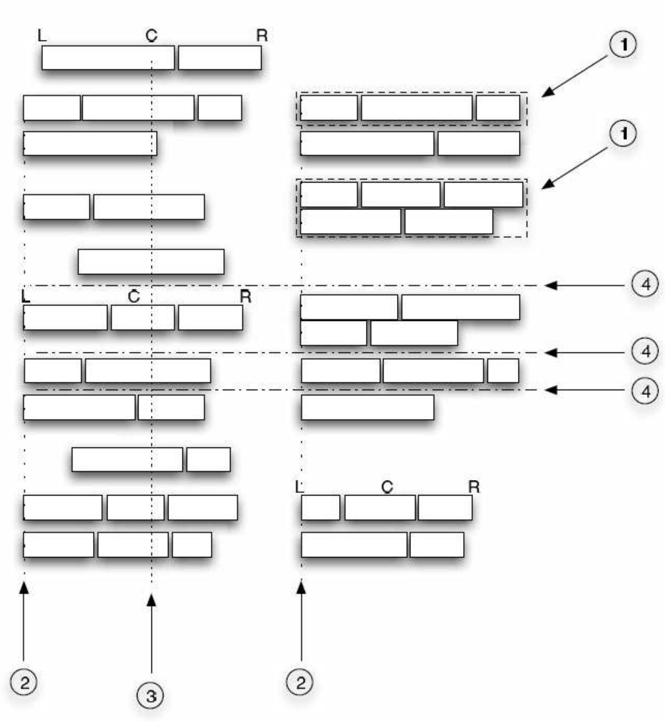


Fig. 14. AllRight table extraction approach [26]

(3) Identify column headers, which are text boxes that share the same coordinates with those in the columns.
(4) Find the best column alignment to form a table.

The table interpretation algorithm can locate more than one table on a webpage. Since it is impossible to decide whether or not any of these tables contains target data, the algorithm stores all of the tables found. Later, the table containing the largest number of detected item characteristics is used and the others are discarded.

5.2 Ontology-based text extraction and conflict resolution

Once the tables are recognized, AllRight's *text extraction* component is used to identify and annotate text that corresponds to attribute and value definitions within the domain ontology. This component includes a number of predefined, specialized spotters, see [19]. Each spotter contains a regular expression template that can efficiently locate specific strings (features) in the text which correspond to attributes and values in the domain ontology. Each spotter associates every found feature with at least one of the definitions of the domain ontology from which its initialization data was taken.

The goal of the fact extraction phase is – roughly speaking – to find correct "attribute-value" pairs that match definitions in the domain ontology. As the extraction process is based on heuristics, different recognition situations can occur:

(1) The found attribute name is valid and the value is consistent with the domain ontology.
(2) The attribute name is valid but the content spotter has found multiple values candidates.
(3) A valid attribute name was found but the associated value found is not consistent with the ontology, i.e. it violates a value restriction defined in the domain ontology.
(4) The found attribute name cannot be uniquely matched with the ontology; values however, have been found.
(5) The attribute name cannot be uniquely matched and no consistent values are found.
(6) The attribute name is not known at all in the ontology; possible values are given though.
(7) Neither the attribute name is known nor have any values been found.

Attribute-value pairs that correspond to Situation (1) are completely recognized and nothing further has to be done. Situation (7) cannot, however, be resolved. The system should simply store the information as a hint to the knowledge engineer that the domain ontology should be revised accordingly, i.e. extended with an additional feature.

In order to deal with the other situations, we first try to analyze (patterns of) attribute-value pairs that appeared in all documents related to the current instance. When, for instance, multiple values are found for an attribute in Situation (2), we check whether a unique value was found in other documents. If such a unique value exists in the majority of the documents , we accept it as the correct value for this attribute. Similar approaches may be followed to resolve Situations (3) to (6).

Of course not all conflicts can be resolved this way. If no valid attribute-value pair can be found that corresponds to the domain ontology, the remaining problematic situations must be resolved by determining the most "plausible" values by sending queries to search engines [9] .

AllRight's web-based extraction and conflict resolution algorithm works as follows. First, the completely recognized attribute-value pairs from situation (1) are used to calculate the Pointwise Mutual Information measure (PMI) to determine a suitable acceptance level for the problematic situations; see also [37]. The measure is calculated by comparing the number of search engine

[9] Previous systems that use comparable fact extraction techniques are described in [8] and [15].

results (hits) returned for two queries $q1$ and $q2$ as follows:

$$PMI(q_1, q_2) = Hits(q_1 \cup q_2)/Hits(q_2). \tag{2}$$

Query $q1$ contains the instance name and the name of the attribute. Query $q2$ consists of $q1$ and in addition the value for the attribute and thus serves as a *discriminator query.* In principle, the number of all documents that describe a certain instance is compared with the number of documents that describe this instance but also mention the specific value. Once the PMI values for all completely recognized attribute-value pairs have been calculated, they are used to train a naive Bayes classifier for each attribute mentioned in the domain ontology. The classifier is then used to determine whether or not a value for an attribute should be accepted based on the web statistics.

After Situation (1) cases have been solved, the conflict resolution procedure continues with the cases from Situation (2) to (6). The threshold values for the attributes are continuously updated throughout this procedure based on the resolution of previous conflicts.

Consider the following simple example Situation (3) problem. Let us assume that the digital camera domain ontology specifies that an attribute “memory type” exists and that “SecureDigital” and “MemoryStick” are allowable values. Furthermore, suppose that a content spotter found the value “CompactFlash” associated with the attribute. Two queries are sent to a public search engine according to the above-described approach. One query includes the inconsistent value “CompactFlash”, which, for instance, leads to a PMI value of 0.002. Based on the knowledge learned from assumedly correct examples from Situation (1), we might however know that the acceptance level for the attribute “memory type” is 0.001. Thus, we accept “CompactFlash” as a value for the attribute and update the classifier to take this new PMI value into account. The same technique is applied – with slight differences – for all Situations (2) to (6).

Note that adding an instance with the value “CompactFlash” for the memory-type attribute introduces an inconsistency in the ontology. Such inconsistencies have to be dealt with by a knowledge engineer, who can, for example, expand such a narrow domain restriction. During this process, the engineer could also be supported by the model-based debugging facility of the AllRight system, which is not only capable of detecting these inconsistencies but also of generating corresponding repair proposals. The details of this component go beyond the scope of this paper and are described in [17].

To evaluate the fact extraction and validation algorithms, we used the same dataset as for the evaluation of the name recognition technique in Section 4.3, i.e. 3135 observations of digital cameras and 2930 documents describing laptop computers.

In Figure 15, recognized attribute-value pairs are grouped into Situations (1), i.e. completely recognized, to Situation (7), i.e. no match. In our experiments, the number of completely unrecognized pairs (Situation 7) was very high because the domain ontology only contained some important attributes whereas many review sites contain long lists of detailed technical information. In the digital camera domain, one review site reported over 60 attributes. Only around 40 of them were defined in the domain ontology. The same phenomenon was observed for laptop computers and is an indication that – depending on the web mining scenario – domain ontologies may often need to be reviewed and extended.

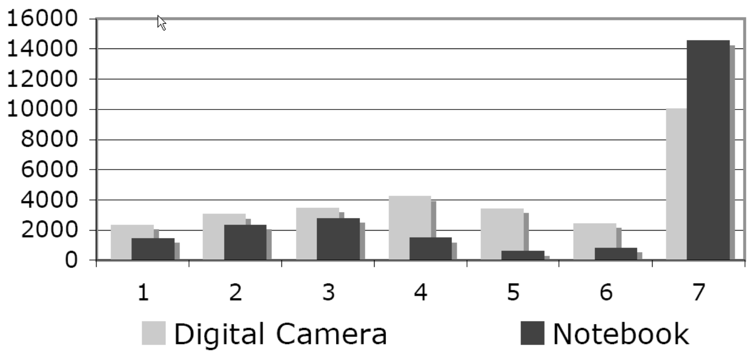


Fig. 15. Number of attribute-value pairs per recognition situation

In order to calculate the standard accuracy metrics for our technique (precision and recall), we again compared the results of the automatic fact extraction process with handcrafted ones. Falsely accepted values and non-assigned attributes were counted as incorrect. The averaged accuracy measures are presented in Table 6.

Domain	Precision	Recall	F-Measure
Digital Camera	0,738	0,91	0,815
Laptop	0,878	0,9275	0,902

Table 6
Fact Validation: Average Precision, Recall and F-Measure for two domains

High values for precision and recall were achieved in both domains, despite the fact that the systems were only initialized with a limited amount of seed knowledge. The average F1-value across all attributes for the digital camera domain is at 0.815. When looking at different attribute types, it can be observed that single-valued features such as the “Effective pixels” achieved the highest values. The average F1-values for multi-valued attributes like the “ISO Rating” were typically lower (average F1: 0.766), mostly because the number of found values for such attributes varied strongly from instance to instance.

With an overall F1-value of 0.902, the system’s performance in the laptop domain was even better than for digital cameras. Slight differences could be observed for different attributes also in this experiment. For the “Processor” attribute, for instance, values were extracted with an average F1-value of 0.859, the “RAM” attribute led to an F1-value of 0.923.

6 Overall ontology instantiation accuracy

Finally, we measured the accuracy of the whole ontology instantiation process in the two domains (digital cameras and laptop computers). A full manual validation of facts was however not possible, since alone for the digital camera domain nearly 19000 attribute value pairs were extracted by the system. Thus, we relied on an *analysis of variance* (ANOVA) in order to reduce the amount of manual validation required and reviewed 946 facts for the camera domain which, according to the Cochran criteria [9], was sufficient to guarantee an acceptable error of 1%. Appropriate random samples of attribute-value pairs were generated for each of the attributes.

Precision was determined by comparing the estimated number of correct attribute-value instances with the number of all instances that were successfully extracted and stored in the domain ontology. *Recall* was measured as the estimated ratio of correct instances to the number of all candidate instances found by the system. This was carried out under the assumption that we are generally interested in instantiating all given attributes of a product (as specified in the domain ontology).

Finally, it has to be mentioned that our experiments were (negatively) influenced by “empty” attributes, i.e. attributes that were defined in the domain ontology but were never found on webpages. This occurred mainly because the manually engineered domain ontology (from a commercial recommender application [16]) used in the experiment also contained attribute descriptions that were outdated. Such empty attribute values were however considered as incorrect in our evaluation thus decreasing recall values.

The overall results for precision, recall and the f-measure are shown in Table 7. Overall, over 77% of all instances of the digital camera ontology and over 85% of the laptop ontology were correctly instantiated. These results are particu-

Domain	Precision	Recall	F-Measure
Digital Camera	0,819	0,74	0,777
Laptop	0,834	0,883	0,858

Table 7
AllRight attribute-value extraction results

larly encouraging as we started with a very small amount of seed knowledge and only relied on domain-independent heuristics and metrics, i.e. no manually defined language or text-dependant methods were used for content spotting.

In the context of the targeted application types such as knowledge-based recommenders or enhanced product search tools, our experiences show that the quality of the retrieved data is sufficiently high for practical purposes, i.e., that the automatically retrieved instance information can serve as a basis for developing commercial applications on top of it.

7 Conclusion

Ontologies are the knowledge bases of the Semantic Web. However, most of the ontologies published on the web only describe domains in terms of concepts and roles and are not able to be used by *knowledge based systems*. The manual instantiation and maintenance of ontologies is both time consuming and error-prone. Therefore, automatization of the ontology instantiation process is necessary to enable the broad usage of the Semantic Web technology.

This paper presented a method for automating the ontology instantiation process by retrieving and extracting data from web sources. In particular, the AllRight system is optimized for dealing with tabular information that is presented in tabular form, which is a data presentation format often used on the web. The developed system relies on a novel combination of new and existing techniques from a variety of fields. The AllRight system includes:

- A crawling component (xCrawl) utilizing two information retrieval strategies: searching and crawling. An evaluation of AllRight's crawler showed that it outperforms a state-of-the-art crawling method in both F-Measure and execution time. Moreover, xCrawl is not dependent on an underlying validation technique and thus can be used wherever source data is available in tabular form.
- A table extraction technique based on optical recognition. This method was shown to be very effective, recognizing more than 95% of all tables presented in different styles on 12 different websites.
- A clustering component for the extracted tables. When applied to tabular data this method is able to cluster documents into single instances in over 90% of cases using metadata from the crawling component.

- Redundancy-based conflict resolution and fact validation algorithms. Prior to AllRight these methods were only used to validate general facts extracted from natural language. The evaluation showed that over 81% of all facts describing digital cameras and over 90% of facts describing laptops could be validated using the suggested technique.

The evaluation of the AllRight system in two popular domains showed that precise instance information can be retrieved even if only a small amount of seed knowledge is available. We thus see our work as a contribution towards the broader application of ontology or semantics-based applications whose adaption is in many cases hampered by the fact that large amounts of ontology instance data have to be manually entered into the system.

References

[1] C. C. Aggarwal, F. Al-Garawi, P. S. Yu, Intelligent crawling on the world wide web with arbitrary predicates, in: Proceedings of the 10th International World Wide Web Conference, New York, NY, USA, 2001.

[2] E. Agichtein, L. Gravano, Querying text databases for efficient information extraction, in: Proceedings of the 19th IEEE International Conference on Data Engineering, Bangalore, India, 2003.

[3] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Proceedings of the 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile, 1994.

[4] H. Alani, S. Kim, D. E. Milard, M. J. Weal, W. Hall, P. H. Lewis, N. R. Shadbolt, Automatic ontology-based knowledge extraction from web documents, IEEE Intelligent Systems 18 (2003) 14–21.

[5] A. Bergholz, B. Chidlovskii, Crawling for domain-specific hidden web resources, in: Proceedings of the Fourth International Conference on Web Information Systems Engineering, Washington, DC, USA, 2003.

[6] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, Computer Networks and ISDN Systems 30 (1-7) (1998) 107–117.

[7] S. Chakrabarti, K. Punera, M. Subramanyam, Accelerated focused crawling through online relevance feedback, in: Proceedings of the 11th International World Wide Web Conference, Honolulu, Hawaii, USA, 2002.

[8] P. Cimiano, G. Ladwig, S. Staab, Gimme' the context: context-driven automatic semantic annotation with C-PANKOW, in: Proceedings of the 14th International World Wide Web Conference, Chiba, Japan, 2005.

[9] W. Cochran, G. Cox, Experimental designs, Wiley New York, 1957.

[10] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, S. Slattery, Learning to construct knowledge bases from the World Wide Web, Artificial Intelligence 118 (1) (2000) 69–113.

[11] A. Dasgupta, A. Ghosh, R. Kumar, C. Olston, S. Pandey, A. Tomkins, The discoverability of the Web, in: Proceedings of the 16th International World Wide Web Conference, Banff, Alberta, Canada, 2007.

[12] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, M. Gori, Focused crawling using context graphs, in: Proceedings of 26th International Conference on Very Large Data Bases, Cairo, Egypt, 2000.

[13] S. Dill, J. Tomlin, J. Zien, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, et al., SemTag and Seeker: Bootstrapping the Semantic Web via automated semantic annotation, in: Proceedings of the 12th International World Wide Web Conference, 2003.

[14] M. Ester, M. Grob, H. Kriegel, Focused Web crawling: A generic framework for specifying the user interest and for adaptive crawling strategies, in: Proceedings of 27th International Conference on Very Large Data Bases, Roma, Italy, 2001.

[15] O. Etzioni, M. Cafarella, D. Downey, A. Popescu, T. Shaked, S. Soderland, D. Weld, A. Yates, Unsupervised named-entity extraction from the Web: An experimental study, Artificial Intelligence 165 (1) (2005) 91–134.

[16] A. Felfernig, G. Friedrich, D. Jannach, M. Zanker, An integrated environment for the development of knowledge-based recommender applications, International Journal of Electronic Commerce 11 (2) (2007) 11–34.

[17] G. Friedrich, K. Shchekotykhin, A General Diagnosis Method for Ontologies, in: Proceedings of the 4th International Semantic Web Conference (ISWC-05), Galway, Ireland, 2005.

[18] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, B. Pollak, Towards domain-independent information extraction from web tables, in: Proceedings of the 16th International World Wide Web Conference, Banff, Alberta, Canada, 2007.

[19] W. Holzinger, B. Krüpl, M. Herzog, Using ontologies for extracting product features from web pages, in: Proceedings of the 5th International Semantic Web Conference, Athens, Georgia, 2006.

[20] P. G. Ipeirotis, E. Agichtein, P. Jain, L. Gravano, To search or to crawl? Towards a query optimizer for text-centric tasks, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, 2006.

[21] D. Jannach, Advisor Suite – a knowledge-based sales advisory-system, in: Proceedings of the 16th European Conference on Artificial Intelligence, Valencia, Spain, 2004.

[22] J. Jobson, Applied multivariate data analysis, Springer, 1992.

[23] J. Kleinberg, Authoritative sources in a hyperlinked environment, Journal of the ACM (JACM) 46 (5) (1999) 604–632.

[24] R. Kosala, H. Blockeel, Web mining research: a survey, SIGKDD Explorer Newsletter 2 (1) (2000) 1–15.

[25] A. Kruger, C. L. Giles, F. Coetzee, E. Glover, G. Flake, S. Lawrence, C. Omlin, Deadliner: Building a new niche search engine, in: Proceedings of 9th International Conference on Information and Knowledge Management, Washington, DC, 2000.

[26] B. Krüpl, M. Herzog, Visually guided bottom-up table detection and segmentation in Web documents, in: Proceedings of the 15th International World Wide Web Conference, Edinburgh, Scotland, 2006.

[27] F. Naumann, A. Bilke, J. Bleiholder, M. Weis, Data fusion in three steps: Resolving schema, tuple, and value inconsistencies, IEEE Data Engineering Bulletin 29 (2) (2006) 21–31.

[28] D. Pelleg, A. W. Moore, X-means: Extending k-means with efficient estimation of the number of clusters, in: Proceedings of 17th International Conference on Machine Learning, Stanford, CA, 2000.

[29] G. Petasis, V. Karkaletsis, C. Spyropoulos, Cross-lingual information extraction from Web pages: the use of a general-purpose text engineering platform, in: Proceedings of the 4th International Conference on Recent Advances in Natural Language Processing, Borovets, Bulgaria, 2003.

[30] A.-M. Popescu, O. Etzioni, Extracting product features and opinions from reviews, in: Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing, Morristown, NJ, USA, 2005.

[31] J. Rennie, A. McCallum, Using reinforcement learning to spider the Web efficiently, in: Proceedings of the Sixteenth International Conference on Machine Learning, San Francisco, CA, USA, 1999.

[32] U. Schonfeld, Z. Bar-Yossef, I. Keidar, Do not crawl in the DUST: different URLs with similar text, in: Proceedings of the 15th International World Wide Web Conference, New York, NY, USA, 2006.

[33] K. Shchekotykhin, D. Jannach, G. Friedrich, Clustering web documents with tables for information extraction, in: Proceedings of the 4th International Conference on Knowledge Capture, Whistler, Canada, 2007.

[34] K. Shchekotykhin, D. Jannach, G. Friedrich, xCrawl: A high-recall crawling method for Web mining, in: Proceedings of 8th International Conference on Data Mining, Pisa, Italy, 2008.

[35] K. Shchekotykhin, D. Jannach, G. Friedrich, O. Kozeruk, AllRight: Automatic ontology instantiation from tabular Web documents, in: Proceedings of 6th International Semantic Web Conference, Busan, Korea, 2007.

[36] G. Stoilos, G. B. Stamou, S. D. Kollias, A string metric for ontology alignment, in: Proceedings of the 4th International Semantic Web Conference, Galway, Ireland, 2005.

[37] P. Turney, Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews, in: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, PA, 2002.

[38] I. Witten, E. Frank, Data mining: Practical machine learning tools and techniques with Java implementations, Morgan Kaufmann, 2000.

[39] H. Yu, J. Han, K. C.-C. Chang, PEBL: positive example based learning for Web page classification using SVM, in: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 2002.