

# DataSpider: discovering data on the web

---

*Sven Groot (0024821)*

## Table of Contents

1	Introduction.....	1
2	Related work.....	1
3	Approach.....	6
3.1	Tools.....	7
3.2	Heuristics .....	7
3.3	Limitations.....	8
4	Results.....	10
4.1	Representativeness.....	11
4.2	Breakdown by used fields .....	12
4.3	Integrating data.....	13
4.3.1	Results from names with multiple occurrences.....	14
5	Conclusion.....	15
6	References.....	17
	Appendix A. Breakdown by used fields .....	19
	Appendix B. Top most occurring names.....	20
	Appendix C. Result samples.....	22
	Appendix D. Statistical breakdown of the search-based experiment.....	24

## 1 Introduction

The World Wide Web is quite likely the largest repository of data in the world. Data about every imaginable subject, in every imaginable language, is kept in the billions upon billions of web pages that are out there.

Unfortunately, most of that data is kept in unstructured form. The only structure that has been applied to it is HTML, which is meant to describe document structure and presentational information, not data structure. This, while perfectly suited to humans, makes it nearly impossible for a computer to process the data automatically.

There are many attempts underway to give the web meaning. The World Wide Web Consortium (W3C) has been attempting for some time to push the Semantic Web, where existing web pages, formatted for human use, are annotated using additional semantic data.

Unfortunately, the web as it is today does not contain this additional semantic data. The problem of the semantic web is that it depends on the authors of web pages to modify their web sites so they include this data. This means that the existing stores of unstructured data continue to be unusable until such time when their authors decide to add these semantic elements. Considering the fact that much of the web today does not (correctly) comply with existing standards such as HTML and CSS, I find it very unlikely that web authors would take the necessary steps to add semantic content, which not only needs to be syntactically correct (more so than HTML to facilitate correct machine processing) but also semantically correct.

Therefore, it is desirable to be able to use the data that is already on the web today, in its unstructured form. This paper documents the DataSpider project, which is an attempt at investigating the feasibility of mining the existing unstructured data on the web, as well as a very limited attempt at classifying what kind of data can be found.

## 2 Related work

Collecting data from the web in an automated fashion is certainly not a new thing. Search engines have been indexing pages for a long time, and have been fairly successful at doing so. Search engines traditionally do not assign any semantic meaning to any of the data they collect. Indeed, early search engines were little more than full text indices of the entire web. As the web grew, this approach grew less and less effective. Automated search engines that rely on keyword matching usually return too many low quality matches. To make matters worse, some advertisers attempt to gain people's attention by taking measures meant to mislead automated search engines.

To rectify this problem, modern search engines, such as Google, do use some of the inherent semantic structure of hypertext documents. Specifically, Google makes use of the links in a page. By counting the number of citations (or links) to a given page, Google calculates its "PageRank" [1], its relevance, and uses this to order the results of a keyword search.

Even so, Google still extracts no semantics from the contents of the page itself. Search engines are incapable of answering queries such as "*What are the research interests of professor X at university Y?*" or "*What are the institutions in Europe that published more than 20 documents on databases in 2004?*", except when the answer to such queries can be found on a page that also contains the query text. This makes it impossible if the query requires the joining of data from more than one page.

Indeed, search engines do not aim to answer your questions; they aim to provide documents that have some relevance to your query. It is then up to you to analyze these documents to see if they hold the information you are looking for. As such, search engines are not truly collecting data on the web. They will treat names no different from telephone numbers; all data is simply a bunch of abstract strings to them with no meaning.

Closely related to web searching is the field of Question Answering (QA). When a search engine is presented with a natural language query, such as “what is a hard disk?”, it will simply attempt to find documents containing the words from that phrase, instead of recognizing the phrase as a question and treating it as such. QA attempts to find better ways to answer such questions. Existing search engines that do QA include AskJeeves ([www.ask.com](http://www.ask.com)) which uses databases of pre-compiled information, metasearching and other proprietary methods, and services such as AskMe ([www.askme.com](http://www.askme.com)) and Google Answers ([answers.google.com](http://answers.google.com)) that facilitate interaction with human experts.

Approaches that attempt to solve QA in an automated way often focus on text mining and machine learning. In such approaches, the software is provided with a corpus of sample questions/answers that help it identify question types and do phrase analyses. One such approach uses a sample corpus to learn the software a language model that tells it how to transform natural language questions into more useful queries that are more likely to yield the correct result when provided to a conventional search engine [2].

These approaches however still do not deal with finding relational data; they simply attempt to make search results more relevant.

There have been attempts to get relational data from web pages. G. Mecca et al [3][4][5] describe a system they call a Web-Base that can be used to manage relational data in web pages. Their approach is centred around the idea of page schemas, that describe the data that can be found on web pages. These page schemas, in combination with special entry-point pages, can be used to query a web site for structured or semi-structured relational data.

Information presented in a website is usually organized according to certain logical structure that corresponds to users' common perception. If the data models that information extraction systems produced reflect this commonly perceived structure, most users will be able to understand and use them easily. Unfortunately, models such as the Araneus model either only deal with information in page-level or represent the extracted information in some proprietary and ad-hoc data structures that do not truly reflect the original logical view of the websites. An alternative data model that better models the logical view of data across pages is proposed by Z. Liu et al [6].

One problem that remains is the creation of such schemas, sometimes called wrappers, which are needed to extract data from a web page in this fashion. Many attempts have been made to at least partially automate the process of wrapper generation. Laender et al [10] provides a broad overview of some of the tools available. Wrapper generation tools are classified in several categories.

HTML-aware tools rely on inherent structural features of HTML documents for accomplishing data extraction. Before performing the extraction process, these tools turn the document into a parsing tree, a representation that reflects its HTML tag hierarchy. Following, extraction rules are generated either semi-automatically or automatically and applied to the tree. Typically, user interaction is required to specify the pieces of data to extract and where to find them in a document. Alternatively, the tools sometimes learn the schema from being presented with several examples of pages derived from the same template, such as several different book pages from amazon.com.

Natural language processing (NLP) techniques have been used by several tools to learn extraction rules for extracting relevant data existing in natural language documents. These tools usually apply techniques such as filtering, part-of-speech tagging, and lexical semantic tagging to build relationship between phrases and sentences elements, so that extraction rules can be derived. Such rules are based on syntactic and semantic constraints that help to identify the relevant information within a document. The NLP-based tools are usually more suitable for Web pages consisting of free text, possibly in telegraphic style, such as job listings, apartment rental advertisements and seminar announcements.

Wrapper Induction Tools generate delimiter-based extraction rules derived from a given set of training examples. The main distinction between these tools and those based on NLP is that they do not rely on linguistic constraints, but rather in formatting features that implicitly delineate the structure of the pieces of data found. This makes such tools more suitable for HTML documents than the previous ones.

Modelling-based Tools include tools that, given a target structure for objects of interest, try to locate in Web pages portions of data that implicitly conform to that structure. The structure is provided according to a set of modelling primitives (e.g., tuples, lists, etc.) that conform to an underlying data model. Following, algorithms similar to those used by the wrapper induction tools identify objects with the given structure in the target pages. This approach most closely matches our DataSpider approach in that we decide a model a priori and try to find data conforming to it across the web.

Ontology-based Tools differ from all approaches described previously because those rely on the structure of presentation features of the data within a document to generate rules or patterns to perform extraction. However, extraction can be accomplished by relying directly on the data. Given a specific domain application, an ontology can be used to locate constants present in the page and to construct objects with them.

HTML aware tools are by far the most common, as it is the only real structure that is inherently present in all pages on the web, and that is not reliant on either the domain or the language of the text. Arasu et al [7] discuss a method of finding a schema based on a set of pages known to have been generated using the same template, for instance the book details pages on Amazon, which contain information about different books but have the same structure. Yang et al [8] point out the difficulties presented to such an approach by null values, which are often omitted from pages. Lerman et al [9] use an approach that uses the fact that many websites will present a list, often in the form of a table, with links to a details page. Zhang et al [11] attempt to extract relational data from websites based on an example set of records, using it to find occurrences of records in web pages, extract patterns from those occurrences, and subsequently extract data using those patterns. They also attempt to solve the problem of duplicate results, which is a complex one since simply matching the data literally will not do. Buttler et al [12] use an approach where the objects of interest are defined a priori and the system automatically learns how to extract these objects from pages encountered using a tree-based parsing system, with high rates of success. Another tree-based system is described by Chang et al [13].

It remains however a problem that there is no one set of rules that can be applied to all pages on the web, and no way to truly automatically derive these rules with enough accuracy so that data could be gathered from all over the web without human intervention. As long as human interaction is needed to extract data from a webpage, it remains infeasible to use this approach on large scale web mining. Finding better ways to automate wrapper and model generation is an area of much research.

One particular approach that shows much promise is covered by Zhai et al [15], which in an extension of their work in [14]. A method is proposed that uses HTML structure, as well as visual layout, to find recurring patterns on a webpage. This is based on the idea that data on a webpage is most often represented in a contiguous region with a repeating structure. By identifying such repeating structures in the DOM tree of the HTML document, likely data segments can be located. Visual information is added to this approach which helps identify gaps between data segments, further helping the identification of those segments. Once the segments have been located, they are aligned to find the data in the segments, after which data can be extracted with a reasonably high degree of accuracy.

One of the drawbacks of the approach, at least so it seems to me, is that it is incapable of labelling the data. The alignment step makes sure that all book titles from a list of books from amazon.com end up in the same column, but the algorithm cannot deduce that the name of that column should be book title. As such, it also cannot know that book titles from amazon.com may be comparable to book title from barnesandnoble.com.

Wang et al [16] propose a system that tries to solve the problem of label assignment in wrapper generation. Their approach is based on several heuristics. They attempt to use form labels for search forms used to query the site's data for labelling (this seems to me problematic at best, since far too few sites actually employ the HTML <label> element), it uses header cells in a table which is an approach also used by DataSpider, it uses prefixes and suffixes shared by all data items on a webpage, and it also looks for known formats, which is also something DataSpider does.

So far, none of these approaches are truly successful in integrating data from different sources. There is an entirely different breed of approaches that attempts to deal with this. So far, we have discussed mainly systems that attempt to extract data from web pages, be they search engines or automated wrapper generators and data extractors. The other way to go about this is to have servers provide the data in a more structured manner in the first place. This solves all the problems with parsing and "understanding" the semantics in unstructured or semi-structured documents, at the cost of asking some work from the data providers.

There already are some systems that approach this idea, usually focussing on a single domain. For instance TheDataWeb (<http://www.thedataweb.org/>) provides access to a wide variety of mainly demographic data, such as census data, economic data, health data, income and unemployment data, population data, labour data, cancer data, crime and transportation data, family dynamics, vital statistics data and more. TheDataWeb focuses on microdata, aggregate time series as well as longitudinal data. Servers can provide their own data to TheDataWeb by providing a metadata file, in a predefined format, which contains a description of the data you provide, allowing TheDataWeb to use it. Users can download an application called the DataFerret, which facilitates searching and integrating data from all the sources contained in TheDataWeb.

One of the largest movements in the past years to attempt to get semantic information out of, or rather, into, the web has been the W3C's Semantic Web strategy. The Semantic Web is a new World Wide Web that will exist side by side with today's Web. Where the Web we all know and use today is a web of pages, the Semantic Web is a web of data. The Semantic Web takes the approach that the concept of machine-understandable documents does not imply some magical artificial intelligence which allows machines to comprehend human mumblings. It only indicates a machine's ability to solve a well-defined problem by performing well-defined operations on existing well-defined data. Instead of asking machines to understand people's language, it involves asking people to make the extra effort.

The semantic web is based on several standard formats that web sites can use to provide data in a structured, self-describing format. The basis for this is the Resource Description Framework (RDF) [17].

RDF is a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page, copyright and licensing information about a Web document, or the availability schedule for some shared resource. However, by generalizing the concept of a "Web resource", RDF can also be used to represent information about things that can be identified on the Web, even when they cannot be directly retrieved on the Web. Examples include information about items available from on-line shopping facilities (e.g., information about specifications, prices, and availability), or the description of a Web user's preferences for information delivery. RDF is based on the idea of identifying things using Web identifiers (called Uniform Resource Identifiers, or URIs), and describing resources in terms of simple properties and property values. This enables RDF to represent simple statements about resources as a graph of nodes and arcs representing the resources, and their properties and values. Although RDF is often serialized as XML [18] it is a fundamentally different mindset; RDF defines a graph where XML defines a tree, and RDF is concerned with semantics where XML is concerned with syntax. Because RDF uses URIs to refer to objects it is making statements about, which in themselves can also be other RDF statements, the Semantic Web is a true web.

The second part of the puzzle is RDF Schema [19], which allows you to define metadata about the objects that are used in RDF graphs. RDF Schema defines classes and properties that may be used to describe classes, properties and other resources.

And finally, the Semantic Web provides an ontology language, OWL [20]. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. Ontologies are critical for applications that want to search across or merge information from diverse communities. Although XML DTDs and XML Schemas are sufficient for exchanging data between parties who have agreed to definitions beforehand, their lack of semantics prevent machines from reliably performing this task given new XML vocabularies. The same term may be used with (sometimes subtle) different meaning in different contexts, and different terms may be used for items that have the same meaning. RDF and RDF Schema begin to approach this problem by allowing simple semantics to be associated with identifiers. With RDF Schema, one can define classes that may have multiple subclasses and super classes, and can define properties, which may have sub properties, domains, and ranges. In this sense, RDF Schema is a simple ontology language. However, in order to achieve interoperation between numerous, autonomously developed and managed schemas, richer semantics are needed. For example, RDF Schema cannot specify that the Person and Car classes are disjoint, or that a string quartet has exactly four musicians as members.

The Semantic Web model is very closely connected with the relational database model [21]. A collection of RDF statements about a node corresponds to a row in a table. A database join is a splicing of graphs. Relational databases are optimized to handle large numbers of instances of statements using the same property, and there might be corresponding optimizations in an XML serialization for large volumes of similar data. But we should expect that the basic structures that support serializing relational databases can be shared with the RDF Directed Labeled Graph data model.

The original RDB model defined by E.F. Codd included datatyping with inheritance, which he had intended would be implemented in the RDB products to a greater extent that it has. For

example, typically a person's home address house number may be typed as an integer, and their shoe size may also be typed as an integer. One can as a result join to tables through those fields, or list people whose shoe size equals their house number. Practical RDB systems leave it to the application builder to only make operations which make sense. Once a database is exported onto the Web, it becomes possible to do all kinds of strange combinations, so a stronger typing becomes very useful: it becomes a set of inference rules.

RDF is great as a basis for ER-modelling, but because RDF is used for other things as well, RDF is more general. RDF is a model of entities (nodes) and relationships. If you are used to the "ER" modelling system for data, then the RDF model is basically an opening of the ER model to work on the Web. In typical ER model involved entity types, and for each entity type there are a set of relationships (slots in the typical ER diagram). The RDF model is the same, except that relationships are first class objects: they are identified by a URI, and so anyone can make one. Furthermore, the set of slots of an object is not defined when the class of an object is defined. The Web works though anyone being (technically) allowed to say anything about anything. This means that a relationship between two objects may be stored apart from any other information about the two objects. This is different from object-oriented systems often used to implement ER models, which generally assume that information about an object is stored in an object: the definition of the class of an object defines the storage implied for its properties.

The Semantic Web is a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web. Although there is no doubt that this is the ideal solution, where all data is expressed in an easily machine readable format, and all attempts to parse and extract data from unstructured documents become unnecessary. Unfortunately, the Semantic Web requires effort on the part of web masters. Not only do they need to know about and understand the RDF model, which although not very complex is such a departure from the XML that most web masters are used to thinking about can take quite some getting used to, they also need convert their data to RDF, providing the appropriate schema and ontologies. Before the Semantic Web becomes useful, it needs a substantial amount of effort, and therefore time, and at this point it is unknown whether it will ever actually happen. Of course, many of the same objections were raised more than ten years ago when the web as we know it today was in its early stages, so the Semantic Web may yet become successful.

For the moment though, we are stuck using the more cumbersome and problematic methods of dealing with the existing unstructured or semi-structured web.

### 3 Approach

Before data can be collected, we must first decide what kind of data to focus on. Although it would be tempting to say "all of it", this is of course not actually possible. Because we are dealing with unstructured data, we need some way of identifying pieces of plain text as data, as well as a way to identify structure in the data.

For our approach, we have decided to focus on tabular data. This data is relatively well structured, and easily identifiable. It is also relatively easy to store this data in a relational database, since that uses the table approach as well. There is however one large problem with this approach, and that is that tables are often used on the world wide web not for tabular data, but for layout. As much as I would want everybody to use XHTML in a semantically valid manner, and use CSS for layout, this is simply not the case on today's world wide web. It is therefore

necessary to separate the wheat from the chaff, and decide which tables contain data and which tables are purely layout.

Once a table that could contain useful data has been located, we must still establish if it actually does, and if so, what that data is. To achieve this, I have used a combination of some simple heuristics rules and regular expression based parsing.

### 3.1 Tools

The main application that has been developed here is basically a crawler: it crawls web pages looking for data. This application, called DataSpider, has been developed using Microsoft .Net 1.1.

The crawler part of this application is based on the C# WebSpider example at CodeProject (<http://www.codeproject.com/csharp/DavWebSpider.asp>), but it has been heavily modified to overcome some of the limitations. For one thing, the example spider could not crawl outside of the domain you started at, and that was obviously needed here. Other modifications have been made, including support for multiple worked threads, reduction of the memory footprint and the time out period for non-responsive websites.

Initial versions of the DataSpider stored the retrieved data directly in a MySQL database, but this turned out to be too slow. Instead, the final version simply dumps the result to a comma-separated values text file, using a separate file for each worker thread to minimise thread synchronization. A second utility, called SpiderImport was written to import these files into the MySQL database for analysis and processing. Existing CSV import utilities could not be used because of some small differences in the actual format of DataSpider's text files and the usual format of CSV files.

In order to process the HTML documents found on the web, an HTML parser would be needed. If all documents followed the HTML spec (or even better, the XHTML spec) to the letter, this would have been a simple matter, but once again that is unfortunately not the case. Initially I used MSHTML via COM InterOp, but this was much too slow, and dependant on the global IE security settings, and since scripting and ActiveX had to be disabled to make it work reliably, this was not desirable. In the final version, I use the HTML Agility Pack (TODO: Reference), which is a tagsoup parser written entirely in C#, which allows me to treat the parsed HTML as if it were valid XML, regardless of how badly written it originally was. This means I can do XPath queries against the document, treating all tag names as if they were lower case, even if they were not in the source document, making processing significantly easier.

### 3.2 Heuristics

Now that we can get at the HTML, and have a way to make sense of it, it is necessary to identify data and discard useless bits. With the HTML parsed into an XML-like DOM, it is now trivial to find tables with a simple `//table` XPath query.

However, as was noted earlier, often tables are used for layout rather than tabular data. In order to make sure no time is wasted evaluating tables that are likely not data tables, the first heuristics rule used is to discard all tables that contain other tables. Nested tables are a commonly used tool in creating complex layouts when CSS is not used, so this is a strong indication of a layout table. Secondly, we discard all tables that have cells spanning multiple rows or columns. As it is, the system doesn't have a good way to deal with such cells anyway even if they did contain data.

Now we are left with tables that have a reasonable chance of containing useful data. The first step in extracting that data is to look at the table header (note: because thead and tbody are rarely used correctly on the web, they are ignored; instead the first row is assumed to be the

table header). Clues in the text of a table header cell are used to determine the semantic meaning of the columns contents. We can for instance assume that a column labelled “name” will contain name data, and one labelled “e-mail” will contain e-mail addresses. The DataSpider uses a few easy to identify keywords for this rule, in both Dutch and English. Doubtless it still misses a great many possibilities, but a start has to be made somewhere. If these heuristics yield results, no further processing is done on this row. The column names themselves are not stored, but the results from this processing step are used to determine which column in the result set to use to store data from the column.

Finally, the contents of the table cells are evaluated for data. One simple rule that is used is that if a cell contains an anchor element whose href attribute uses the mailto: protocol, the e-mail address from the attribute is collected. Further processing is done using DataTypeXml.

DataTypeXml is a technology that I developed for a different project. The aim of DataTypeXml is to allow for format analysis of plain text data. Text can be processed using regular expressions, where semantic meaning can be assigned to all or parts of the text. For instance the string “16-08-2005” can not only be identified as a date, the tool can also determine that 16 is the day, 8 is the month and 2005 as the year. The rules for identifying these strings are stored in an xml file (hence the name DataTypeXml), and include not only regular expressions, but can also include other expressions that can validate or modify part of the string before yielding the result. This is for instance used to recognise that 30-02-2005 cannot be a valid date. DataTypeXml is capable of several more things, such as comparing parsed strings based on their semantic meaning, and generating ODBC-compliant SQL literals for the data types, but these are not used by the DataSpider as they fall outside of its intended purpose.

DataSpider uses DataTypeXml to identify the following types of data: currency amounts (e.g. “€ 5,30”), e-mail address (e.g. [someone@example.com](mailto:someone@example.com)), Dutch post codes (e.g. “2333 CA”), British post codes (e.g. “SW7 2AP”), phone numbers (e.g. “071-5277061”), and dates (e.g. “16-12-2005”).

### 3.3 Limitations

The approach used by the DataSpider has several limitations. One is immediately obvious: if the data is not in tables, it won’t be found. Should for instance someone’s address be formatted in the usual envelope address form instead of a table, nothing will be found.

Another limitation is that only row-based data is considered. If data is in fact not grouped by row but by column, it is not correctly identified.

One final limit is that in the anchor heuristic is used to identify an e-mail address, the actual contents of the anchor element are ignored, even though they might very well contain a name.

And of course it cannot recognise data if it does not use one the formats described in the DataTypeXml format file.

I have also discovered a flaw in the handling of zipcodes. If a column is identified as a zipcode column by its header, it gets labelled with the type “zipcode”. If the contents of the cells of that column are recognized by the DataTypeXml rules as well, they’ll get labelled as either “dutchzipcode” or “britishzipcode”. This causes DataSpider to erroneously label the row as inconsistent, which causes it to discard it. Unfortunately, this rather serious flaw was discovered too late to correct and collect new results to base this paper on.

Despite these limitations, the resulting simplicity is also a strength of the DataSpider. It is completely automated and still fast enough to be applied to a large corpus of sample data which is needed for our experiment. Many of the more advanced approaches described in section 2,

while far more successful in finding and extracting data, are simply far too slow to do what we intended to do, and those approaches that require human interaction are ruled out by default.

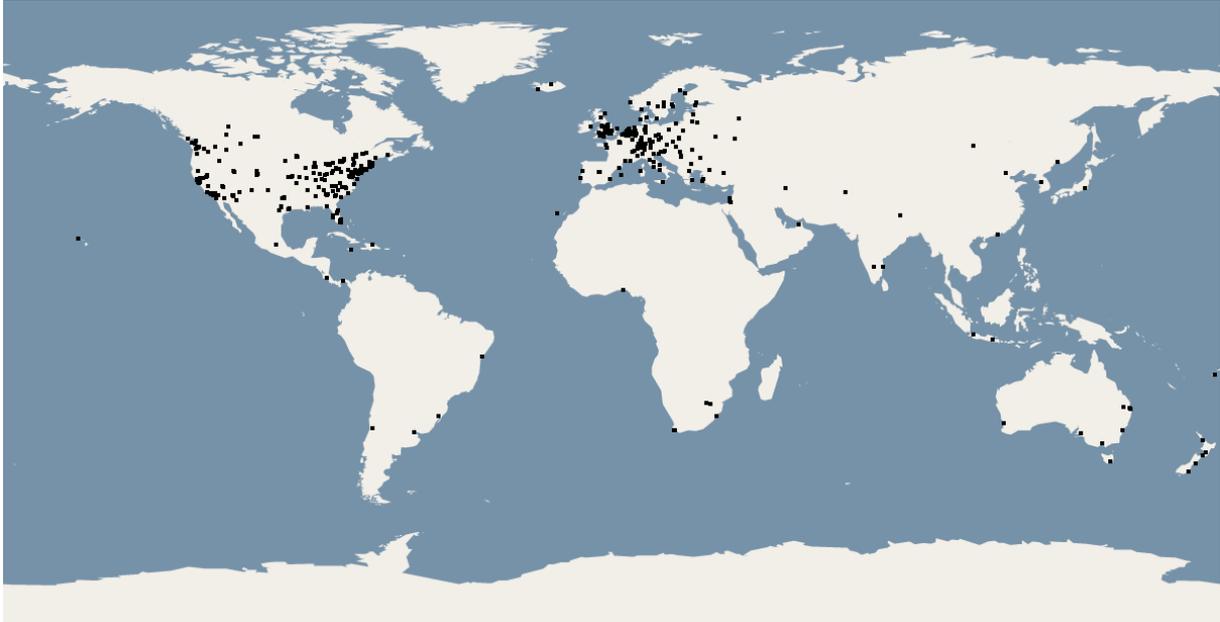


Figure 1 Geographical breakdown of web servers

## 4 Results

After running the DataSpider, in several different, constantly tweaked, versions for several weeks, it had managed to visit approximately 7,500,000 web pages. Because of memory and performance constraints it was not possible to completely eliminate visiting a page twice while the experiment was conducted. Unfortunately we have not kept sufficient records to determine after the fact how many duplicates there were, however applying the ratio of duplicate urls in the results to the total amount of urls leads us to an estimate of approximately 6,000,000 unique pages visited.

After importing the data into a MySQL database, eliminating duplicates, a total number of 156,394 rows has been collected from 15,784 web pages on 4,393 different domains. A geographical breakdown of the locations of the servers for these domains is shown in figure one. Note that for 2,477 of the visited IP addresses the location could not be determined. The large number of servers visited in the US and Europe is easily explained by the fact that our search method prefers English language results, and is in fact incapable of finding results in most other languages.

It is important to know that, even though duplicate urls have been removed, this doesn't mean that there are absolutely no duplicates in the database. Unfortunately, there are many ways to write a url that maps to the same path. Although it is sometimes possible to normalise these urls so that they become the same, another hurdle exists in the form of urls that really are different (often only by query string) but return the same data. So it cannot be avoided that there are still some duplicates left.

One thing that is immediately obvious is the low ratio of pages that yield results. Using the corrected estimation of six million pages, only about 2.6% actually yielded a result. This doesn't mean that the remaining 99.7% of pages don't contain any data, but simply that any data that might have been there is not in a form that the DataSpider could find. What form this data does take and whether it's structured enough to extract with an automated process is beyond the scope of this paper.

## 4.1 Representativeness

If we want to draw any conclusions from our results, it is important to consider how representative our sample is to the entire World Wide Web.

In order for a sampling to hold true predictive value, it must be a random sample. Unfortunately, the nature of the Internet makes it difficult to be truly random, and the method by which the five million results have been collected was not random. The DataSpider selected a few starting points using Google, and then started following links. Because of this, it is very difficult to reason about the representativeness of this sample.

In order to more effectively approach this, an additional experiment was performed. In this experiment, a sample was collected that consists of a number of “runs”, each of which used a different starting url from which around 5000 pages were visited by following links. The starting url was chosen by searching in Google using a randomly selected word from a dictionary, and then choosing a random search result from the top 10 (selecting from more of the results would have been desirable, but unpractical due to limitations of the Google API).

We assume that the dictionary-based Google search is random. The web is thought to have a structure consisting of clusters of pages that link to each other, where each cluster will have links to other clusters. A large central core is thought to exist which contains a large number of interlinked pages, and a periphery of less densely linked clusters that link to the core (see for instance [22]). Google’s PageRank algorithm is more likely to return results from the core since it is based on link counts. Because the used dictionary is English we are further limited to English-language result pages; this is not a problem since DataSpider would not be able to extract data from most other languages anyway. Based on this, we can reasonably assume that our starting points are random within the core of the English-language web and perhaps also some portion of the periphery.

Our sample then consists of in total 40 random collections of 5000 urls. The 5000 urls are likely to be related, but the 40 collections are random and have no correlation. No effort is made to ensure that the collections do not overlap, so we can consider this as random sampling with replacement.

For each of these collections we measure the number of pages that contained data. The results of this can be seen in Appendix D.

We calculate the mean of the sample:

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$$

With  $n = 40$  and  $X_i$  the number of pages that contained data for each sample.

Next we estimate the population standard deviation using the sample:

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2}$$

Although we would need to correct the bias of this sample, the number of pages on the web is sufficiently high compared to the number of pages in the sample that this makes no difference.

Finally, we estimate the standard error for sampling with replacement:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}}$$

The results of all these values are shown in the table in Appendix D. Although the standard deviation is rather high, we see that this is mainly caused by a few high values such as 62 and 55. On the whole, the impression given by the table is that 10 is indeed a reasonable expected value for the number of pages with data.

The estimated standard error shows us that we are already fairly close to the expected actual mean. Since sample means from multiple samplings are distributed normally around the actual mean, it means that when repeating this experiment 68% of the experiments should yield a result of  $10.7 \pm 2.3$ , which is very reasonable. Considering the estimated standard error will half when the amount of samples is multiplied by four, the already quite low value of the standard error compared to the estimated mean, and the amount of time it takes to gather samples we consider this to be good enough for our experiment.

This becomes interesting when we go back to our original experiment of 5,000,000 pages. From the above experiment we have concluded with reasonably accuracy that we would expect approximately 10 in every 5000, or 0.23% of pages to contain data. In the main experiment, approximately 15,000 pages out of 6,000,000 contained data, which is around 0.26%. These results are very close indeed, which leads me to conclude that the data set is reasonably representative of the world wide web.

One further thing to consider is the issue of domains. So far we have looked at pages, however it stands to reason that the pages within one domain are likely to be similar in the types of data that they can contain. As such, a domain with a large amount of pages that contain data tends to bias the results toward the type of data that is one this domain. It would therefore be better to examine the results not on a per-page but a per-domain basis, however as you can see from Appendix D the number of domains visited in a single run varies tremendously. This means not only that we cannot reason about domains based on the second experiment, but that it is also impossible to estimate the number of domains visited in the main experiment since we did not keep track of it while the experiment was run. Although a domain-based breakdown would be desirable, it would require a significantly different approach to collecting pages, and is therefore not done here. It is however important to keep this in mind, especially when looking at the numbers from Appendix A.

## 4.2 Breakdown by used fields

Shown in Appendix A is a breakdown of all records based on which fields were captured. I have chosen to ignore the “other” field, since that field is present on all records. Because it is always present, including it adds no information; any combination of  $n$  fields including other automatically has the same number of occurrences as the set of fields  $n - \{\text{other}\}$ .

We will first take a look at individual fields. The runaway winner here is quite obviously name. This may seem a surprising result considering that name is the only field that can only be recognized by the table header heuristic described in section 3.2. All the other fields have other means of discovery (by regular expressions) as well as table headers, which makes it seem more likely that those fields would surpass name.

So it would appear that tables quite often use a header field that allows the contents of the column to be identified as a name. This result is slightly misleading, because a rather large number of results that include a name field comes from a single source, namely the Internet Movie Database (imdb.com). This is caused by the domain bias mentioned in 4.1. A total of

53,194 results, or 61.3% of all results including a name, come from the IMDB “On this day” pages, which list people that were born on a specific day. A random sample of results from imdb.com can be seen in Appendix C. A closer inspection of the other field in this sample set shows that each record contains not just one, but two names. It turns out IMDB abuses tables, as do so many pages on the web, for layout purposes. Here the table is not only a data table, but also serves to break the page up into two columns. Because IMDB does this with just one table – as opposed to nesting the real data table into another table that provided the columns – it is impossible for any automated agent to tell that this is happening. And so the two names that are seen in the same row of the table are captured together, even though they have nothing to do with each other whatsoever.

It should also be noted that “name” does not always denote the name of a person. There are many examples in the collected data where name refers to different things, including airline companies, Greek restaurants, and sports matches.

Another interesting thing to notice in the list of individual fields is that e-mail is relatively low. Only 14.8% of the results contains an e-mail address, where intuition says that the number of e-mail addresses in tables on the web should be much higher.

What I think we’re seeing here is the result of spam. Because so-called spam bots roam the web looking for e-mail addresses, many people have taken measures to prevent spam bots from finding their e-mail address. This includes using javascript to write the addresses to a page, and using alternative ways of writing them (such as “somebody (at) example (dot) com” instead of “somebody@example.com”). And what fools a spam bot, unfortunately also fools the DataSpider.

Since most fields are of little value by themselves, let’s look at some combination. The largest combined number of fields is four, with the name, e-mail, zipcode and phone fields all occurring. The total number of records where this combination occurs is five, which admittedly is not much. All those records are listed in the second sample in Appendix C, together with some records that contain only the name, zipcode and phone fields, which is a superset of that set. As you can see, three of the five results are from a list of names. These contain, including the data from the other field, a complete name, address, zipcode, phone and e-mail result, all from a single page. These results are the only three captured from that page, even though there are many more names listed in a similar fashion on that page. This is due to the zipcode inconsistency error described in section 3.3.

We also see a number of Greek restaurants, and some stores, complete with address and phone numbers. Unfortunately, here again a number of results from the source pages were not captured due to the zipcode inconsistency error.

Although it is a far too limited number of samples to draw any real conclusions from, I do believe that intuitively, what we see here is part of a trend. Most of the physical address data we’ve been able to gather belongs to retailers or firms, not to individuals. This is not surprising, since it seems that companies would be far more likely to disclose contact information on the web than individuals.

### 4.3 Integrating data

One of the goals in collecting data from the web is to try and integrate data from multiple sources. It remains nearly impossible to do, and when reading this section one must keep in mind the limitations of the approach.

DataSpider attempts to label data fields, which is a very complex problem. The labelling applied by DataSpider is only partly semantic; while it can tell that something is a date, it doesn’t know if it is dealing with a birth date, a graduation date, a sales date, etc. Nor can it tell if a phone

number is really a fax number, or whether a currency is the price of some product, a fine, or prize money, or whether a name is the name of person, place, object or event. The only one that's fairly unambiguous is e-mail address, although it remains difficult to determine what the e-mail address belongs to. There are quite a few examples where both the name and e-mail fields are present, but if we want to determine the owner of the address it now becomes subject to the same ambiguities as name. Does the address belong to a person, a company, a website as a whole? And if it's for some larger organization, are you dealing with a specific contact person within that organization, or a generic address for the entire organization or perhaps only one department? These are all questions that DataSpider cannot answer, and that are indeed nearly impossible to answer despite the research that has been done in this area.

Another problem is the values themselves. The values "Sven Groot", "S. Groot", "Mr. Groot", "Dhr. S. Groot" etc. might all refer to the same person. And similarly, two occurrences of the value "Sven Groot" might actually not refer to the same person at all.

Fortunately, some of the other fields are less susceptible from this. As long as we're not dealing with internal addresses (which is a reasonable assumption since the data comes from the world wide web, not from intranets), e-mail addresses are unique. Similarly, a zip code identifies a single place, and a date identifies a single day. Date does however have the problem of ambiguity between formats; "3-10-2005" might equally likely refer to October 3<sup>rd</sup> and March 10<sup>th</sup>. In some cases it might be possible to use other dates from the same page to resolve it; if the page also contains a date "27-8-2005", it is a likely assumption that all dates on that page follow that same format (in this case "dd-mm-yyyy"). DataSpider makes no attempt to disambiguate dates, and in fact simply stores them as plain text in the format they were found on the webpage. This means that if we want to do a join on dates, further processing is needed to normalize the date format found in the results.

In the following section, we will try to find multiple occurrences of the same field value to see how feasible it would be to join on a certain field, using the name field as an example.

#### 4.3.1 Results from names with multiple occurrences

A vast amount of names that occur multiple times in the database are actually names from, once again, IMDB. Even though this is in line with the fact that IMDB is by far the largest contributor to the name field in general, this does not automatically mean we would find the same name more than once. After all, the names come from the "On this day" page, and how likely is it that someone has more than one birthday? However, as noted before, IMDB puts two names on one row to give the page a two-column layout, and as it turns out two consecutive visits to the same page yielded different results. Under normal circumstances, two visits to the same page would be eliminated as duplicate, but in this case the urls of the pages differ, even though they are the same. IMDB uses both [www.imdb.com](http://www.imdb.com) and [us.imdb.com](http://us.imdb.com) to serve the same pages. Because these multiple occurrences are useless, we will consider them no further.

The most occurring "name" is the text "E-Mail Address" with which has 1112 unique occurrences in the results. This "name" is indicative of a big problem that our spider has currently, and that can be seen at many more places. In fact, this problem fills most of the top-ten most occurring names. As previously indicated, our spider can only deal with vertically laid out data tables, and these come from horizontal tables. In this case, not the first row, but the first column is the table header, and the data is laid out per column instead of per row. Because the DataSpider is not prepared for these cases, it will apply the table header heuristic to the first row, and encounters a cell named "Name". It then assumes that that column contains names, while in fact that row contains names. When looking at the next row, it sees the value "E-Mail Address" in

the same column as “Name”, so it assumes it’s a name and puts it in that field for the database. It’ll usually recognize the e-mail address on that row as well, with the DataTypeXml rule, but it will end up storing a row that relates the name “E-Mail Address” with some e-mail address, while the actual name belonging to that e-mail address is nowhere to be found in the database. It is therefore clear that any future attempts at building a spider must be prepared to deal with horizontal tables, although finding a good way to do this might prove difficult or processing intensive.

The runner up is “Tel. nummer” with 719 unique occurrences, but this result is not what it seems. At first glance, it would appear to be the same type of result as “E-Mail Address”, but it turns out all of these are from domains that redirect to a search page on [www.bisnis.nl](http://www.bisnis.nl), which does not contain the data our spider found. How this happened I cannot explain. There are several other values in there, including “KvK nummer” where the same thing happened.

We find a number of first-name only values at the top. These include “katja” (472 occurrences), “Jacqueline” (53 occurrences), “Ton” (30 occurrences) and others. In the database, we find that these names have been related with a date. It turns out that these are from a webforum. What we have collected are the names of the posters, combined with postdates. Not a whole lot can be said about that, except perhaps that katja seems to post rather much compared to the other people on that forum. Here it is indeed possible to join data at least from the same site. Using our database, we could easily answer the question how often katja posted on a certain day, something which traditional search engines would not be able to answer.

We also find a number of things that are obviously not names of people anyway. A number of items, such as “Jeugdwedstrijd”, “1stre klasse”, “2de klasse” and “Wisselbokaal 2005” are from a match schedule for, believe it or not, fierljeppen. Another one that’s hard to miss are the names of airline companies, such as “American Airlines Inc”, “Delta Air Lines Inc”. These comes from the website of the US Federal Aviation Administration (FAA), and are in fact from a list of sanctions taken against the named airlines regarding aviation safety. Samples from both of these can be seen in Appendix C. Take a look for instance at the first record from the airlines sample Although most of the data has been grouped in other, we can see that Northwest Airlines was fined \$6200 for a security violation committed on July 1<sup>st</sup> 1999.

None of the tested examples had the same name value actually occurring on different sites with usable results, which confirms the expected difficulty in actually doing these kinds of joins.

## 5 Conclusion

In this paper we have seen a brief overview of what a very limited search for data would yield. It is indeed possible to find data and categorize them by name, date and similar. We have also estimated in what quantity this data can be found. Although that amount was pretty low, one must keep in mind the extremely limited approach that was taken here; as was shown in the related work section, far more intelligent approaches to extracting data from web pages exist which would undoubtedly lead to better results, although most of these would probably prove too slow to actually apply to a similar experiment as was done here.

This and other approaches to extracting data from the existing web however show us the difficulty involved in truly understanding the data. Some of the examples here have shown that something as simple as a name might have very different meanings, such as the name of a person, a company or a tournament. The ultimate goal of being able to correctly join data from different sources is difficult and far off indeed.

The data is there, but unless the semantic web makes a major breakthrough in popularity, it is my opinion that we are a long way off from making sense of it all.

## 6 References

- [1] Sergey Brin and Lawrence Page. 1998. *The Anatomy of a Large-Scale Hypertextual Search Engine*. Computer Science Department Stanford University.
- [2] Eugene Agichtein, Steve Lawrence, and Luis Gravano. 2002. *Learning to Find Answers to Questions on the Web*. ACM Transactions on Internet Technology.
- [3] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. 1997. *To Weave the Web*. Proceedings of the 23rd VLDB Conference
- [4] G. Mecca P. Atzeni, A. Masci, P. Merialdo, and G. Sindoni. 1998. *The ARANEUS Web-Base Management System*. SIGMOD '98
- [5] Giansalvatore Mecca, Alberto O. Mendelzon, and Paolo Merialdo. 2002. *Efficient Queries over Web Views*. IEEE Transactions On Knowledge And Data Engineering.
- [6] Zehua Liu, Wee Keong Ng, Ee-Peng Lim, and Feifei Li. 2004. *Towards building logical views of websites*. Data & Knowledge Engineering
- [7] Arvind Arasu, Hector Garcia-Molina. 2003. *Extracting structured data from Web page.*, Proceedings of the 2003 ACM SIGMOD international conference on on Management of data
- [8] Guizhen Yang, I. V. Ramakrishnan, and Michael Kifer. 2003. *On the complexity of schema inference from web pages in the presence of nullable data attributes*. Proceedings of the twelfth international conference on Information and knowledge management.
- [9] Kristina Lerman, Lise Getoor, Steven Minton, and Craig Knoblock. 2004. *Using the structure of Web sites for automatic segmentation of tables*. Proceedings of the 2004 ACM SIGMOD international conference on Management of data
- [10] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, and Juliana S. Teixeira. 2002. *A brief survey of web data extraction tools*. ACM SIGMOD Record
- [11] Ruth Yuee Zhang, Laks V. S. Lakshmanan, and Ruben H. Zamar. 2004. *Extracting relational data from HTML repositories*. ACM SIGKDD Explorations Newsletter
- [12] D. Buttler, L. Liu, and C. Pu. 2001. *A fully automated object extraction system for the World Wide Web*. IEEE ICDCS-21
- [13] C. Chang, and S-L. Lui. 2001. *IEPAD: Information extraction based on pattern discovery*. WWW-10.
- [14] B. Liu, R. Grossman, R. and Y. Zhai. 2003. *Mining data records from Web pages*. KDD-03

- [15] Yanhong Zhai, and Bing Liu. 2005. *Web data extraction based on partial tree alignment*. Proceedings of the 14th international conference on World Wide Web
- [16] J.-Y. Wang, and F. Lochovsky. 2003. *Data extraction and label assignment for Web databases*. WWW-03
- [17] Frank Manola, Eric Miller and Brian McBride. 2004. *RDF Primer*. W3C Recommendation
- [18] Dave Beckett and Brian McBride. 2004. *RDF/XML Syntax Specification (Revised)*. W3C Recommendation
- [19] Dan Brickley, R.V. Guha, and Brian McBride. 2004. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation.
- [20] Deborah L. McGuinness, Frank van Harmelen. 2004. *OWL Web Ontology Language Overview*. W3C Recommendation.
- [21] Tim Berners-Lee. 1998. *Web design issues; What a semantic web is not*.
- [22] Albert-László Barabási. 2001. *The physics of the web*. Physics World July 2001

## Appendix A. Breakdown by used fields

Note: field combinations that didn't occur at all have been removed from the list.

1	name	86818	55,51%
1	currency	45469	29,07%
1	date	40216	25,71%
1	email	23150	14,80%
1	phone	6850	4,38%
1	zipcode	2356	1,51%
2	name date	17515	11,20%
2	name email	9066	5,80%
2	currency date	8081	5,17%
2	email date	4681	2,99%
2	name phone	3928	2,51%
2	email phone	2074	1,33%
2	name currency	1986	1,27%
2	name zipcode	1231	0,79%
2	zipcode phone	745	0,48%
2	phone date	471	0,30%
2	zipcode date	375	0,24%
2	zipcode currency	96	0,06%
2	email zipcode	66	0,04%
2	phone currency	42	0,03%
2	email currency	37	0,02%
3	name currency date	713	0,46%
3	name email phone	582	0,37%
3	name zipcode phone	281	0,18%
3	name zipcode date	22	0,01%
3	name email date	15	0,01%
3	name email zipcode	5	0%
3	email zipcode phone	5	0%
3	email currency date	5	0%
3	name phone date	3	0%
3	phone currency date	1	0%
4	name email zipcode phone	5	0%

## Appendix B. Top most occurring names

The number of unique occurrences is determined by eliminating those rows where not only the name, but all other fields (excluding the url) are equal as well.

Name	Occurrences	Unique occurrences
E-Mail Address	1133	1112
Tel. nummer	1247	719
E-mail	894	637
Anonymous	650	613
katja	639	472
Email adres	697	385
KvK nummer	572	327
Fax nummer	523	306
Hurricane #2	76	76
Hurricane #3	72	71
Hurricane #4	65	65
Hurricane #1	53	53
Jacqueline	55	53
Tropical Storm #1	53	52
Hurricane #5	50	50
2de klasse	70	44
Hermien	43	43
1ste klasse	64	41
Georgios	41	41
Hurricane #6	41	41
Jeugdwedstrijd	83	40
AMERICAN AIRLINES INC	37	37
Tropical Storm #5	35	35
Jet	33	33
Contactinfo	34	31
Tropical Storm #7	31	31
Unknown	35	31
Ton	43	30
Tropical Storm #6	30	30
Tropical Storm #4	29	29
Hurricane #8	27	27
pstlgeo	27	27
Tropical Storm #3	27	27
DELTA AIR LINES INC	26	26
Ria O	26	26
Hurricane #7	25	25
Natuurwinkel	25	25
paula	27	25
Tropical Storm #2	24	24
Webmaster	25	24
email	22	22
UNITED AIR LINES INC	22	22



## Appendix C. Result samples

Note: if a field is null in all listed samples, it will be omitted to improve readability of the samples.

Sample 1 Various results from imdb.com

uri	name	other
<a href="http://www.imdb.com/OnThisDay?day=18%26month=July">http://www.imdb.com/OnThisDay?day=18%26month=July</a>	Glenn Hughes (I) (54)	1950;(name)Rose Michtom (107)/1897;
<a href="http://www.imdb.com/OnThisDay?day=8%26month=January">http://www.imdb.com/OnThisDay?day=8%26month=January</a>	Canon M. Ramm	2002;(name)Keith Lockett/1973;
<a href="http://www.imdb.com/OnThisDay?day=18%26month=January">http://www.imdb.com/OnThisDay?day=18%26month=January</a>	Dusan Karuovic (51)	1997;(name)Reese 'Goose' Tatum (45)/1967;
<a href="http://www.imdb.com/OnThisDay?day=21%26month=January">http://www.imdb.com/OnThisDay?day=21%26month=January</a>	Karina Bakker (23)	1982;(name)Raita Ryu (65)/1940;
<a href="http://www.imdb.com/OnThisDay?day=25%26month=January">http://www.imdb.com/OnThisDay?day=25%26month=January</a>	Herman Wedemeyer (74)	1999;(name)Sherwood MacDonald (87)/1968;
<a href="http://us.imdb.com/OnThisDay?day=1&amp;month=January">http://us.imdb.com/OnThisDay?day=1&amp;month=January</a>	Felix Bergsson (38)	1967;(name)Eric Winstone (90)/1915;

Sample 2 Results that include the fields name, zipcode and phone

uri	name	email	zipcode	phone	other
<a href="http://www.nijmegen.nl/bestuurorganisatie/Bestuur/Gemeenteraad/adressen.asp?ComponentID=22717&amp;SourcePageID=6592">http://www.nijmegen.nl/bestuurorganisatie/Bestuur/Gemeenteraad/adressen.asp?ComponentID=22717&amp;SourcePageID=6592</a>	Mevr. S.R. Abdoelbasier	s.abdoelbasier@chello.nl	6521 EJ	06-12098173	Bachstraat 2;
<a href="http://www.nijmegen.nl/bestuurorganisatie/Bestuur/Gemeenteraad/adressen.asp?ComponentID=22717&amp;SourcePageID=6592">http://www.nijmegen.nl/bestuurorganisatie/Bestuur/Gemeenteraad/adressen.asp?ComponentID=22717&amp;SourcePageID=6592</a>	Dhr. B. Groothuis	bart_groothuis@yahoo.com	6521 CD	06-21552332	Coehoornstraat 93;
<a href="http://www.nijmegen.nl/bestuurorganisatie/Bestuur/Gemeenteraad/adressen.asp?ComponentID=22717&amp;SourcePageID=6592">http://www.nijmegen.nl/bestuurorganisatie/Bestuur/Gemeenteraad/adressen.asp?ComponentID=22717&amp;SourcePageID=6592</a>	Dhr. J.V.J. van Deurzen	Jvdeurzen@hetnet.nl	6511 ZA	06-2022253	Kromme Elleboog 2;
<a href="http://www.grieksegids.nl/hornrestaurants.htm">http://www.grieksegids.nl/hornrestaurants.htm</a>	TAVERNA CYPRUS	info@tavernacyprus.nl	1621 CV	0229-215547	RODE STEEN;3;
<a href="http://www.grieksegids.nl/hornrestaurants.htm">http://www.grieksegids.nl/hornrestaurants.htm</a>	GRIEKS EETCAFE PLAKA	tanjakappos@hotmail.com	1621 CV	0229-266645	ROODE STEEN;2;
<a href="http://www.grieksegids.nl/dordrechtrestaurants.htm">http://www.grieksegids.nl/dordrechtrestaurants.htm</a>	De Grote Griek		3311 GN	078-6489495	Spuiboulevard;95;
<a href="http://www.grieksegids.nl/dordrechtrestaurants.htm">http://www.grieksegids.nl/dordrechtrestaurants.htm</a>	OLYMPIADA		3311KW	(078)6312323	VISBRUG;2;;
<a href="http://www.nieuweband.nl/winkel.htm">http://www.nieuweband.nl/winkel.htm</a>	Weegschaal		1011	020-	Jodenbr

			NK	6241765	eestraat 20;Amsterdam;
<a href="http://www.nieuweband.nl/winkel.htm">http://www.nieuweband.nl/winkel.htm</a>	Natuurwinkel		1013 JK	020-6266310	Haarlemmerdijk 168;Amsterdam;
<a href="http://www.nieuweband.nl/winkel.htm">http://www.nieuweband.nl/winkel.htm</a>	Belly, de		1015 HG	020-3309484	Nieuwe Leliestr. 174;Amsterdam;

Sample 3 A match schedule for fierljeppen

uri	name	date	other
<a href="http://www.pbholland.com/kalender.php?&amp;lang=nl&amp;context=PBH">http://www.pbholland.com/kalender.php?&amp;lang=nl&amp;context=PBH</a>	Seizoensopening 2005 Van der Woude Auto's Bokaal	za 14 mei 2005	S;14:00;Linschoten;>>;
<a href="http://www.pbholland.com/kalender.php?&amp;lang=nl&amp;context=PBH">http://www.pbholland.com/kalender.php?&amp;lang=nl&amp;context=PBH</a>	Jeugdwedstrijd	za 14 mei 2005	14:00;Linschoten;>>;
<a href="http://www.pbholland.com/kalender.php?&amp;lang=nl&amp;context=PBH">http://www.pbholland.com/kalender.php?&amp;lang=nl&amp;context=PBH</a>	Wisselbokaal 2005	za 21 mei 2005	14:00;Polsbroekerdam;>>;
<a href="http://www.pbholland.com/kalender.php?&amp;lang=nl&amp;context=PBH">http://www.pbholland.com/kalender.php?&amp;lang=nl&amp;context=PBH</a>	Jeugdwedstrijd	za 21 mei 2005	14:00;Polsbroekerdam;>>;
<a href="http://www.pbholland.com/kalender.php?&amp;lang=nl&amp;context=PBH">http://www.pbholland.com/kalender.php?&amp;lang=nl&amp;context=PBH</a>	Vlisterbokaal	za 28 mei 2005	14:00;Linschoten;>>;
<a href="http://www.pbholland.com/kalender.php?&amp;lang=nl&amp;context=PBH">http://www.pbholland.com/kalender.php?&amp;lang=nl&amp;context=PBH</a>	Jeugdwedstrijd	za 28 mei 2005	14:00;Vlist;>>;

Sample 4 Airlines fined by the FAA

uri	name	date	other
<a href="http://www.faa.gov/agc/enforcement/2000qtr3.htm">http://www.faa.gov/agc/enforcement/2000qtr3.htm</a>	NORTHWEST AIRLINES INC	7/1/99	1999GL720192;A/C or COMM OPER;ORD ASSESS CIVIL PENALTY;6200;DOLLARS;SECURITY;7/5/00;
<a href="http://www.faa.gov/agc/enforcement/2000qtr3.htm">http://www.faa.gov/agc/enforcement/2000qtr3.htm</a>	AMERICA WEST AIRLINES INC	9/26/99	1999GL720253;A/C or COMM OPER;ORD ASSESS CIVIL PENALTY;6500;DOLLARS;SECURITY;7/5/00;
<a href="http://www.faa.gov/agc/enforcement/2000qtr3.htm">http://www.faa.gov/agc/enforcement/2000qtr3.htm</a>	AMERICAN AIRLINES INC	10/23/99	2000CE720003;A/C or COMM OPER;ORD ASSESS CIVIL PENALTY;5000;DOLLARS;SECURITY;7/5/00;
<a href="http://www.faa.gov/agc/enforcement/2000qtr3.htm">http://www.faa.gov/agc/enforcement/2000qtr3.htm</a>	NORTHWEST AIRLINES INC	10/15/99	2000GL730008;A/C or COMM OPER;ORD ASSESS CIVIL PENALTY;7000;DOLLARS;SECURITY;7/5/00;

http://www.faa.gov/agc/enforcement/2000qtr3.htm	AMERICAN AIRLINES INC	12/16/98	1999CE720010;A/C or COMM OPER;ORD ASSESS CIVIL PENALTY;4000;DOLLARS;SECURITY;7/7/00;
-------------------------------------------------	-----------------------	----------	--------------------------------------------------------------------------------------

## Appendix D. Statistical breakdown of the search-based experiment

Pages visited	Domains visited	Pages with data
5006	751	8
5008	265	7
5025	46	1
5007	749	62
5013	501	8
5008	640	23
5012	66	0
5007	260	10
5009	44	1
5006	122	5
5007	1892	0
5009	8	0
5006	879	7
5009	412	11
5004	72	0
5009	340	13
5016	766	21
5009	432	4
5022	146	1
5024	150	21
5016	667	11
5023	431	1
5020	709	16
5003	1054	17
5440	286	5
5007	83	52
5010	731	6
5005	817	55
5008	589	5
5019	1183	13
5011	234	6
5006	159	3
5008	1111	12
5008	96	2
5037	387	4
5006	218	8
5008	295	3
5008	787	4

5184	195	1
5015	219	1
	Sample mean:	10,7
	Standard deviation:	14,57641228
	Standard error:	2,304733145