A WaCky Introduction

Silvia Bernardini, Marco Baroni and Stefan Evert

1 The corpus and the Web

We use the Web today for a myriad purposes, from buying a plane ticket to browsing an ancient manuscript, from looking up a recipe to watching a TV program. And more. Besides these "proper" uses, there are also less obvious, more indirect ways of exploiting the potential of the Web. For language researchers, the Web is also an enormous collection of (mainly) textual materials which make it possible, for the first time ever, to study innumerable instances of language performance, produced by different individuals in a variety of settings for a host of purposes.

One of the tenets of corpus linguistics is the requirement to observe language as it is produced in authentic settings, for authentic purposes, by speakers and writers whose aim is not to display their language competence, but rather to achieve some objective through language. To study "purposeful language behavior", corpus linguists require collections of authentic texts (spoken and/or written). It is therefore not surprising that many (corpus) linguists have recently turned to the World Wide Web as the richest and most easily accessible source of language material available. At the same time, for language technologists, who have been arguing for long that "more data is better data", the WWW is a virtually unlimited source of "more data". The potential uses to which the Web has been (or can be) put within the field of language studies are numerous and varied, from checking word frequencies using Google counts to constructing general or specialized corpora

of Web-published texts. The expression "Web as corpus" is nowadays often used to refer to these different ways of exploiting the WWW for language studies.

In what follows we briefly consider four different ways of using the Web as a corpus, focusing particularly on those taking the lion share of this volume of working papers: the Web as a "corpus shop", and the "mega-corpus/mini-Web" as a new object. The latter in particular will be described in some detail, and special attention will be paid to the design of this resource and the challenges posed by its development.

2 *Web as Corpus (WaC)*: four senses

There is currently no unified understanding of the expression Web as corpus. We have identified four separate senses, though there are probably others:

1. The *Web as a corpus surrogate*
2. The *Web as a corpus shop*
3. The *Web as corpus proper*
4. The *mega-corpus/mini-Web*

Researchers (and users in general) using the Web as a corpus *surrogate* turn to it via a standard commercial search engine for opportunistic reasons. They would probably use a corpus, possibly through corpus analysis software, but none exists for their purposes (e.g., because available corpora are too small), or they do not have access to one, or they do not know what a corpus is. The translator trainees at the School for Interpreters and Translators, University of Bologna (Italy), for instance, use the Web as a reference tool in their translation tasks, though the search

is often time consuming, the relevance and authoritativeness of the solutions found is hard to assess, and the observation of recurrent patterns very difficult. It would make sense for them to use a corpus, if one existed as large as the Web, and if they knew how to use it.[1] Similarly, researchers who rely on Google-like hit counts for their studies (e.g., Chklovski and Pantel 2004) live with the brittleness[2] and reduced flexibility of the search engine, though they would no doubt prefer a more stable resource, allowing replication and providing facilities for more sophisticated queries. Linguist-oriented meta-search engines like KWiCFinder[3] and WebCorp[4] wrap around the standard output of Web search engines some of the features and facilities of corpus search engines (e.g., the KWIC format, a collocation tool, and so forth). Though this solution leaves questions linked to the datasets and retrieval strategies untouched, users can to some extent pretend to be consulting the Web in a corpus-like environment.

Others using the Web as a corpus treat it as a corpus *shop*. They query a traditional search engine for combinations of search words, taking advantage of the facilities offered by the engine (e.g., selection of language, provenance, URL-type etc.) to focus their queries. They (select and) download the texts retrieved by the engine, thus creating a corpus in the traditional sense of the term. This procedure, which can be automatized to various degrees, is adopted by those who require specialized corpora, e.g., for translation, terminology or text analysis purposes. Several researchers have discussed the didactic advantages of "disposable" corpora (e.g., Varantola 2003) in the teaching of foreign languages and translation skills. Castagnoli (this volume) describes a classroom

[1] Unless stated otherwise, by "Web" and "corpus" we refer to both the text materials and the search engines used to index and search them.

[2] `http://aixtal.blogspot.com/2005/02/web-googles-missing-pages-mystery.html`

[3] `http://miniappolis.com/KWiCFinder/KWiCFinderHome.html`

[4] `http://www.webcorp.org.uk/`

experience where learners a) use the BootCaT toolkit (Baroni and Bernardini 2004), a set of Unix tools, to construct corpora from specialized domains in a semi-automated way, b) evaluate the corpora and c) use them to investigate terminology and retrieve typical instances of usage in context. Castagnoli suggests that the limits of this automatic procedure can be turned into advantages in a pedagogic context, where learners can be made to reflect on their text selection strategies and documentation skills. The development of a Web interface for the BootCaT tools (Baroni et al. 2006) should remove the technical hurdles for less computer-literate users and favor a more widespread use in the classroom and among language professionals.

Fantinuoli (this volume), Sharoff (this volume) and Ueyama (this volume) also use the BootCaT tools, but take a more descriptively oriented perspective. Their aim is an evaluation of corpora constructed semi-automatically from the Web. While Fantinuoli (like Castagnoli) focuses on the construction of specialized corpora for language professionals, Sharoff uses this methodology to build general language corpora of Chinese, English, German and Russian, and Ueyama of Japanese. These authors focus on different ways of evaluating their products: the comparison between manually and automatically constructed corpora and manually and automatically extracted terms in the case of Fantinuoli, the qualitative and quantitative observation of topics, genres and lexical items in Web corpora built in different ways in Ueyama, and the comparison of word frequency lists derived from the Web and from traditional corpora in Sharoff. These articles contribute to the establishment of good practices and open the way to the empirical study of a set of still under-investigated questions such as: In what ways can we say that traditional corpora differ from Web-derived corpora? How does the corpus construction methodology affect the nature of the resulting corpus?

So far we have been concerned with ways of using the Web opportunistically, to derive generalizations about (subsets of) a language either directly through search engine queries or indirectly through the downloading of Web published texts. For these purposes paper texts would equally be appropriate, if not for the obstacle of digitizing them. Our third possible meaning of the notion of *Web as corpus*, the "Web as corpus proper", is different inasmuch as it purports to investigate the nature of the Web. In the same way as the British National Corpus aims to represent the whole of British English at a given point in time, it is possible to envisage a corpus that represents Web English at a given point in time.[5] This research paradigm could tell us something about the language used on the Web (glimpses of this are provided in this volume by Ueyama and Sharoff). Clearly, extensive discussion and experimentation is needed to develop criteria for Web sampling. Input might come from taxonomy-oriented surveys (along the lines of the article by Mehler and Gleim in this volume). We expect this area of research to feature prominently in WaC linguistics in the next few years.

Lastly, our fourth and most radical way of understanding the expression *Web as a corpus* refers to attempts to create a new object, a sort of mini-Web (or mega-corpus) adapted to language research. This object would possess both Web-derived and corpus-derived features. Like the Web, it would be very large, (relatively) up-to-date, it would contain text material from crawled Web sites

[5]One could argue that this sense of Web as corpus is somewhat different from those discussed so far (and indeed from the one discussed below). After all, the *corpus surrogate* and the *corpus shop* approaches are different ways of using Web data for similar purposes (to investigate linguistic issues), whereas in the WaC proper approach the purposes of the investigation differ (i.e., we are trying to learn about the Web, rather than using the Web to learn about language). We include this sense here anyway because the aim is simply telling different understandings of the expression apart, rather than providing a consistent classification.

and it would provide a fast Web-based interface to access the data. Like a corpus, it would be annotated (e.g., with POS and lemma information), it would allow sophisticated queries, and would be (relatively) stable. Both people wanting to investigate aspects of language through the Web, and people wanting to investigate aspects of the Web through language could profit from this corpus.

We are convinced that this is a valuable research project because it answers a widely-felt need in our community of (computational) linguists, language and translation teachers and language professionals for a resource that combines the reliability and the flexibility of corpora and their search tools with the size, variety and timeliness of the Web. The chances that commercial Web search engines be interested in such a research agenda are very low, and relying on the less standard facilities they offer may not be a good idea in the long run.[6]

Besides being valuable, we believe that this is also a feasible, though challenging, endeavor. The present authors and several contributors to this volume are currently involved in the piloting of very large Web-derived corpora in English, German and Italian, in a project (started at the end of 2004) that emphasizes the development and sharing of open tools and resources. A series of workshops have been organized which have provided a public discussion space (the Web as corpus workshop in Forlì, January 14, 2005; the Web as corpus workshop at CL05 in Birmingham, July 14, 2005; and the Web as corpus workshop at EACL, Trento, April 3, 2006). Discussion is constantly taking place also through the project wiki, the so-called *WaCky wiki*.[7] Many WaCky contributors are actively involved in the recently established Special Interest Group on Web as Corpus (SIGWAC) of the Association for

[6] AltaVista discontinued the NEAR operator in 2004. The Google API keys (on which the BootCaT tools currently rely) have provided very discontinuous functionality during tests carried out in the last few months of 2005 and in early 2006.

[7] `http://wacky.sslmit.unibo.it/`

Computational Linguistics (ACL).[8] At the same time, infrastructure building has also started in Forlì, with the aim to secure the minimum technical prerequisites to begin the piloting phase. Two mega corpora are at an advanced stage of development (*deWaC* and *itWaC*, for the German and Italian languages, respectively: Baroni and Kilgarriff 2006; Baroni and Ueyama 2006), the construction of other corpora is under way for other languages (English, Chinese and Russian), and more funds to proceed with the project are being sought.

Among the papers in this collection, the one by Emerson and O'Neil presents in detail the first steps of data collection for the purposes of building a mega corpus of Chinese. There are two main sides that are relevant to the construction of a mega corpus/mini-Web. First, one has to retrieve, process and annotate Web data. Second, one has to index these data, and construct an interface to allow prospective users to access the data. In the next two sections, we will present our ideas about both aspects of the process, relying on our experiences with deWaC and itWaC and on the work reported in the remainder of this volume.

3 Constructing Web corpora

The basic steps to construct a Web corpus are:

1. Select the "seed" URLs
2. Retrieve pages by crawling
3. Clean up the data
4. Annotate the data

We discuss each of these in turn.

[8] `http://www.sigwac.org.uk/`

3.1 Selecting seed URLs

The crawl has to start from a set of seed URLs. For special-purpose corpora, it is relatively straightforward to decide the seeds (e.g., if one wants to build a corpus of blogs, one can select a random set of blog URLs from one or more blog servers). For a general-purpose corpus, one would ideally want to draw a random sample of pages that are representative of the target language. As discussed in the article by Ciaramita and Baroni, this is not the same as drawing a random sample of webpages. For example, suppose that the Italian Web is composed of a 90% of pornographic pages, 9% of Linux *howtos*, and that all other text types together make up just 1% of the whole. For the purpose of building a corpus, we would probably prefer a sampling method heavily biased in favor of selecting from this 1%, rather than a true random sample that would lead to a corpus of mostly pornography plus the occasional bash shell guide. The fact that the notion of "representativeness" of a corpus (and how to measure it) is far from well-understood (Kilgarriff and Grefenstette 2003) complicates matters further. Ciaramita and Baroni propose a measure of "unbiasedness" of a Web-derived corpus based on the comparison of the word frequency distribution of the target corpus to those of deliberately biased corpora.

Both Sharoff and Ueyama select seed URLs by issuing (automated) queries for random content word combinations to Google, and retrieving the URL lists returned by the engine. The qualitative evaluation carried out by Sharoff suggests that the variety (in terms of parameters such as genre and domain) of the collections of documents corresponding to these URLs is closer to what we would expect from a balanced corpus than to what we find in biased collections, such as newswire corpora. An important aspect of this methodology is how the words used in the queries are selected. Ueyama's experiments suggest that selecting words from traditional corpora might bias the queries towards pages containing

higher register prose and "public life" domains, thus missing some of the most interesting linguistic material available on the Web – non-professionally written, informal prose on everyday topics. Pages of this sort can be found using words from basic vocabulary lists. The seed URLs chosen to build the WaCky initiative German and Italian corpora were retrieved from Google with combinations of words extracted both from traditional newspaper corpora and from "basic vocabulary" lists for language learners, in the hope to tap into both higher register/public and lower register/private sections of the Web.

Emerson and O'Neil select URLs matching their target language (Chinese) from the Open Directory Project (ODP),[9] a large, open directory of webpages maintained by volunteers. This method has several advantages over the former: It does not rely on a commercial enterprise such as Google, and the metadata information provided by ODP can be exploited for sampling. On the negative side, the set of URLs listed by ODP is much smaller than the set indexed by Google (at the moment of writing, about 5 million vs. 8 billion). Moreover, ODP seems biased in favor of top level pages, whereas the pages retrieved by random content word queries to Google often come from deeper layers of websites, and as such tend to be characterized by richer textual content. Devising and comparing seed URL selection strategies will be an important area for future WaC research.

3.2 Crawling

If the list of seed URLs is long and/or one does not aim to build a very large corpus, crawling can be as simple as retrieving the documents corresponding to the seed URLs (this is what Sharoff and Ueyama do). Otherwise, one uses the seed URLs to start a crawl of the Web, i.e., a program is launched that retrieves pages

[9] `http://www.dmoz.org`

corresponding to the seed URLs, extracts new URLs from the links in the retrieved pages, follows the new links to retrieve more pages, and so on. Conceptually, crawling is a straightforward procedure; however, only a sophisticated implementation of the procedure will allow one to perform a successful large-scale crawl. There are several issues that need to be addressed.

- *Efficiency*: As more pages are retrieved, the queue of discovered URLs grows very large. Thus, the crawler must be able to manage such a large list in a memory-efficient way.
- *Duplicates*: The crawler must make sure that only URLs that have not been seen already are added to the list.
- *Politeness*: Minimally, the crawler must respect the directives specified by webmasters in a site's `robots.txt` file. However, it should also avoid hammering the same site with thousands of requests in a short time span, and provide an easy way to contact the owner of the crawl.
- *Traps*: The crawler should avoid "spider traps", i.e., malicious sites that try to stop it, e.g., by luring it into a loop in which it will continue downloading dynamically generated pages with random text forever (not a good thing for corpus building!)
- *Customization*: The crawler should be easy to customize (e.g., for a linguistic crawl one might want to limit the crawl to pages from a certain national domain, and focus on HTML documents) and, given that a large crawl will probably take weeks to complete, it should be possible to monitor an ongoing crawl, possibly changing parameters on the fly.
- *File handling*: Finally, given that a large crawl will retrieve millions of documents, the crawler should handle the retrieved data in an intelligent manner (on the one hand, we

would not want to have to deal with millions of output files; on the other, a single file of a few hundreds gigabytes would also be hard to manage).

For all these reasons, simple tools such as the Unix utility `wget` are not appropriate for large-scale crawls, and programs specifically designed for such task should be used. The crawl described in Emerson and O'Neil's article is based on one such tools, i.e., Heritrix, the open-source Java crawler developed at the Internet Archive.[10] Heritrix is also employed by the WaCky project.[11]

3.3 Data cleaning

Once the crawl is over, we are left with a (possibly very large) set of HTML documents,[12] and we have to convert them into something that can be used as a linguistic corpus. For many purposes, HTML code and other non-linguistic material should be removed. Presumably, language/encoding detection and (near-)duplicate discarding are desirable steps independently of the purposes of the corpus.

An interesting side effect of WaC activities is that, because Web data are so noisy, data cleaning must take center stage, unlike in traditional NLP, where it has been seen as a minor preprocessing step that is not really worth talking about (standard introductions to NLP, such as Jurafsky and Martin 2000 and Manning and Schütze 1999, do not dedicate any space to the topic). In-

[10] `http://crawler.archive.org`

[11] An alternative fully featured crawler is the Ubicrawler (`http://ubi.imc.pi.cnr.it/projects/ubicrawler`), which, however, at the moment of writing does not appear to be publicly distributed under a GNU-like license, and, consequently, is not supported by the same kind of wide community that supports Heritrix.

[12] Other formats, such as Acrobat's PDF and Microsoft's doc might also be converted to text and added to the corpus. We do not discuss this possibility here.

deed, the Special Interest Group on Web as Corpus of ACL is currently preparing a competitive data cleaning task, CLEANEVAL, as its first public activity.[13]

3.3.1 HTML code removal and boilerplate stripping

Tools such as `vilistextum`[14] (used by Emerson and O'Neil) and the standard Unix textual browser `lynx` (used by Sharoff) extract plain text from an HTML document, while attempting to preserve its logical structure and the hyperlink information. This is appropriate for certain purposes – e.g., to parse a document according to the "graph grammar" of webpages proposed in Mehler and Gleim's chapter (indeed, for such purposes it might be desirable to preserve the HTML code itself). Logical structure and hyperlink information might also be useful for purposes of document categorization. However, structural markup and links will constitute noise for the purposes of further linguistic processing (tokenization, POS tagging, etc.).

Equally problematic, in terms of linguistic processing and extraction of linguistic information, is the presence of "boilerplate", i.e., the linguistically uninteresting material repeated across the pages of a site and typically machine-generated, such as navigation information, copyright notices, advertisement, etc. Boilerplate can clutter KWIC displays, distort statistics and linguistic generalizations (we probably do not want "click here" to come up as the most frequent bigram of English), and make duplicate detection harder. Boilerplate is harder to identify than HTML/javascript, since it is regular text, not overtly delimited code. For corpora based on crawls of a limited number of domains, it might be possible to analyze pages from the domains and manually develop regular expressions to spot and remove boilerplate. For larger crawls,

[13] `http://cleaneval.sigwac.org.uk/`
[14] `http://bhaak.dyndns.org/vilistextum`

domain-independent methods must be applied. For the development of the WaCky corpora, we used HTML tag density as a fast heuristic method to filter out boilerplate (re-implementing the algorithm of the Hyppia project BTE tool).[15] The idea is that the content-rich section of a page will have a low HTML tag density, whereas boilerplate text tends to be accompanied by a wealth of HTML (because of special formatting, many newlines, many links, etc.) Thus, of all possible spans of text in a document, we pick the one for which the quantity $\text{Count}(tokens) - \text{Count}(tags)$ takes the highest value.

If we are interested in the Web as a source of linguistic samples, boilerplate stripping is fundamental. If we are studying the make-up of HTML documents and their linking structures, boilerplate stripping might be undesirable, as it might destroy the logical structure of a document. Optimally, a Web-based corpus should satisfy both needs by providing access to the original, unprocessed HTML documents as well as to a linguistically annotated version that had code and boilerplate removed.

3.3.2 Language/encoding detection

For Western European languages, language filtering can be a simple matter of discarding documents that do not contain enough words from a short list of function words (this is the strategy we employed when building the German and Italian WaCky corpora). For other languages, encoding detection must be performed together with language filtering, since webpages in the same languages can come in a number of different encodings. Free tools such as the TextCat utility[16] or the proprietary tools used by Emerson and O'Neil can perform this task. Tools such as the `recode` utility[17] can then convert all pages to the same encod-

[15] `http://www.smi.ucd.ie/hyppia/`

[16] `http://odur.let.rug.nl/~vannoord/TextCat`

[17] `http://recode.progiciels-bpi.ca/`

ing for further processing. Language detection will typically work poorly if the HTML code has not been stripped off. Moreover, if the detection algorithm relies on statistical models extracted from training data (often, character n-grams), these training data should not be too dissimilar from the Web data to be analyzed. For example, when using TextCat on German Web data, we noticed that the tool systematically failed to recognize German pages in which nouns are not capitalized – an informal way of spelling that is common on the Web, but virtually unattested in more standard sources such as newspaper text. A more difficult issue is that of dealing with pages that contain more than one language – however, given the wealth of data available from the Web, it might be sufficient to simply discard such pages (assuming they can be identified). Lastly, word lists can be used to identify and discard "bad" documents in the target language (e.g., pornography and Web-spam).

3.3.3 (Near-)duplicate detection

Identical pages are easy to identify with fingerprinting techniques. However, crawl data will typically also contain *near duplicates*, i.e., documents that differ only in trivial details, such as a date or a header (e.g., the same tutorial posted on two sites with different site-specific material at the beginning and/or end of the document). In principle, near duplicates can be spotted by extracting all n-grams of a certain length (e.g., 5-grams) from each document and looking for documents that share a conspicuous amount of such n-grams. However, for large corpora a procedure of this sort will be extremely memory- and time-consuming. Standard methods have been developed within the WWW-IR community (see, e.g., Broder et al. 1997 and Chakrabarti 2002) to obtain an estimate of the overlap between documents on the basis of random selections of n-grams in a very efficient manner. These techniques can also be used to find near duplicates in linguistic Web cor-

pora (a simplified version of Broder's method has been used in the clean-up of the WaCky corpora).

Notice that near duplicate spotting will work better if boilerplate stripping has been performed, as boilerplate is a source of false positives (documents that look like near duplicates because they share a lot of boilerplate) as well as false negatives (near duplicates that do not look as similar as they should because they contain different boilerplate). A more delicate issue concerns document-internal duplicate detection, e.g., pages downloaded from a bulletin board that contain a question and several follow-up postings with the question pasted into the replies. Not only can this sort of duplication be hard to spot, but its removal might disrupt the discourse flow of the document. Indeed, one might wonder whether removal of document-internal duplication is a theoretically justified move.[18]

3.4 Annotation

Tokenization, POS annotation and lemmatization of a Web corpus that has undergone thorough clean-up are straightforward operations. However, one has to be careful about the peculiarities of Web language, such as smileys, non-standard spelling, high density of neologisms, acronyms, etc. Ideally, tokenizing rules should take these aspects into account, and POS taggers should be re-trained on Web data.

The diversified, ramshackle nature of a crawled Web corpus means metadata are at the same time sorely needed (who is the author of a page? is the author a native speaker? what is the page about?) and difficult to add, both for technical reasons (the sheer size of the corpus) and because the Web presents a wealth of new "genres" and peculiar domains (how do you classify a page

[18]Indeed, if the corpus is seen as a random sample of the Web, any form of (near-)duplicate removal becomes a theoretically dubious move.

about 9/11 written by a religious fanatic that is half blog and half advertisement for his book?)

The articles of Sharoff and Ueyama in this volume report manual classification of samples of Web corpus documents in terms of genre, domain and other parameters. It is clear from these preliminary investigations that the categories used to classify traditional corpora, often based on library classification systems, have to be extended and revised in order to account for Web data. At the very least, the sizable proportion of "personal life" domains and genres present on the Web requires a fine grained taxonomy that is not present in traditional corpora (since they typically do not contain many specimens of this sort). In order to annotate the whole corpus, rather than a sample, one has of course to use automated, machine-learning techniques and, for very large corpora, efficient methods will have to be adopted (see, e.g., Chakrabarti et al. 2002).

While Sharoff and Ueyama categorize their Web corpus on a document-by-document basis, as one would do with a traditional corpus, Mehler and Gleim propose a rich representational system for Web hypertext, acknowledging that, to find meaningful textual units on the Web, we must look at whole webpages, which may or may not be spread across multiple HTML files. Again, we see here a difference in purpose. Traditional document-level annotation is probably more appropriate if we see the Web as a very rich source of data for what is ultimately to be used as a traditional corpus representative of a specific natural language, whereas Mehler and Gleim's approach looks at Web text as an object of study in itself. In any case, to annotate a connected set of Web documents according to Mehler and Gleim's proposal, automated categorizations techniques are also needed. Whether and how the complex, layered structures proposed by these authors can be induced using machine-learning techniques is an interesting topic for further research.

4 Indexing and searching a Web corpus

After the collection and linguistic annotation of a Web corpus as detailed in section 3, the data will typically be available as a collection of plain text files, usually in one-word-per-line format or in some XML format. The rich amount of authentic linguistic evidence provided by such a corpus, however, is useful only to the extent that the phenomena of interest can be retrieved from the corpus by its users. Like data cleaning, tools to store and retrieve linguistic data have been somewhat overlooked in traditional NLP work. However, they become fundamental when handling very large Web-derived corpora, where the classic ad hoc "disposable retrieval script" approach often adopted by computational linguists does no longer look like an attractive option. Development of indexing and retrieval software featuring a powerful query syntax and a user-friendly interface is probably the area in which most work still needs to be done, before we can start seriously thinking of a fully fledged "linguist search engine". Indeed, articles in this collection deal with nearly all other aspects of WaC work, but this is an area that is virtually unexplored by our authors. Consequently, we dedicate the longest section of this introduction to this topic.

In general, corpus data can be exploited in two ways: either by sequential processing (a typical example would be unsupervised training of a statistical NLP tool or a linguist reading through a corpus sentence by sentence), or by targeted search for a certain linguistic phenomenon (typically a particular lexical and/or syntactic construction). This type of search is often called a *corpus query*. A second distinction can be made between *online* processing, which is fast enough to allow interactive refinement of searches (for this purpose, query results should be available within a few seconds, or at most several minutes) and *offline* processing, where a task is started by the user and results might be ready for inspec-

tion after several hours or even days. For most linguistic purposes, the focus will be on online corpus queries.

In the following subsections, we discuss the requirements for an online query tool for Web corpora, henceforth called the *WaCky query engine*. Section 4.1 introduces four general requirements on software for the linguistic exploitation of Web corpora, as a basis for the ensuing discussion. Section 4.2 addresses the expressiveness of the query language itself, followed by the related technical issues of corpus storage and indexing strategies in section 4.3. Finally, section 4.4 argues for the combination of corpus queries with precompiled frequency databases, drawing on the advantages of both online and offline processing. The diverse components of the WaCky query engine can then be integrated seamlessly under a uniform Web interface that provides a familiar and convenient front-end for novice and expert users alike.

4.1 Requirements for linguistic search

The main challenge that online query tools for Web corpora face is to find a good balance between several conflicting requirements:

1. *Expressiveness*: The query tool should offer a flexible query language that allows the formulation of sophisticated queries to identify complex linguistic patterns in the corpus.

2. *Ease of use*: It should present a convenient and intuitive front-end to novice users.

3. *Performance*: It should support fast online searches, with response times that are short enough for interactive work even on very large corpora.

4. *Scalability*: It should be able to handle Web corpora of a billion words and more.

Different query tools will satisfy these requirements to varying degrees, focusing either on expressiveness or on speed and convenience. The two extremes of the range of possible approaches are perhaps best embodied by the Google search engine on the one hand (focusing on requirements 2–4) and the Nite XML Toolkit[19] on the other (focusing on requirement 1). Google searches several hundred billion words with ease and often returns the first page of matches within less than one second. However, it is restricted to simple Boolean queries on word forms, i.e., queries which test for the co-occurrence or non-co-occurrence of given word forms within the same document (a webpage, PDF file, etc.).[20] In contrast to this, the query language of the Nite XML Toolkit allows for complex logical expressions that build on multiple layers of equally complex linguistic annotations, but the current implementation is only able to handle corpus sizes well below 100,000 words.

The discussion in the following subsections depends to some extent on the type and complexity of annotations that have to be supported. Hence we will briefly sketch our assumptions about the annotations of Web corpora. Following section 3.4, we understand a Web corpus essentially as a sequence of word form tokens annotated with linguistic interpretations such as POS tags and lemmas. As pointed out there, meta-information about the speaker/writer of a text, its language, date of publication, genre, etc. is crucial for many applications. In most cases, such metadata can be represented as simple attribute-value pairs attached to the documents in the corpus (or to individual paragraphs, e.g., when a document contains text in different languages). In addition to this most basic

[19] See section 4.2 below

[20] For some languages, including English, Google also performs stemming, i.e., it attempts to remove morphological suffixes from search terms to broaden the search. However, since stemming is not performed in a linguistically consistent way and since it is not clear whether stemming can be disabled/enabled explicitly (i.e., to search for literal word forms), this renders the Google search engine unsuitable for many kinds of linguistic research.

set of linguistic annotations, shallow structural markup – ranging from text structure (paragraphs, sentences, lists, tables, etc.) to non-recursive chunk parsing – can significantly facilitate corpus queries and can be added by automatic methods with sufficient accuracy and efficiency. We will therefore also assume that many Web corpora contain such structural markup, with start and end points of structures indicated by non-recursive XML tags in the text or XML files.

Many users would certainly also like to have access to complete syntactic analyses of all sentences in the corpus, in the form of parse trees or dependency graphs. Such highly structured datasets put greater demands on the internal representation of corpus data, and require a fundamentally different type of query language than token-based annotations. Currently, we are not aware of any automatic tools that would be able to perform deep syntactic analysis with the required accuracy and coverage,[21] and the computational complexity of state-of-the-art systems leads to parse times ranging from several seconds to several minutes per sentence, rendering them unsuitable for the annotation of billion-word Web corpora.[22] Therefore, in the following discussion we assume that Web corpora are not annotated with complex syntactic structures. While this assumption reduces the demands on representation formats, it also means that the query language will have to provide sophisticated search patterns to make up for the lack of pre-annotated syntactic information.

[21] A recent statistical parser for German (Schiehlen, 2004) achieves F-scores between 70% (for constituents) and 75% (for dependency relations). While this level of accuracy might be sufficient for information retrieval and training of statistical NLP models, it does not provide a reliable basis for linguistic corpus queries.

[22] A syntactic parser that manages to analyze on average one word per second (which is faster than most systems that we have tested on current off-the-shelf hardware), would take 30 years to annotate a billion-word corpus.

4.2 A query tool for Web corpora

There is a wide range of query languages and implementations, which can be used for linguistic searches of different complexity. Here, we summarize the four most common approaches to corpus queries and discuss their suitability for the WaCky query engine.

The simplest method is *Boolean search*, which identifies documents that contain certain combinations of search terms (expressed with the Boolean operators AND, OR and NOT, hence the name). This basic type of Boolean search is exemplified by the Google search engine. More advanced implementations such as that provided by the open-source search engine Lucene[23] allow wildcard patterns for individual terms, constraints on metadata and to some extent also on linguistic annotations, and proximity searches for terms that occur near each other (similar to AltaVista's famous but discontinued NEAR operator). From the perspective of Web corpora, such tools can be used to build a simple concordancer that looks up individual keywords or phrases with optional metadata constraints. Proximity search allows for some variation in the phrases, and, with access to linguistic annotations, generalizations can also be expressed (e.g., that one of the words in a phrase is an arbitrary noun rather than a particular one). However, the Boolean combination of search terms is primarily designed to find documents about a particular topic (for information retrieval purposes) and will rarely be useful to linguists (although it could be used to identify simple collocational patterns).

Most of the currently available query engines for large corpora build on a *regular query language*.[24] Prominent implementations are the IMS Corpus WorkBench (CWB) with its query processor

[23] `http://lucene.apache.org/`

[24] "Regular" is used here as a the technical term from formal language theory, i.e., referring to patterns that can be described by regular expressions and finite-state automata.

CQP, the similar Manatee corpus manager (which is now part of the Sketch Engine) and Xaira, the successor of the SARA query tool supplied with the British National Corpus.[25] All three implementations are available under the GPL license. Regular query languages typically use regular expressions at the level of words and annotation values, and similar regular patterns to describe contiguous sequences of words (CQP and Manatee use a basic regular expression syntax for these patterns, but queries could also take the form of non-recursive rewrite-rule grammars, e.g., through use of CQP's built-in macro language). Many of these query languages extend the basic regular patterns. They may provide support for shallow structural markup, e.g., by inserting XML start and end tags in the query expressions. In CQP, matching pairs of start and end tags can be used to express shallow nesting of structures (e.g., PP within NP within S). Query languages will often also allow constraints on metadata, either appended to a query expression as "global constraints" or by pre-selecting a subcorpus of suitable documents for the search.

Some systems go one step further and allow queries to be formulated as *context-free grammars*. Unlike regular languages, this approach can identify recursively nested patterns of arbitrary complexity.[26] In addition, linguists often find it more intuitive to describe a search pattern with familiar context-free phrase-structure rules than to formulate an equivalent regular expression pattern (even when recursion is not required). Gsearch[27] is an *offline* corpus query tool based on context-free grammars, which is also

[25] More information about these tools can be found at the following URLs: `http://cwb.sourceforge.net/` (CWB), `http://www.textforge.cz/download.html` (Manatee), `http://www.sketchengine.co.uk/` (Sketch Engine), and `http://xaira.sourceforge.net/` (Xaira).

[26] As an example for such a structure, consider German noun phrase chunks, which may – at least in principle – contain an unlimited number of recursively nested, center-embedded noun phrases.

[27] `http://www.hcrc.ed.ac.uk/gsearch/`

available under the GPL license. We are currently not aware of any software using context-free rules for online queries.

In order to make use of deep linguistic analyses such as parse trees or dependency structures, *graph-based query languages* interpret a corpus together with all its annotations as a directed acyclic graph. Implementations of a graph-based query language include `tgrep`,[28] and the more recent TIGERSearch[29] and Nite XML Toolkit (NXT).[30] While graph-based query languages arguably offer the most flexible and powerful types of searches, they are also computationally expensive. Therefore, current implementations are limited to corpus sizes far below those of typical Web corpora.

The four approaches also differ in the type of results they return (the *query matches*). Boolean searches return matching documents or sets of tokens (i.e., instances of the search terms in each document). Regular query languages return contiguous sequences of words that match the specified lexico-syntactic pattern, and most implementations allow individual tokens within the sequence to be marked as "targets". Context-free grammars also return contiguous strings, but will often indicate substrings that correspond to constituents of the grammar (i.e., left-hand sides of the context-free rules), leading to a more structured search result. Finally, graph-based query tools return arbitrary tuples of graph nodes, which will often mix surface tokens with annotation nodes.

We consider regular query languages the most useful choice for searching Web corpora, because they strike a good balance between expressiveness and efficient implementation. Although a more structured representation of query results would sometimes be desirable, large result sets can only be stored and manipulated efficiently when they are limited to contiguous sequences (which

[28] `http://tedlab.mit.edu/~dr/Tgrep2/`

[29] `http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERSearch/`

[30] `http://nite.sourceforge.net/`

can compactly be represented by their start and end positions). Both the IMS Corpus WorkBench and Manatee seem to provide a good starting point for the implementation of the WaCky query engine. Being open-source software, they can be modified and extended to meet the requirements formulated in section 4.1. Since Manatee is closely modeled on CQP, we will henceforth use the label CQP (or, more generally, Corpus WorkBench) to refer collectively to both tools. There is only very limited information available on the query language and performance of Xaira at the moment, hence we do not pursue this option any further.

4.3 Indexing and compression

For fast online queries, *indexing* of the corpus and its annotations is essential in order to avoid linear search of the entire corpus data, which will typically occupy many gigabytes of disk space (even up to the terabyte range for broad crawls of the Web, cf. Clarke et al. 2002). In the most common form, an index contains, for every word and every annotation value, a complete ordered list of the occurrences of the word or annotation in the corpus.[31]

With the help of an index, simple keyword queries can be performed by direct lookup rather than by linear search through the full corpus. Especially for low frequency terms, index-based search can thus be faster by several orders of magnitude in some cases. More complex queries can still make use of the index by narrowing down the search space to stretches of text that contain an occurrence of the least frequent term specified: a query for noun phrases ending in the noun *beer* only has to consider sentences (or even

[31] Web search engines can substantially reduce the data size of their index by removing high frequency "stop words", which are then ignored in user queries. While this approach makes sense in an information retrieval setting (the word *the* is not a good indicator of the topic of a given document, since it occurs almost everywhere), stop words will often play a central role in linguistic searches and cannot be removed from the index.

smaller units) in which *beer* occurs at least once.

Index-based optimization of regular queries is often problematic because of their sequential, "left-to-right" structure. If the first term of a query is relatively infrequent, its occurrences can be looked up in the index. Matching of the regular expression pattern is then attempted only starting from these positions rather than from every token in the corpus. Such a strategy is employed by the CQP query processor, but it fails whenever the first query term describes a high frequency category such as a determiner or a noun.[32] Using an infrequent term that occurs in another place in the query for index lookup poses technical challenges. Because of the complex patterns of alternatives, optionality and repetition allowed by a regular expression, it is non-trivial to determine whether a given term must necessarily be part of every match (only in this case can it be used for index lookup). Even when such a term has been found, it will not be clear how far the start position of a match might be away from an occurrence of the term, so that the regular expression has to be matched "inside-out" from the lookup term rather than in the common "left-to-right" fashion.

Index-based optimization fails completely for regular queries that search for sequences of POS tags or other very general and frequent categories, e.g., queries that function as a "local grammar" for a particular syntactic construction. In this case, index lookup will have no substantial benefit, even if the final result set is very small. Optimization of such purely "grammatical" queries would only be possible with an *extended index* that includes combinations of POS tags in various configurations, combinations of POS tags with lexical elements, and perhaps also combinations of high frequency words. However, there is no limit to the number of relations that might need to be indexed: pairs of nearby POS

[32]The problem is compounded by the fact that there may be multiple potential start positions for a match in a regular expression, if the expression begins with optional elements or with a disjunction of alternatives.

tags at different levels of maximal distance, combinations of three or more POS tags, etc. Comprehensive indexing would thus lead to an explosive growth of data size beyond all practical limits.

Even in those cases where the index lookup narrows down the search space drastically, the resulting performance gain will often not be as large as one might have hoped. The reason for this behavior is that occurrences of the lookup term are usually spread evenly across the corpus, so that matching the full regular expression query requires random access to the huge amount of corpus data on disk. Purely index-based queries can be processed more efficiently because they access data sequentially, reducing the number of disk seeks and data blocks that have to be loaded. Such index-based implementations are straightforward and widely used for Boolean queries. This is what makes search engines like Google as fast as they are, and it may also be the key to fast online searches through Web corpora. Both CQP and Manatee provide at least rudimentary functionality for Boolean queries, though this feature does not fit in well with their standard regular query languages.

A final topic to be addressed in this section is the issue of data compression. Since disk reads are comparatively slow even when the data are accessed sequentially, as much of the corpus data as possible should be cached in main memory (where random access also is much less detrimental than for on-disk data). Therefore, better data compression translates directly into faster query execution: the benefits of a compact representation easily outweigh the decompression overhead. The IMS Corpus WorkBench applies specialized data compression techniques (Witten et al., 1999) both to the corpus data (word forms and their annotations) and to the index files.[33] Aggressive data compression is not without draw-

[33] In this way, the disk size of a 100-million word corpus (including the index, but without annotations) can be reduced to approximately 300 megabytes. For comparison, a plain text version of the same corpus has a size of 500 megabytes, and a gzip-compressed version has a size of 175 megabytes (but note that these

backs, though, mostly with respect to query execution speed. The block compression technique used by the CWB to store sequences of word forms and annotations makes random corpus access expensive even when the data are cached in main memory.[34]

To summarize the main points of this section, we have seen that indexing is essential to process online queries fast enough for interactive sessions. While basic indexing is a well-understood technique, it is of limited use for most linguistically interesting queries. Clearly, further research into suitable indexing techniques is needed in order to develop a powerful and fast query engine for Web corpora. The usefulness of data compression techniques is debatable, provided that fast and large hard disks are available. A stronger focus on extended indexes may be called for, not least because compression has fewer drawbacks for index data than for the text itself and its annotations.

4.4 The corpus as Web

While a powerful query language and a fast query processor are certainly essential for the linguistic exploitation of Web corpora, there are other important requirements as well. The potentially huge result sets returned by a query have to be managed and presented to the user, a task for which query engines like CQP and Manatee provide only rudimentary functionality. A minimum requirement is that users must be able to browse the query results (displayed with varying amounts of context), sort the matches according to different criteria, and look at random subsets of the results to get a broad overview. Especially for very large sets of results, additional functionality is desirable that helps to reduce and structure the massive amounts of data brought up by the

sizes do *not* include any index data).

[34] The reason is that for every access, an entire block of data (usually 256 tokens or more) containing the relevant token has to be decompressed.

corpus query. For instance, it should be possible to compute frequency lists of the matching word sequences (or individual target elements), to calculate distributions of the matches across metadata categories, and to identify collocations (in the sense of Sinclair 1991) or collostructions (Stefanowitsch & Gries 2003). All these functions are provided by BNCweb, a user-friendly interface to the British National Corpus (see, e.g., Hoffmann and Evert 2006).

While such analyses can be performed online for moderately large result sets, more advanced analysis options (e.g., exhaustive collocational analyses of the lexico-syntactic behavior of a word and automatic identification of other terms and phrases that have a similar distribution in the corpus) would further increase the usefulness of Web corpus data. Such complex analysis functions can only be performed offline, and the same is true for simpler functions when they are applied to result sets that contain millions of matches.

For each type of analysis, the final results can be represented as a table of corpus frequencies, statistical coefficients, similarity measures, etc. (usually linked back to individual query matches). A *relational database* software is ideally suited to store, process and query such tabular data structures (and this is the approach that BNCweb takes). Such a database provides an excellent environment to combine results from online and offline processing, where the latter can either stem from offline analysis of query results or from precompiled frequency tables for common words and phrases. We recommend the open-source implementation MySQL,[35] which is widely acclaimed for its stability, speed and flexibility.

A sketch of an architecture for the WaCky search engine is beginning to take shape, but we have to deal now with at least three distinct software packages: the query engine proper, a result browser, and a relational database. Moreover, at least two of these

[35] `http://dev.mysql.com/downloads`

tools require some amount of practice and in-depth knowledge of their (not entirely intuitive) query languages in order to achieve good results. Does this mean that Web corpora are essentially inaccessible for novice and non-technical users?

The enormous popularity that Google enjoys among linguists can only in part be explained by the fact that it makes an unprecedented amount of language data available. We believe that an equally important role is played by the fact that Google search is easy to use and can be accessed through a familiar user interface, presents results in a clear and tidy way, and that no installation procedure is necessary. For these reasons, we conjecture that the success of the WaCky query engine and its acceptance among linguists will hinge on its ability to offer a similarly user-friendly, intuitive and familiar interface. As in the case of Google, a Web interface has the potential to satisfy all three criteria. In other words, we should not only use the Web as a corpus, but also present the *corpus as Web*, i.e., provide access to Web corpora in the style of a Web search engine. A crucial advantage of the "corpus as Web" approach is that it allows us to hide the three (or even more) quite different components of the WaCky query engine behind a uniform Web interface. For the end user, the transition between query engine, result browser and tables in a frequency database will be seamless and unnoticeable, even if the technical implementation of this integration is a complex task. The key insight here is that complexity can and must be hidden from the user. Once again, BNCweb provides a good illustration of this approach, and a substantial part of the functionality sketched here has been implemented in the commercial Sketch Engine (built on top of Manatee and MySQL).

What is most urgently needed by the community now is an open-source implementation of a "corpus as Web" framework for the WaCky query engine, which should be easily configurable and extensible with new modules (providing, e.g., alternative visualiza-

tions of query results, additional analysis functions, or simplified query languages that shorten the learning curve for new users). For individual components of the system, open-source software packages are already available (such as the IMS Corpus WorkBench, Manatee and MySQL, as well as specialized software packages for statistical and distributional analyses), but may need to be improved and extended in order to meet the requirements listed in section 4.1. We are currently working on a detailed sketch of a possible architecture for the WaCky query engine and suggestions for the implementation of its components.

5 Conclusion

This introductory article looked at different ways in which the by now ubiquitous expression *Web as Corpus* can be interpreted, and provided an overview of the major issues involved in turning WaC from hype to reality. While doing this, we tried to provide a survey of some recurring themes in this collection, as well as describing some of our current and future work.

Despite the many daunting tasks that we might encounter on the way to its exploitation (actually, in part *because* of these daunting tasks), the Web is probably the most exciting thing that happened to data-intensive linguistics since the invention of the computer, and we would like to conclude this introduction by reiterating our invitation to the readers to engage, with us, in the WaCky adventure.

References

Baroni, M. and Bernardini, S. (2004). BootCaT: Bootstrapping corpora and terms from the Web. *Proceedings of LREC 2004*, 1313-1316.

Baroni, M. and Kilgarriff, A. (2006). Large linguistically-processed Web corpora for multiple languages. *Proceedings of EACL 2006, demo session*, 87-90.

Baroni, M. and Ueyama, M. (2006). Building general- and special-purpose corpora by Web crawling. *Proceedings of the 13th NIJL International Symposium*, 31-40.

Baroni, M., Kilgarriff, A., Pomikálek, J., Rychlý, P. (2006). WebBootCaT: Instant domain-specific corpora to support human translators. *Proceedings of EAMT 2006*, 247-252.

Broder, A., Glassman S., Manasse, M. and Zweig, M. (1997). Syntactic clustering of the Web. *Proceedings of the Sixth International World-Wide Web Conference*

Chakrabarti, S. (2002). *Mining the Web: Discovering knowledge from hypertext data*, San Francisco: Morgan Kaufmann.

Chakrabarti, S., Roy, S. and Soundalgekar, M. (2002). Fast and accurate text classification via multiple linear discriminant projections. *VLDB Journal* 12(2), 170-185.

Chklovski, T. and Pantel, P. (2004). VerbOcean: Mining the Web for fine-grained semantic verb relations. *Proceedings of EMNLP-04*.

Clarke, C. L. A., Cormack, G. V., Laszlo, M., Lynam, T. R. and Terra, E. L. (2002). The impact of corpus size on question answering performance. *Proceedings of SIGIR '02)*.

Hoffmann, S. and Evert, S. (2006). BNCweb (CQP-edition): The marriage of two corpus tools. In Braun, S., Kohn, K. and Mukherjee, J. (eds.) *Corpus technology and language pedagogy: New resources, new tools, new methods*, Frankfurt am Main: Peter Lang, 177-195.

Jurafsky, D. and Martin, J. (2000). *Speech and language processing*, Upper Saddle River: Prentice Hall.

Kilgarriff, A. and Grefenstette, G. (2003). Introduction to the special issue on the Web as corpus. *Computational Linguistics* 29(3), 333-347.

Manning, Ch. and Schütze, H. (1999). *Foundations of statistical natural language processing*, Boston: MIT Press.

Schiehlen, M. (2004). Annotation strategies for probabilistic parsing in German. In *Proceedings of COLING 2004*, 390-396.

Sinclair, J. (1991). *Corpus, concordance, collocation*, Oxford, OUP.

Stefanowitsch, A. and Gries, S. (2003). Collostructions: Investigating the interaction of words and constructions. *International Journal of Corpus Linguistics*, 8(2), 209-243.

Varantola, K. (2003). Translators and disposable corpora. In Zanettin, F., Bernardini, S. and Stewart, D. (eds.) *Corpora in translator education*, Manchester: St. Jerome, 379-388.

Witten, I. H., Moffat, A., and Bell, T. C. (1999). *Managing gigabytes, 2nd edition*, San Francisco: Morgan Kaufmann.