Experience Building a Large Corpus for Chinese Lexicon Construction

Thomas Emerson and John O'Neil

1 Introduction

The World Wide Web (WWW) provides a large and constantly growing renewable source of natural language data in many of the world's languages. Computational linguists and lexicographers have been trying to harness this bounty (and arguing about its applicability to any given task) for over six years (Kilgarriff and Grefenstette 2003). This chapter discusses our experience with using the Chinese Web for lexicon construction, focusing on the low-level details and problems we experienced during our initial proof-of-concept experiments.

1.1 Chinese text segmentation

Chinese is written without the use of spaces between words, which is problematic for natural language processing (NLP) applications which operate on words, information retrieval and data mining being two important and lucrative examples. The importance of accurately segmenting Chinese has made it an area of active research (Sproat and Emerson 2003; Wu 2003; Gao et al. 2004) around the world, and a variety of methods are used.

Initial attempts at addressing the problem used a variety of dictionary-based methods, such as maximal-matching: starting from the beginning of each sentence find the longest match in a dictionary, and move forward until the sentence is exhausted.

If a multi-character word cannot be found, then it is treated as a single character word and we move to the next position. Modifications of this simple algorithm to account for word frequency and other heuristics (such as POS compatibility) have been proposed. These techniques are often only as good as the dictionary that supports them, and as is well known constructing a lexicon is a time consuming (and often expensive) proposition. Further, unless the dictionaries are regularly updated they soon become stale as new words are used.

The lexical approach to word segmentation was overshadowed by the use of various statistical methods. These systems can be quite effective, so long as the text being processed is similar to that used to train them. These methods also suffer from a severe resource bottleneck problem, since they *a priori* require segmented text, which is (like lexicons) time consuming and expensive to obtain.

Recently more hybrid approaches have been proposed that utilize a mixture of statistical and lexical information. While these systems would seem to mitigate each other's limitations, they still need a comprehensive lexicon.

Building an electronic Chinese lexicon for use in a segmentation system is problematic: to generate the dictionary you need to segment a text collection, but to segment the text collection you need the lexicon. Substantial work has been done on developing techniques for constructing lexica with little or no human supervision from unsegmented text (Ge et al. 1999; Chang and Su 1997; Lin and Yu 2004; Jin and Wong 2002). In all cases the various techniques require a significant corpus to work on, due to the Zipfian nature of word frequency distributions (Baayen 2001).

1.2 Using corpora for linguistic analysis

Collecting a large corpus of Chinese text is challenging and difficult on its own, but of course the purpose of corpus collection is to

put it to good use. Extracting information about Chinese from a corpus poses a number of unusual challenges, and it is illuminating to discuss them, especially in comparison to corpus work in other languages.

Possibly the most common task for which corpora are used is lexical development, and this is precisely where the first difficulty lies. Because Chinese words are written without interstitial white space, it is necessary to develop a tokenizer for Chinese before one can develop a dictionary from a corpus. However, since the most obvious ways to tokenize arbitrary Chinese text involve using a lexicon, we immediately have a chicken-and-egg problem.

In practice, it has almost always been easier to use an existing lexicon to form the core of a tokenization tool, since lexica are more common than large tokenized corpora. (Of course, both can be used together, leading to systems that employ both lexical and statistical knowledge for segmentation.) In the absence of a good corpus, or in the absence of a corpus relevant to the domain of interest, and with a lack of relevant training data, a purely corpus-driven, unsupervised approach must be chosen.

Work on Chinese segmentation using only corpus materials has been an active research topic for some time (Sproat et al. 1994; Sun et al. 1998), and is still active (Gao et al. 2004). In general, abstracting away from statistical details, these methods look for sequences of characters that occur together more often than expected, and the more often they co-occur, the more likely they are to form a single token. This takes place in the context of an assumed segmentation algorithm. Any reasonable segmentation algorithm has to balance the absolute likelihood of a token, the likelihood of a token in a context, the total likelihood of a proposed segmentation of an entire sentence, and the possibility that there might be a new word in the sentence. These choices affect what lexical items are found, and how sentences are segmented.

Because these choices all have effects on the statistical anal-

ysis of the corpus, it can be difficult to create a consistent segmentation standard across a corpus, using entirely unsupervised methods. Since there is no generally accepted definition of what constitutes a word in Chinese, it can be unclear for an unsupervised learner, as it is for a human, how to decide on an appropriate level of granularity for segmentation, and apply it consistently. A lexicon is not created in a vacuum, and it can be difficult using unsupervised learning to create a lexicon useful for tasks other than segmentation, such as POS tagging.

Given the volume of the data, it would be advantageous to make the segmentation learner work incrementally on a stream of documents. Most previous work assumes a static, though large, corpus. Nevertheless, a continuous stream of new documents allows more accurate segmentation to be created over time. Also, if an unsupervised segmentation learner works on a continuous stream of documents, it has the benefit that it can be extended to continuously find new lexical entries. This is especially important in Chinese, since most new words permit alternate segmentations using tokens already in the corpus. Only additional statistical information, especially on new documents which might have a "burst" of a new word, can help identify new words.

There are other types of unsupervised learning from a corpus, especially once there is a reliable segmentation for the corpus. For example, clustering tokens based on their neighbors can bootstrap an assignment of POS tags to tokens in a lexicon. Clustering can also be used to group documents based on similar bags of words. However, since the lexical data is sparse, some clustering algorithms may not be optimal. In fact, since Zipf's law holds for even the largest of corpora, we are assured of continued sparseness.

As the corpus grows, we gain increased accuracy at the cost of being forced to process ever larger amounts of information. This makes it necessary to tune learning algorithms to use large corpora. The most common way to do this is to implement learning that

increases the precision at the expense of recall (Pasca 2004). As the corpus grows larger, it matters less what might be missed, since it will be seen again and again, but it becomes more important to avoid learning the noise and choking on the collected corpus data.

2 Problem statement

Crawling and post-processing large amounts of Chinese-language data are the first steps in a system designed to perform nearly continuous lexical development. The ultimate goal of the project is to develop an environment for finding and tagging possible neologisms in text from all Chinese-speaking communities for human adjudication before inclusion in a lexicon. This is similar to the ongoing LIVAC (Linguistic Variation in Chinese Communities) project at the City University of Hong Kong (T'sou et al. 1997). Unlike LIVAC, which focuses on news sources, we are interested in casting as wide a net as possible to include data from numerous registers and language levels.

We are also interested in utilizing the data we collect for other non-lexicographic purposes, such as the construction of POS tagged and entity tagged corpora for other high-level NLP tasks. These activities are currently secondary to the primary goal of lexicon construction.

Given the usage requirements, the following desiderata are apparent:

1. A Web crawler capable of processing millions or tens of millions of URIs (Uniform Resource Identifiers): a crawl will start with a seed set of several thousand URIs and will discover thousands more as it progresses.

2. The crawler must be polite to the sites being crawled, while optimizing throughput. This means obeying a site's robot-

exclusion preferences and not fetching documents from the site multiple times per second.

3. We are only interested in textual information, so we want to avoid downloading images, sounds, movies, and other arbitrary binary content: this is both a waste of bandwidth and storage. Ideally we would only download Chinese language content.

4. We do not want to wait for the crawl to complete before starting to process the data: this will become more important as the amount of stored text increases.

5. We want to be able to regularly recrawl sites that are known to change on a regular basis: online news sites and personal Web logs (blogs) are two obvious examples.

Writing a robust Web crawler from scratch would be an interesting project, and doing so may be a feasible solution for building small corpora. There are source code libraries available that provide HTTP protocol support and that can extract outgoing links from HTML, and these can be used in any number of programming languages including Python, Perl, and C/C++. Indeed, for a small enough collection just using a text-mode browser like `lynx` may be sufficient. However, for crawls on the scale we foresee the thought of having to maintain and extend the crawler was unappealing.

There are some open-source, command-line based downloading tools, such as `wget`[1] which can be used for downloading content, and even for mirroring entire sites. However, it has often been misused to the point that many sites' `robots.txt` file blocks all access from it. `wget` is also not designed to be a crawler and not suited for large ongoing crawls. We experimented with a modified

[1] `http://www.gnu.org/software/wget/wget.html`

version of `wget` for website archiving and link analysis but found that more time was spent inside the code than was spent on the real task. This is not to say that `wget` is poorly written, rather it simply did not serve our exact needs.

Ubicrawler (Boldi et al. 2004) has many of the features we could hope for, based on the above desiderata. However, it is not publicly available and we were wary of using software whose source we did not have ready access to.

Heritrix (Mohr et al. 2004) came to our attention when it was first announced in January 2004. We began using it for small focused crawls starting in May 2004, contributing bug fixes and new functionality to better serve our (and hopefully other linguistic researchers') needs. We quickly gathered experience with the code base, the developers, and the architecture and found it to fit our requirements very well.

3 Heritrix overview

Heritrix is an open-source Web crawler developed by the Internet Archive (IA).[2] Development of the crawler began at the beginning of 2003 after they determined that it would be beneficial for them to perform crawls internally. The crawler is written in Java, is modular, multi-threaded, and is capable of handling large crawls: the National University of Iceland has used it to crawl the entire `.is` domain (11,000 domains, 35 million URIs).[3]

There are three primary interacting components in Heritrix: the *Scope*, the *Frontier*, and the *Processor chains*.

The Scope determines whether or not a discovered URI should be included in the crawl, without actually fetching the data pointed to. Scopes can limit URIs to certain domains or sub-domains. A scope can use arbitrarily complex regular expressions to make the

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

[3] `http://groups.yahoo.com/group/archive-crawler/message/1385`

decision, and can reject files that are more than a certain number of links from a seed URI. Users can also develop their own scopes in Java if the built-in modules are inappropriate for a particular application.

The Frontier maintains the internal state of the crawl. It keeps track of which URIs have already been fetched, which are scheduled to be downloaded (i.e., that have been declared in scope), and which are currently being processed. It is responsible for determining which URI should be fetched next, paying attention to limitations set by a site's `robots.txt` file and to other forms of "politeness".

The Processor chains contain modules that operate over the URIs (and associated data, once it is fetched) to perform actions ranging from URI normalization to filtering based on length or headers to writing the fetched data to disk and providing crawl status information. Much of Heritrix's power lies in the configurability of these processor chains. All of the processors, as well as the scopes and frontiers, are extremely configurable.

Items downloaded by Heritrix are stored in an "ARC" (Web archive) file,[4] along with associated metadata, including the original URI, time stamp of when it was downloaded, MIME header, length, and fingerprint. By default each ARC file contains up to 100 MB of compressed data: during the crawl Heritrix maintains a pool of open ARC files (signified on disk by the ".open" extension on their file name) into which content is written by the crawler as it is processed. When an ARC file is full, the `.open` extension is removed and that ARC file is "complete": it will not be touched by the crawler again. This makes it possible to work with a crawl's ARC while the crawl continues to run – it can be moved to more permanent storage or its contents processed immediately.

Heritrix was designed for synchronic archiving and does not

[4]The format for ARC files is available at `http://www.archive.org/Web/researcher/ArcFileFormat.php`

support incremental crawling: an incremental crawler will refetch pages on a regular basis and update the stored copy with the updated version if it is different. Nevertheless, the ability to revisit sites was added to Heritrix (Sigurdsson 2005). At the time of writing, we have not had an opportunity to evaluate this addition.

The crawler has a Web-based user interface (WUI) that allows you to setup and monitor crawl jobs. You can define profiles with common settings that can be reused. These are stored on disk in XML and can be edited (or created) outside of the UI. You have full control of every aspect of the crawl operation from this console. Recent versions of the software have added a JMX (Java Management Extensions) interface, allowing it to be controlled from any JMX-enabled application or device.

Heritrix has an active developer community. The core team at the Internet Archive is supplemented by a number of people from around the world in both industry and academia, including linguists, digital librarians, and computer scientists. Further, they have worked with the Ubicrawler developers to incorporate some of their code.

3.1 Heritrix vs. desiderata

Successive generations of Heritrix have become increasingly capable in performing large crawls. While the Icelandic crawl mentioned in the previous section was done over 4 separate crawl jobs, it is believed that the entire 35 million URI snapshot could be done with a single crawl job. This is more than sufficient for our first requirement.

The default configuration for the crawler is to look for and obey a site's `robots.txt` file: this is an inviolate prerequisite that the users need to go out of their way to circumvent. There are numerous configurable settings for throttling the frequency with which documents are fetched from a given server. For example, you can set the delay between successive requests as a function of

the round-trip time of the last request made to a server. The frontier can also be configured with different scheduling mechanisms for handing off URIs to the worker (or toe, in Heritrix parlance) threads. Therefore only through operator error (or malice) will the crawler be impolite.

Given that one of the express design goals of Heritrix is to archive the Web, it is no surprise that in its default configuration it will attempt to fetch *everything* it can (assuming it isn't prevented by the `robots.txt`, of course.) However, through the use of the existing filtering mechanism offered by the architecture, one can almost eliminate all unwanted data from the crawl.

Content fetched from the Web is written into ARC files, which are closed after they reach around 100 MB in size. From that point Heritrix is done with them and they are available for processing: one can develop a work-flow that starts operating on the data while the rest of the crawl continues on.

Incremental crawling is the only desideratum that the current release of the crawler lacks, though Sigurdsson's (2005) work looks promising. For our current needs incrementality is not essential: we can just start new crawls based on the previous ones, relying on post-processing to remove duplicate documents.

4 Practical crawling issues

4.1 Seed generation

Our goal is to collect as much text as possible: rather than looking for specific linguistic constructs we need vast amounts of text to mine for neologisms. To this end we needed some way to find thousands of URIs with which to seed our crawls.

The Open Directory Project (ODP)[5] claims to be, "the largest, most comprehensive human-edited directory of the Web." Hun-

[5] `http://dmoz.org`

dreds of volunteers world-wide categorize millions of URIs according to defined criteria. All of the ODP data is freely available under the Open Directory License, which allows unlimited research and commercial use of the data as long as appropriate attributions are made and the rights outlined in the license are not impinged by subsequent distribution. Snapshots of the ODP database are made in slightly modified RDF every month or so: the release dated 28 July 2005 was 210 MB compressed and 1 GB uncompressed, containing some 4.5 million URIs in 551,578 categories.

The classification scheme used by the ODP includes regional and language-specific categories. For example, the category

```
Top:World:Chinese Simplified
```

contains pages that are known to be in Simplified Chinese. There are 1,993 sub-categories of this, counting for 16,535 URIs. The upshot of this is that it is trivial to extract all URIs in this category from the RDF file.

The ODP data is processed by extracting the categories and associated URIs from the RDF into a simple two-column tab-delimited file containing just the category and the URI. This reduces the size of the ODP database by almost 50% by eliminating the XML markup and removing unused information. This only has to be done once for each release of the data. After this file is created, it is trivial to extract just the links matching a particular category using `grep` and `cut`:

```
% grep Top/World/Chinese_Simplified ext.ut8 |
    cut -f 2 > zh_sc_uris.txt
```

A simple Python script is then used to generate a random sample of the extracted URIs:

```
% python pick_random.py 1500 zh_sc_uris.txt > seeds.txt
```

The resulting `seeds.txt` can now be used in a Heritrix job specification.

4.2 Job configuration

For lexicon construction we are only interested in HTML documents. Other document types, such as PDF or Microsoft Word, require more extensive processing than we chose to deal with. The first approximation for this is to exclude URIs from the scope with file extensions we do not care about. This can be done with a `URIRegExpFilter` with a long regular expression similar to:

```
.*(?i)\.(gif|pdf|wav|dvi|ps|iso)$
```

The expression that we actually use is considerably larger, containing 176 extensions.[6] There are two file extensions that could not be included in the filter, `au` and `txt`. The `au` cannot be excluded because this would cause sites in Australia (whose ISO 3166 code is also `au`) to be excluded from the scope in some situations, and `txt` was kept because its omission would cause `robots.txt` to be excluded, violating the hard prerequisite Heritrix has for handling the robot exclusion protocol.

Unfortunately filtering just on file extension does not exclude all content: very often URIs that yield images or PDFs are generated from CGI scripts or other dynamic methods and lack a file extension. To account for these cases, we install a `ContentType RegExpFilter` as a MidFetch filter (run after the HTTP response headers are received but before the content) to filter on the content type:

```
(?i)text/html.*
```

There are two other options that need to be set for each job. The first is `default-encoding`, which is the character encoding that is used for files that do not explicitly declare one. When working with multi-byte character sets it is important that Heritrix

[6] `http://www.dreamersrealm.net/~tree/blog/?p=4`

know what encodings it is likely to see. Failure to set this appropriately can result in broken link extraction. The second is to add an appropriate `Accept-Language` header to the `accept-headers`. Some sites do content negotiation to send appropriately translated content to the browser. Without explicitly specifying the `Accept-Language` you may not receive the content you expect. For Simplified Chinese sites it is best to set the default encoding to CP936 (Microsoft's Simplified Chinese code page) and add:

```
Accept-Language: zh-cn, zh-sg
```

For Traditional Chinese sites, the default encoding is CP950 (Microsoft's Traditional Chinese code page) and the accept header:

```
Accept-Language: zh-tw, zh-hk
```

5 Crawl experiences

Using the methods described in the previous section we generated a random set of 1,500 Simplified Chinese URIs from the May 2005 ODP data release. A sample of the 16,000 URIs available in the ODP Simplified Chinese section were used to constrain the size of this crawl. We ran a local pre-release build of Heritrix 1.4.0 on an old dual-CPU 666 MHz x86 machine with 1 GB physical memory and running Gentoo Linux 2005.1 with Sun's JDK 1.4.2. This machine was dedicated to the crawl. We gave the Java virtual machine a 512 MB heap and this was sufficient for the crawl.

The crawler was initially configured to use 50 threads (i.e., fifty concurrent connections). This was increased every other day until we reached 150 threads. We elected to use the "Domain" scope, which allows any URI in the domain of one of the seeds to be crawled. A depth restriction (number of hops from a seed) of 25 was used. We let the crawl run for approximately 11 days before manually stopping it due to a lack of disk space. When

URIs stored:	7,372,351
ARC files:	300
Total ARC File Size:	28 GB
Unique Hosts Crawled:	4,032
Total HTML size:	109.7 GB
Total Stripped size:	15.8 GB
Languages found:	28

Table 1. Statistics on the first large Chinese crawl

Simplified Chinese	5,510,748	Romanian	52
Traditional Chinese	50,030	Persian	38
Russian	5,986	Hungarian	32
Japanese	4,059	Finnish	28
Korean	393	Bulgarian	26
Arabic	365	Spanish	11
Polish	198	Albanian	11
Greek	136	Vietnamese	10
Thai	120	Swedish	8
Turkish	83	Latvian	5
Czech	67	German	5
Portuguese	65	Icelandic	3
Hebrew	58	Slovak	2
Lithuanian	55	French	1

Table 2. Breakdown of languages found in the first large Chinese crawl

the crawler was shutdown it had stored 7,372,351 URIs, or approximately 27,926 per hour, or around 8 documents per second. Further statistics on the crawl can be found in tables 1 and 2.

5.1 Disk issues

The gating factor on the length of this crawl was disk space: the crawl had been running for almost two weeks until running out of disk space due in large part to the amount of "state" data that was being stored: it dwarfed the amount of data stored in the ARC

files (48 GB to 28 GB). This saved state data is only needed during the crawl: once the crawl is terminated the state information can be deleted. It appears that the ratio of state to "content" is highly dependent on the type of content being stored: our use of Heritrix to only download textual data is somewhat unique. The IA has observed that for archival crawls the state is only around 15% of the ARC file size.[7] The Heritrix developers were subsequently able to implement some size reduction on the data stored in the state files, though we have not had an opportunity to study the effects of this change in our crawls.

During a crawl "disk contention" can become a performance bottleneck too:

- The crawler keeps a pool of ARC writers, which the threads use to write the content they are downloading. Each of these contends for the disk. Interestingly enough, the IA found that increasing the number of ARC writers does not help performance, but can actually lower it. The rationale is that increasing the number of writers increases the amount of contention for the disk, which ends up being a more time-consuming operation than keeping threads waiting for a writer.

- The crawler maintains at least four log files during the crawl, so there is contention for writing (and, for the administrative interface, reading) these.

- The state data. As observed with the current crawl, there is a lot of this: not only does it consume disk space, it can result in disk contention during the crawl.

Heritrix allows you to split the ARCs, logs, and state across different physical disks. This can go a long way to reducing con-

[7] `http://groups.yahoo.com/group/archive-crawler/message/1870`

tention on a single disk, and is the recommended way of dealing with this.

Based on our experience with the Chinese crawl, we need to allocate about 150% of the space taken by the expected crawled data size for state information. This storage is only needed during the crawl, and can be reclaimed when it completes. This becomes a real problem if we run multiple large-scale crawls on a single machine where you could expect to use 50-150 GB of disk space, per crawl, for the state information.

6 Post processing

For vocabulary acquisition we need to extract the raw text from the Chinese documents stored in the ARC files. This processing was done after the crawl was completed, but it could be done incrementally as the ARC files are closed: the steps are repeated for each ARC.

Post-processing is done in two phases: we first extract all interesting documents from the ARC files, and then lift the text from the HTML.

The first phase works as follows: each `text/html` item in the ARC file has its HTML markup stripped. If the amount of text left after removing markup is greater than a threshold (1024 bytes) then we perform language and encoding detection using Basis Technology's Rosette Language Identifier, a commercial language/encoding detection system. With the 1024 byte threshold the identifier is almost 100% accurate. Documents that are detected to be Simplified Chinese (regardless of character encoding) are then marked for further processing.

Items that reach this stage are extracted from the ARC file in the original HTML, *except* that they are transcoded from the detected character encoding to UTF-8 with HTML character entities expanded. These are written to disk into numbered files

contained in numbered directories, with at most 1,000 files per directory. This is done since few file systems are capable of working reliably with directories containing tens of thousands (if not millions) of files. Note that we do not rewrite any character set declarations that may exist in the original HTML file: these are never used.

Once the ARC files have been processed in this way, they can be moved to offline storage since the "interesting" content has been extracted. Our policy is to keep the ARCs for each crawl for future use and research.

The second post-processing phase is lifting the text from the markup. For some purposes (though not necessarily lexicon extraction) it is useful to have the rough physical structure of the text preserved, and many utilities which merely remove markup do not preserve this. We envision "sifting away" the markup and leaving the text in place, with structure preserved. To do this we use an open source tool called `vilistextum`[8] which is robust in the face of "broken" markup and does a decent job of preserving the logical structure of the documents. Each file extracted in the first phase is passed through Vilistextum and saved with the same basename but different file extension. The HTML files generated in phase 1 can then be deleted (since they can be trivially regenerated from the ARCs) or moved to offline storage. Table 3 gives some statistics on the resulting text.

We do not yet perform any (near-)duplicate or boiler-plate removal. This is an important future direction, and we are examining various techniques to do this. Most existing duplicate document detection algorithms presume efficient tokenization of the input documents, which we do not have in the case of Chinese. This is a problem that we will need to tackle in the near term, since duplicate documents will artificially inflate the statistics we use to find new words in the texts.

[8] `http://bhaak.dyndns.org/vilistextum/`

Number of files:	3,291,985
Average file size:	4,935 bytes
Total *hanzi*:	3,861,758,249

Table 3. Statistics on the Simplified Chinese text

7 Data management

7.1 Crawl data

Data management becomes a significant issue as the size of the crawls increases.

Given that the content we store is almost exclusively textual the compression ratios are quite good (17:1). However, a large crawl still generates a lot of data that needs to be stored and backed up.

The raw data that is crawled is not immediately useful for many of our tasks, so it must be post processed. This raises several issues, including: when is the processing performed? Is the processed data saved, or do we always process on demand? How do we deal with the duplicate and near-duplicate data problem? How do we extract the data we're interested in from the huge amount available? Again, compression can be used to help with disk space issues. Do we want (or need) to be able to map back from processed data to the original ARC file and to a specific crawl?

We have no way of knowing how much data is available for a particular set of parameters (e.g., language, encoding, content type).

Backups and data integrity are difficult; backing up 28 GB of ARC files requires at least 7 DVD-R discs. One solution is to use one or more external FireWire drives to archive the data after it is crawled. Unfortunately this single-point of failure caused us to earlier loose about 100 GB of data when the file system on

the external drive became corrupted and unrepairable. This may have been an issue with the FireWire drivers on Gentoo, or an issue with the ext3 file system, or a combination of these.

7.2 Processed data

The data from each physical URI is stored in a single file after all processing is complete. It is possible to work backwards from the file name to the ARC file containing the original HTML. This means, however, that there are hundreds of thousands of files living on the file system, which is obviously problematic for many reasons.

Initially we generated a `bzip2` compressed `tar` file containing the extracted data. We do much of our linguistic processing in the Python language, which has the ability to read the entries of these archives. Unfortunately this didn't work since Python was unable to process files over 4 GB in size (a bug which has since been fixed).

Another large collection, the LDC's Chinese Gigaword (Graff et al. 2005)[9] contains 349 compressed SGML files which in turn contain multiple news articles along with other markup. We could concatenate multiple files into one, and compress this, but doing so involves the addition of extra markup that we do not want to add.

8 Conclusion

Since the crawl documented here, we have performed a second Simplified Chinese and a first Traditional Chinese crawl of similar size. We have not yet started lexicon extraction on any of these corpora, although this will proceed in the near future.

[9] This corpus contains approximately 1.3 billion characters, slightly less than one-third the size of the crawl described here.

Heritrix has worked very well for the tasks we have given it. Nine times out of ten the problems we've encountered have been of our own doing, and the responsive development team have been quick to point out our errors or to correct problems that we have encountered. The software continues to improve, and the architecture is proving itself again and again. The addition of the JMX interface is particularly exciting, as we can envision integrating the crawler into a Web-based lexicographer's workbench.

The biggest concerns are generally pragmatic: finding enough disk space to actually store the crawl data and associated transient state; sharing bandwidth with the rest of the company; post-processing the collected data. These are problems that any large-scale crawling effort will encounter.

Our next steps involve integrating the crawler and its data into the linguistic processing modules of the system, and making the crawls incremental so that we can continue to expand our lexica as time goes on. We are also expanding our crawling efforts into other languages, and looking at ways of expanding Heritrix to perform directed crawls of specific languages for which readily available corpora of any size do not exist (Ghani et al. 2001).

Acknowledgments

The authors would like to thank Michael Stack of the Internet Archive for his comments on the section describing Heritrix. The work reported in this article was conducted while Thomas Emerson worked at Basis Technology.

References

Baayen, R. (2001). *Word frequency distributions*, Berlin: Springer.

Boldi, P., Codenotti, B., Santini, M. and Vigna, S. (2004). Ub-

icrawler: A scalable fully distributed Web crawler. *Software: Practice & Experience* 34(8), 711-726.

Chang, J. and Su, K. (1997). An unsupervised iterative method for Chinese new lexicon extraction. *Computational Linguistics and Chinese Language Processing* 2(2), 97-148.

Gao, J., Li, M., Wu, A. and Huang, C. (2004). Chinese word segmentation: A pragmatic approach. Technical Report MSR-TR-2004-123, Microsoft Research.

Ge, X., Pratt, W. and Smyth, P. (1999). Discovering Chinese words from unsegmented text. *Proceedings of the 22nd International SIGIR Conference*, 271-272.

Ghani, R., Jones, R. and Mladenić, D. (2001). Mining the Web to create minority language corpora. *Proceedings of the 10th International Conference on Information and Knowledge Management*, 279-286.

Graff, D., Chen, K., Kong, J. and Maeda, K. (2005). Chinese Gigaword, second edition. Lexical Data Consortium, LDC2005T14.

Jin, H. and Wong, K. (2002). A Chinese dictionary construction algorithm for information retrieval. *ACM Transactions on Asian Language Information Processing* 1(4), 281-296.

Kilgarriff, A. and Grefenstette, G. (2003). Introduction to the special issue on the Web as corpus. *Computational Linguistics* 29(3), 332-347.

Lin, Y. and Yu, M. (2004). The properties and further applications of Chinese frequent strings. *Computational Linguistics and Chinese Language Processing* 9(1), 113-128

Mohr, G., Kimpton, M., Stack, M. Ranitovic, I. (2004). Introduction to Heritrix, an archival quality Web crawler. *Proceedings of the 4th International Web Archiving Workshop.*

Pasca, M. (2004). Acquisition of categorized named entities for Web search. *Proceedings of the Thirteenth ACM Conference on Information and Knowledge Management (CIKM 04)*, 137-145.

Sigurdsson, K. (2005). Adaptive revisiting in Heritrix. Master's thesis, University of Iceland.

Sproat, R. and Emerson, T. (2003). The first international Chinese word segmentation bakeoff. *Proceedings of the Second SIGHAN Workshop on Chinese Language Processing*.

Sproat, R., Shih, C., Gale, W. and Chang, N. (1994). A stochastic finite-state word-segmentation algorithm for Chinese. *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, 66-73.

Sun, M., Shen, D. and T'sou, B. (1998). Chinese word segmentation without using lexicon and hand-crafted training data. *Proceedings of COLING-ACL '98*, 1265-1271.

T'sou, B., Lin, H., Liu, G., Chan, T., Hu, J., Chew, C. and Tse, J. (1997). A synchronous Chinese language corpus from different speech communities: Construction and applications. *Computational Linguistics and Chinese Language Processing* 2(1), 91-104.

Wu, A. (2003). Customizable segmentation of morphologically derived words in Chinese. *Computational Linguistics and Chinese Language Processing* 8(1), 1-28.