Mining Web-scale Treebanks

Stijn De Saeger　Kentaro Torisawa　Jun'ichi Kazama

Language Infrastructure Group, MASTAR Project,
National Institute of Information and Communications Technology
{stijn,torisawa,kazama}@nict.go.jp

Abstract

Empirical linguistic research presupposes the availability of large collections of syntactically annotated natural language text. When such data collections grow to terabyte size, investigating specific linguistic phenomena or harvesting lexico-syntactic patterns using simple tools quickly becomes unmanageable, and specialized methods for handling large tree data become necessary. Instead of reporting on specific research results this paper will introduce and compare a number of existing tools that can be used to mine Web-scale Japanese treebanks. Facing data sets of these dimensions, we found that an information retrieval tool with some knowledge of XML worked better for us than highly specialized tree mining tools.

1 Why Size Matters

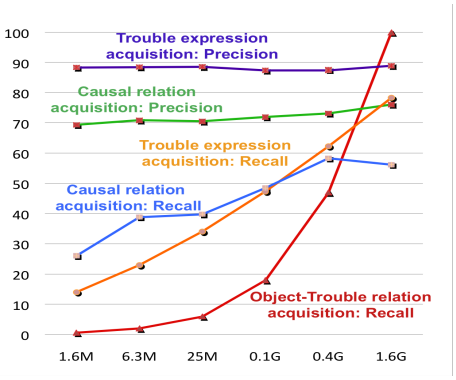
Knowledge acquisition algorithms crucially depend on the availability of large collections of natural language text for harvesting instances of lexico-syntactic constructions that are indicative of some semantic phenomenon of interest. Exploiting such linguistic clues to automatically discover semantic relations in unstructured text was pioneered by Hearst ([4]) and has since given rise to the field of relation extraction in NLP.

The power law distribution of word occurrences known as Zipf's law is particularly important in the context of knowledge acquisition and relation extraction. To appreciate the effect of corpus scale we performed a number of experiments with drastically reduced data sets. Figure (a) shows how precision and recall figures for some earlier knowledge acquisition experiments (see De Saeger et al., [3]) change when scaling our Web corpus (see below) down to the proportions of the average news articles archive (i.e. around 1.6 million sentences, or 0.1% of the original corpus). While *precision* seems to remain largely constant, the *recall* graphs are heavily affected by the data reduction. This is especially the case for the "*Object-Trouble relation acquisition: Recall*" graph, which shows the coverage of what we defined in [3] as a semantic relation between objects or artifacts and potential problems associated with their use. It clearly demonstrates the long tail of rare but relevant relation instances that can only be acquired given a sufficiently large corpus of text.

2 A Web-scale Treebank

NLP researchers usually try to tap the vast seas of information on the Web by submitting queries such as "*tobacco* * * * * * *cancer*" to a commercial search engine in order to retrieve text snippets containing the words "*tobacco*" and "*cancer*" in a 5 word window. This approach has a number of drawbacks. First, search engines tend to limit programmatic access to their indices to a fixed number of queries per day. Second, the number of returned results is often limited as well, and the mechanism behind their ranking is unclear. Finally, using results from string-based queries means that one has to work with bare word sequences as data. While in theory one could do syntactic analysis *after* retrieving the patterns, the text snippets returned by search engines typically don't preserve sentence boundaries, making this option difficult in practice. Often though, syntactic dependency information presents stronger evidence for the existence of a particular semantic relation than mere co-occurrence in the same n word window.

To help address some of these problems, the TSUB-



(a) Knowledge acquisition precision and recall as a function of data scale.


```
<?xml version="1.0" encoding="utf-8"?>
<StandardFormat Url="http://www.umekouji.co.jp/voice/view.php/print_20.html" OriginalEnco
  <Header>
    <Title Offset="768" Length="45" is_Japanese="1" JapaneseScore="0.93333" is_Japanese_S
      <RawString>うめこうじ　お客様の声のご紹介</RawString>
    <Annotation Scheme="Knp"><![CDATA[* 1P <BGH:埋める/うめる><文頭><用言:動><係:連用><レ
:837><連用要素><並列類似度:3.774><並結句数:2><並結文節数:1><提題受:30>
+ 1P <BGH:埋める/うめる><文頭><用言:動><係:連用><レベル:B><並キ:述:&レベル:強><区切:3-5><
うめ うめ うめる 動詞 2 * 0 母音動詞 1 基本連用形 8 "代表表記:埋める/うめる" <代表表記:埋
動詞:産む/うむ ドメイン:家庭・暮らし 代表表記:産める/うめる"><ドメイン:家庭・暮らし><品曖
><活用語><自立><意味有><タグ単位始><文節始>
* 5D <BGH:講ずる/こうずる|高ずる/こうずる|講じる/こうじる|高じる/こうじる><用言:動><係:連
用><RID:837><連用要素><並列#犹###:-100.000>
+ 5D <BGH:講ずる/こうずる|高ずる/こうずる|講じる/こうじる|高じる/こうじる><用言:動><係:連
用><RID:837><連用要素>
こうじ こうじ こうずる 動詞 2 * 0 ザ変動詞 17 基本連用形 8 "代表表記:講ずる/こうずる" <代
る -2-0-17-8-"代表表記:高ずる/こうずる"><ALT-こうじ-こうじ-こうじる-2-0-1-8-"代表表記:講じ
表記:高じる/こうじる"><品曖-動詞><品曖-その他><原形曖昧><かな漢字><ひらがな><活用語><自立
* 3D <受けNONE><係:NONE><RID:1485>
+ 3D <受けNONE><係:NONE><RID:1485>
        特殊 1 空白 6 * 0 * 0 NIL <英記号><記号><自立><意味有><タグ単位始><文節始><疑似代
* 4D <SM:人><BGH:御/お+客/きゃく><連体修飾><人名><助詞><体言><係:ノ格><区切:0-4><RID:1067
+ 4D <SM:人><BGH:御/お+客/きゃく><連体修飾><人名><助詞><体言><係:ノ格><区切:0-4><RID:1067
節内:T解析格-ヲ>
お お お 接頭辞 13 名詞接頭辞 1 * 0 * 0 "代表表記:御/お" <代表表記:御/お><かな漢字><ひらか
客 きゃく 客 名詞 6 普通名詞 1 * 0 * 0 "漢字読み:音 ドメイン:ビジネス 代表表記:客/きゃく"
漢字><かな漢字><名詞相当語><自立><意味有>
```


(b) TSUBAKI's web standard format.

AKI search engine[1] ([6]) provides unrestricted programmatic access to a collection of 100 million Japanese Web pages containing 6×10^9 sentences (1.6×10^9 unique), or 3.1TB worth of gzip compressed data. The crawled Web pages have been preprocessed, including sentence boundary detection, morphological analysis with JUMAN[2] and dependency parsing using KNP[3]. The usual work flow is that users submit a query to the TSUBAKI search engine API as an HTTP request, and get an XML document in response containing basic statistics like hit counts, text snippets and a list of document IDs of relevant web pages. These preprocessed pages (an example of which is shown in Figure (b)) can then be retrieved by ID.

While obviously a great improvement over closed commercial search engines, for the specific task of mining lexico-syntactic patterns this process is still not as efficient as could be, and reconstructing the relevant syntactic constructions from the KNP output in the XML documents still requires a fair amount of programming on the part of the user. Moreover, for large data such naive pattern extraction method rapidly exceeds the limits of what can be achieved in any reasonable amount of time, and the challenges implied by Web-scale data mining require the use of specialized tools and architectures. For this reason we have compiled the TSUBAKI corpus into a large, distributed treebank which can be mined using specialized tools. In section 3 we discuss the treebank's specifics. Section 4 introduces some tools we used for mining patterns, and section 5 concludes.

[1] http://tsubaki.ixnlp.nii.ac.jp/index.cgi

[2] http://www-lab25.kuee.kyoto-u.ac.jp/nl-resource/juman-e.html

[3] http://www-lab25.kuee.kyoto-u.ac.jp/nl-resource/knp-e.html

3 Treebank Format

We converted the TSUBAKI document collection from Web standard format (Figure (b)) to the Penn treebank format. Processing was done in parallel on 28 cluster nodes of the "*New IT infrastructure for the Information-explosion Era*" [4] research project's InTrigger platform.

The KNP dependency parsed sentences were mapped to trees as follows. The content of a terminal tree node consists of a word, and its JUMAN stem and polarity, separated by colons. Polarity and stem only matter for verbs and adjectives and is often just ignored. Tree node labels are composed of a major label and a minor label corresponding to the node's POS and sub-POS, separated by a hyphen. For example, JUMAN POS "名詞" and sub-POS "サ変名詞" become a tree node with label "`N-sa`". The label of a bunsetsu's last terminal node that is a content word gets passed up the tree to make phrases. As node labels get indexed by the tree mining tools it pays to include as much useful information in the labels as possible. Therefore we inserted the following additional information in the label of non-terminal nodes — the KNP dependency kind (`D`, `P` or `A`), the main verb's polarity in the case of VPs (1 or 0) and some basic case information

[4] http://www.infoplosion.nii.ac.jp/info-plosion/ctr.php/m/IndexEng/a/Index/

for NPs (the phrasal head's post-position). An example tree is shown below.

```
(VP-1:D
   (NP-ni:D
     (N-sa 旅行:旅行:1)
     (PP-kaku に:に:1))
   (V-* 行って:行く:1)
   (SUFF-verb きた:くる:1)
   (PUN-period 。:。:1))
```

After construction the tree data remained distributed over the 28 cluster nodes, with about 15 GB worth of gzip compressed trees per node.

4 Tree Mining Tools

In this section we introduce three tools for mining the treebank introduced above: *TGrep2*, *Tregex* and *Wumpus*. All three are licensed under the GNU General Public License (GPL), which makes their usage free for research purposes. Throughout the last year we have used them extensively for extracting lexico-syntactic patterns from the TSUBAKI tree data, yet as we will see they each represent a different compromise in the trade-off between query expressivity and scalability.

We wish to stress that this is not meant to be an exhaustive overview of all possible tools available for this task, nor to imply that these are necessarily the most appropriate for the job. Also, query execution timings reported below should be interpreted as rough approximations, as our benchmarks could not control for parameters like system load on a multi-user cluster.

TGrep2 TGrep2 [5] aims to be a *grep for trees*, yet it offers much more functionality than that. It accepts data in the Penn Treebank format and has a sophisticated tree expression language that allows one to assert complex Boolean, dependency and precedence relationships between tree nodes, use regular expressions and even name nodes to create back-references inside a tree expression. A full treatment of TGrep2's tree expression grammar is beyond the scope of this paper, but to give an example the query below can be used to retrieve all instances of verbs occurring in a VP that dominates an NP marked with post-position "で" and either a noun "スプーン" or "ナイフ".

[5] http://tedlab.mit.edu/~dr/TGrep2/

```
/^V-/ > (/^VP/ < (/^NP/ < (/^PP/ < /で/) [ <
(/^N/ < /スプーン/) | < (/^N/ < /ナイフ/) ]))
```

TGrep2 first needs to build a binary corpus file which doubles as an index. One of TGrep2's greatest strengths is that while these corpus files contain a full copy of the data they can be gzip compressed with little effect on extraction performance (decompression cost is offset by reduced disk IO). On each of the data nodes our TGrep2 corpus weighs in at around 13GB, which is impressive considering the compressed tree data is already about 15GB. This makes TGrep2 by far the most scalable of the three in terms of space. Building the corpus files took several days. Query execution time is highly variable, ranging from minutes to hours. Running the example query above on one node took about 16m45.41s.

However, we found some cases where the tree expression engine's behaviour deviates from the documentation. For instance, in some cases the relation "`(A $.. B)`" (meaning `A` is a sister of and precedes `B`) does not look beyond its immediate right sister. Ultimately this is what prompted us to investigate alternatives.

Tregex *Tregex*[6] was developed at the Stanford Natural Language Processing Group. It is written in Java and comes with a GUI program that helps the user construct tree patterns on a set of in-memory trees. Tregex's tree pattern language extends TGrep2's expressiveness with (i) constrained dominance and precedence relations between nodes, (ii) headship as a primitive relation[7], and (iii) variable groups for co-indexation relations. Its pattern language is effectively a strict superset of TGrep2's. This makes it relatively painless to switch from TGrep2 to Tregex. An added benefit is that Tregex comes with a *tree transformation tool* called "Tsurgeon" which takes Tregex tree patterns to move, prune, replace, adjoin or in other ways transform a matching node (Levy et al., [5]).

For our purposes the most substantial demerit is that Tregex does not preindex the trees — it needs to load each data file into memory. This rules it out as a stand-alone mining tool for Web-scale treebanks, but it may still have its use as a post-processing tool. For instance, Tregex and Tsurgeon may be utilized for pruning irrelevant subtrees from a smaller set of trees, like the results of a TGrep2

[6] http://nlp.stanford.edu/software/tregex.shtml

[7] To be defined by the user. Tregex does come with headship rules for the English, Chinese and Arabic Penn Treebanks, and the NEGRA and TIGER treebanks for German (see [5]).

query. Tregex is by far the most expressive query tool of the three, but scalability issues prevented it from being the final solution to our tree mining problem.

Wumpus *Wumpus* (Buettcher et al. [1]) was not conceived as a tree mining tool per se. It was developed at the University of Waterloo as a generic information retrieval system. It boasts excellent performance in the TREC Terabyte Track information retrieval competitions [8] and handles text collections of several hundreds of gigabytes, and millions of documents.

Wumpus features a simple but powerful query language called GCL ("*Generalized Concordance Lists*", based on Clarke et al. [2]) that allows one to emulate tree expressions to some extent. GCL queries return *index extents* (integer intervals in the index address space that map to a byte-offset in the original data file). Instead of dependency relations, GCL defines a number of *scope* operators. A query `(A > B)` returns the shortest extent that matches `A` and contains an extent matching `B`, while its inverse `(A < B)` returns extents that are contained in (the shortest extent matching) B. The negations of these scope operators are written "`/>`" and "`/<`", and `[`n`]` finds all extents of length n.

Combined with a structured text format like XML, these scope operators allow one to emulate tree expressions. For instance, the following query is the GCL equivalent of our earlier example.[9]

```
("<V>" .. "</V>") < (("<VP>" .. "</VP>")
> (("<NP>" .. "</NP>") > (("スプーン" + "ナイ
フ") ^ (("<PP>" .. "</PP>") > "で"))))
```

The subtree corresponding to an extent can then be retrieved using the `@get` command. Furthermore, Wumpus has commands for retrieving frequency counts for a query and for ranked retrieval of documents based on Okapi BM25, the vector space model, language modeling or divergence from randomness.

For our experiments we converted the treebanks from the Penn treebank format to their equivalent XML structures[10], roughly giving 75GB of XML data per node. Indexing this data took around 12 hours and resulted in a 60GB index. Running the above query on a "cold" system (no part of the index cached in memory) took 1m29.84s, with subsequent queries running in around a minute (retrieval itself is instantaneous). A frequency count of all occurrences on one node took 1m31.45s.

In summary, Wumpus offers fast indexing and excellent scalability and retrieval performance, at the cost of some reduced query expressivity when searching for complex tree structures. In addition, the ability to get basic statistics like co-occurrence frequencies from the search engine is a great benefit for many NLP tasks. Combined with the wide-spread availability of XML processing tools for managing the data itself this makes it our current tool of choice for mining large treebanks.

5 Conclusion

Each tool we discussed offers its own unique combination of features — Tregex offers an expressive tree expression language and with Tsurgeon the ability to perform complex trees transformations. However, it does not bother with the storage or indexing of tree data, which diminishes its usefulness for mining Web-scale treebanks. On the other end of the spectrum is Wumpus, which offers a complete solution for indexing and retrieval of large data sets, at the expense of some reduced expressiveness when searching for complex tree patterns. Finally, TGrep2 sits somewhere inbetween the other two in the expressiveness/scalability trade-off.

References

[1] Stefan Buettcher and Charles L. A. Clarke. Efficiency vs. effectiveness in terabyte-scale information retrieval. In *Proceedings of the 14th Text REtrieval Conference (TREC 2005), Gaithersburg, MD*, 2005.

[2] Charles L. A. Clarke, G. V. Cormack, and F. J. Burkowski. An algebra for structured text search and a framework for its implementation. *The Computer Journal*, 38:43–56, 1995.

[3] Stijn De Saeger, Kentaro Torisawa, and Jun'ichi Kazama. Looking for trouble. In *Proc. of The 22nd International Conference on Computational Linguistics (Coling2008)*, 2008.

[4] M. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proc. of COLING'92*, pages 539–545, 1992.

[5] Roger Levy and Galen Andrew. Tregex and tsurgeon: tools for querying and manipulating tree data structures. In *Proceeding of LREC 2006*, 2006.

[6] Keiji Shinzato, Tomohide Shibata, Daisuke Kawahara, Chikara Hashimoto, and Sadao Kurohashi. Tsubaki: An open search engine infrastructure for developing new information access. In *Proc. of IJCNLP*, pages 189–196, 2008.

[8] http://trec.nist.gov/

[9] This is an approximation though, as ">" and "<" only emulate a sort of *dominance* relation, not a direct parent-child dependency.

[10] We did strip off minor labels from the XML tags to save space.