Design of a Multi-lingual, Parallel-processing Statistical Parsing Engine

Daniel M. Bikel
Department of Computer and Information Science
University of Pennsylvania
200 South 33rd Street, Philadelphia, PA 19104-6389
dbikel@cis.upenn.edu

1. INTRODUCTION

Ever since the widespread availability of the Penn Treebank [9], there have been numerous, statistical parsers developed for English, *e.g.* [8, 5, 3]. To varying degrees, these parsers and others—while very successful at the tasks for which they were designed—had the following limitations:

- they had a fairly fixed probabilistic structure, which could only be changed by re-coding some significant portion of the program[1]
- they had hard-coded features specific to English
- they had hard-coded features specific to the Penn Treebank
- they were designed only for a uniprocessor environment

Building on our work in [1] and [2], we have developed a design for a head-driven, chart parsing engine that addresses all of the above limitations, and we present this design here. In particular, our design provides

- appropriate layers of abstraction and encapsulation for quickly porting to different languages and/or Treebank annotation styles,
- has "plug-'n'-play" probability structures and
- is multi-threaded for use in a multiprocessor and/or multi-host environment for throughput far superior to traditional uniprocessing parsers.

2. BACKGROUND

[1]Based on remarks by Charniak at the NAACL 2000 conference, the parser of [3] does not appear to have this limitation, possessing some flexible mechanism to specify different probability structures.

.

2.1 Probability structures and sentence-level parallelism

In preparation for the work presented in [1], we explored a variety of lexical probability structures involving WordNet synsets. To facilitate this experimentation, we developed a "plug-'n'-play" lexical probability structure architecture. The model we were extending was extracted from BBN's SIFT system [10], which is a history-based model that is derived in large part from Model 2 of [5].

For the computation of probability estimates, the BBN parser had hard-coded construction of data objects to represent the future and history events, computing a maximum-likelihood estimate of $P(X|Y)$ by $\frac{c(X,Y)}{c(Y)}$, where $c(\cdot)$ delivers the count of its argument in the training data. The architecture of [1] placed a layer of abstraction at the computation of the histories for the various back-off levels of the parser: in pseudocode, the construction of a history object for a particular back-off level became simply

```
history =
  probabilityStructure.get(backOffLevel,
                           fullContext);
```

We also made an initial foray into parallel-processing by developing a multi-threaded *sentence server*, to provide parallelism at the sentence level in a cluster computing environment. This work paved the way for the significantly greater degree of parallelism in the architecture of our current parsing engine.

2.2 Language independence

For [2], we took two parsing models, BBN's SIFT-derived parser and David Chiang's stochastic TAG parser [4] and adapted them to parse Chinese. The results were that the Chinese-adapted models performed with accuracy close to their English counterparts when trained on comparably-sized corpora and tested on their respective domains.[2] The main difficulty of the project was ferreting out and replacing all traces of English-specific or Treebank-specific code in the parsers, such as replacing code dependent on the default character encoding and finding and abstracting away from language-specific word features and parts of speech. This project greatly influenced the language-independent design of our current architecture, which allows for a "language package" that contains all data and methods specific to a particular language and/or Treebank annotation style.

[2]Our two comparably-sized corpora were: ˜100k words of WSJ text from the (English) Penn Treebank and ˜100k words of Xinhua newswire text from the Chinese Treebank. The stochastic TAG model performed at 77%/78% labeled precision/recall on the Xinhua test set, compared to 79%/80% on the English test set. See [2] for more details.

3. DESIGN OVERVIEW

The parsing engine we have developed supports a wide range of head-driven parsing models, including that of BBN's SIFT system, as well as both Models 2 and 3 of [7]. Figure 1 gives a pictorial overview of the design.

3.1 Language package

The most significant data encapsulation of the design is that of the language package. The language package, which will typically be an actual Java package,[3] is a collection of Java classes that are extensions of several abstract classes which provide the specification of data and methods specific to a particular language and Treebank annotation style. There are four such required classes: `Treebank`, `Training`, `HeadFinder` and `WordFeatures`. The `Treebank` class provides all data and methods specific to a particular Treebank, such as predicates for preterminals and complex nonterminal labels. The `Training` class provides methods and data specific to preprocessing training trees. The primary function of the `HeadFinder` class is to read a data (text) file specifying head rules specific to a particular language's Treebank and provide a head-finding method. Finally, the `WordFeatures` class provides a mapping of lexical items of a particular language to orthographic/morphological word-feature–vectors, to aid part-of-speech tagging.[4] As an additional language-independent design criterion, all relevant input and output files are written to and read from with a user-settable character encoding.[5] Our goal is, for treebanks that are not too radically different from the Penn Treebank (the Chinese Treebank [13] and possibly the Korean Treebank have this property, *e.g.* [12]), to be able to create a new language package in about 1–2 weeks. The one language-dependent module that currently does not sit inside the language package is WordNet; this is simply due to the fact that we have yet to implement our WordNet-extended model. Now that the initial implementation is complete, we will make language-specific lexical resources part of the language package framework.

3.2 Probability structure objects

Every type of output element of the models supported by our engine—including nonteminals, preterminals (parts of speech), words, word-features, gaps and subcat frames—has an associated `ProbabilityStructure` object, that specifies how to form the data objects representing the future and history at all possible back-off levels for that output element. Objects containing maximal context—called `TrainerEvent` objects—are passed in to a single method which then delivers the appropriate history or future for a specified back-off level, as per the pseudocode example above in Section 2.1. A layer of abstraction also exists for these probabilistic events: objects that represent events must conform to an `Event` interface.[6] This abstraction allows for maximum flexibility in the underlying types and representation of event elements, which are typically the output elements of the parser, but can also include other features derived from those output elements. In this way, the user can not only easily test new probability structures in our architecture, but also new events with new *types* of event elements.

[3] A Java *package*, much like its Lisp progenitor, provides a separate name space for a collection of classes and their data members, as well as providing yet another level of access control for data and methods (i.e., data and methods can be made accessible only within their enclosing package).

[4] Unlike [5, 6], part-of-speech tagging can be fully integrated into the models supported by our architecture. Along these lines, word features have been shown to reduce part of speech ambiguity for unknown words [11] and may be employed by models in our architecture.

[5] Java stores all characters internally as unicode characters, and most runtime environments support a very wide variety of character encodings, supporting an even wider variety of the world's languages, making Java an ideal language with which to implement internationalizable software.

[6] An *interface* is a Java construct (borrowed from the Smalltalk programming language) that specifies a contract for implementations to follow, in the form of a set of method (function) signatures, augmented as needed by API documentation.

3.3 Probability-level parallelism

The simplest form of parallelization is to increase a parser's throughput by splitting up a test file into multiple segments and have multiple executables run on these segments, each on a separate host. Our sentence server provided a finer granularity of parallelism, but still required a separate parser on each host. Schemes such as these are a monumental waste of memory resources: each parser contains the same massive tables containing counts of events observed in the training data. Our current design exhibits a much finer degree of parallelism by introducing a *probability server*. The idea is that a `DecoderServer` object can sit on a multiprocessor with a large amount of RAM, providing smoothed top-level probability estimates to mutliple, small chart parsing clients. The architecture also supports load-balancing if multiple, large multiprocessing hosts are available. Java provides an elegant solution to such distributed computing via a built-in technology called *remote method invocation*, or RMI.

An additional feature of our distributed-computing parsing engine is that it is fully *fault-tolerant*: any of the `DecoderServer` objects or parsing clients can fail and the system will keep on operating. Even the `Switchboard` itself can fail (see Figure 1), since the servers and clients will patiently wait to reconnect to a switchboard and to each other, and a new switchboard, when it is brought up, will recover the same state of the old switchboard at the time of its failure.

Finally, the distributed system need not be a collection of closely-knit, identical hosts. Given that the Java runtime environment is available for virtually any operating system and architecture, the parsing engine can and has been run on a disparate set of hosts, running, for example, Solaris, Linux, Windows and MacOS X, where the only requirement is internet connectivity.

4. BUILT FOR SPEED

We did not want all the flexibility above to compromise the efficiency of our system; thus, we spent significant effort at optimizing the decoder.[7] We employed the following categories of optimizations:

1. precomputation of all log-probability estimates and log-lambdas, necessitating only floating-point add operations for computing probability estimates during decoding

2. creation and use of hash maps optimized for mapping arbitrary Java objects to primitive types, achieving spatial locality in probability lookups and eliminating the need for primitive wrapper objects

3. short-circuiting all decoding operations when it is evident that the resulting new chart item will have a zero probability

4. elimination of almost all new-object creation during decoding, including the use of an object pool for efficient recycling of chart item objects

5. smaller optimizations based on profiling

[7] Our thanks to Scott Miller of BBN for his invaluable advice for parser optimization.

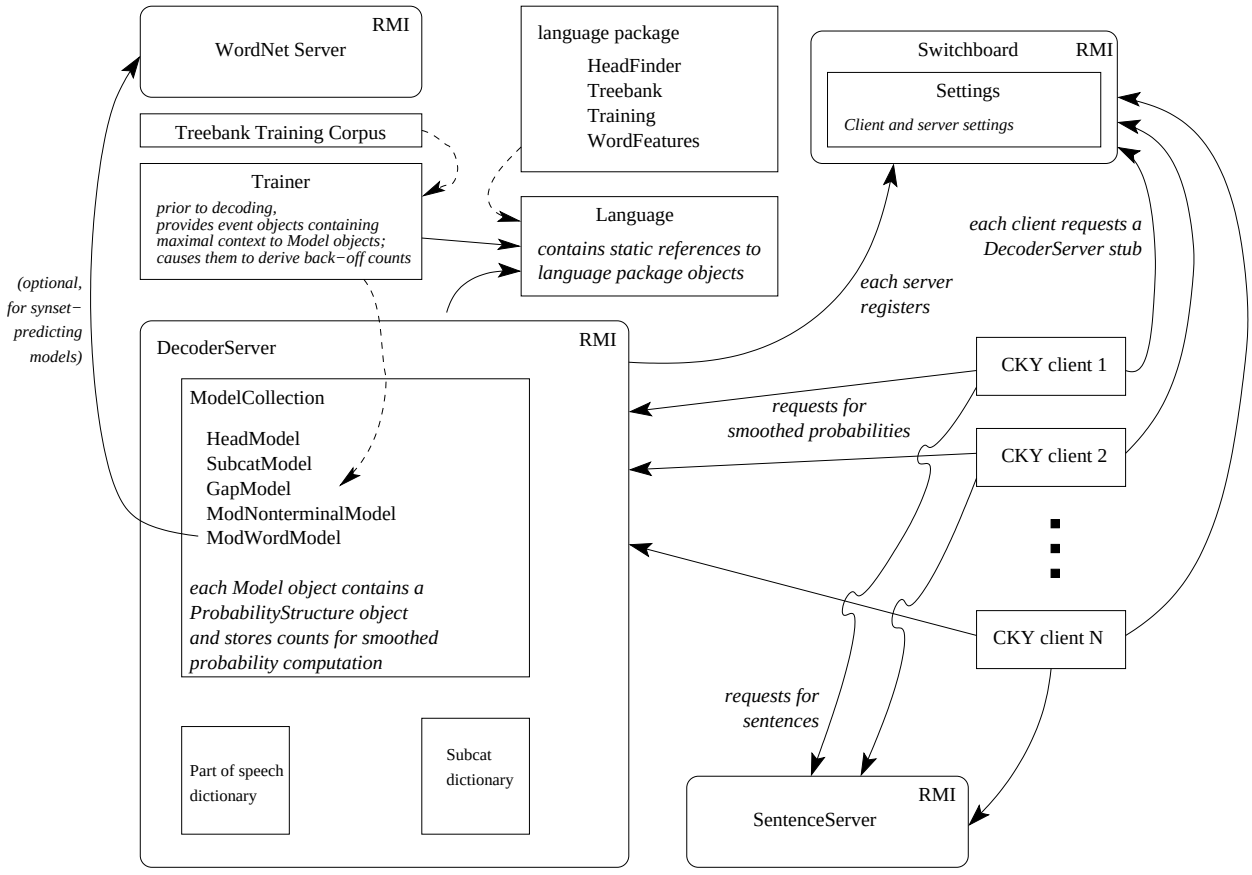

Figure 1: Pictorial design overview. Arrows indicate functional relationships. While there is necessarily always a bi-directional flow of information, the solid arrows indicate the direction of requests from a client to a server (or from a consumer to a producer). Dashed arrows explicitly indicate the flow of information or objects.

	≤ 40 words		≤ 100 words	
Parser	**LR**	**LP**	**LR**	**LP**
Collins	89.75	90.19	88.47	89.30
Bikel	89.89	90.14	88.72	89.03

Figure 2: Results of Collins' thesis Model 2 and our replication of this model on Section 00 of the Penn Treebank. LR = labeled recall. LP = labeled precision.

Optimization technique 3 was of crucial importance, achieving the single greatest speed improvement of them all. In particular, before the computation of the modifier-generation probability, the decoder first checks to see if the last level of back-off for the modifier-generation model (the history) ever occurred with the particular modifier being generated (the future) in training. If not, then it will be impossible for the model to deliver anything but a zero probability. Put another way, if the coarsest context never occurred with the future being generated, then certainly more specific contexts could never have occurred, and thus the model would delive a zero probability for the entire smoothed estimate. This optimization alone resulted in a five-fold speedup, and so it was also implemented for the subcat-generation model, resulting in an even greater speedup.

5. FLEXIBILITY PUT TO THE TEST

5.1 Replicating Collins' thesis model

The models described in [7] have been used for many tasks and are thus of great importance in the field. As a baseline for our own research, we attempted to replicate Model 2 from Collins' thesis using our new engine. Although this replication turned out to be more difficult than anticipated, happily, we have now achieved results on the development test set that are nearly identical to those reported in Collins' thesis (see Figure 2). [8]

We have documented all our implementation efforts with extreme thoroughness using `javadoc`, a documentation tool that is integrated with the standard Java software development kit. So that Collins' important model will be fully described in the public domain, we aim to release our system to the public for research purposes next year, both as a means to provide a fully documented version of Model 2 from Collins' thesis, and also as a resource for other researchers interested in statistical parsing or language processing in general. We hope that, given its flexibility, our engine will be a useful research testbed for others.

5.2 Developing a language package for Chinese

As mentioned in Section 3.1, a desideratum of our engine was to be able to implement a new language package in 1–2 weeks. We recently implemented a language package for Chinese for use with the Chinese Treebank in only *one and a half days*. Not only was this significantly shorter than our expected implementation time, but we also achieved state-of-the-art results. Using the division of data into training and test sets set up in our previous work [2], on the sentences of length ≤ 40, our engine achieved a labeled recall of 77.0% and a labeled precision of 81.6%. In comparison to the other parsers of which we are aware that have trained and tested on these same data, our parser's recall is equivalent and the precision is significantly higher. With these encouraging results, we hope to improve Chinese parsing performance even further, and also to implement several more language packages for the rapidly-expanding set of treebanks in other languages.

[8] After reading [7] thoroughly—it is a very detailed and well-written document—and then implementing its Model 2, there was a puzzling gap between the performance of our parser and the model's documented performance. As it happened, the replication of Model 2 required a painstaking analysis of the data files output by Collins' trainer in order to reconstruct what the trainer was doing, as well as a detailed analysis of the parser code. These analyses revealed, among other things, discrepancies between details described in [7] and their actual implementation. Additionally, the replication required several e-mail exchanges with Collins himself to clear up ambiguities in both his thesis and the parser code. Many thanks to Mike Collins for his invaluable advice during our efforts to replicate his model.

6. CONCLUSION

In the design presented here, we have shown how using various engineering practices and technologies of modern computer science, we can address several limitations of current statistical parsers. Indeed, almost all of the techniques and technologies we used here can be applied to NLP software in general, particularly in light of the growing need to develop efficient, internationalizable software. By abstracting away from the idiosyncrasies of training data and from a particular probability structure for parser output elements, we provide parser developers a means to test various probability structures, using features and back-off schemes that are the result either of pure theorizing or empirical study or (presumably) both, and we allow this experimentation while maintaining a high degree of computational efficiency. As has been shown in [6], choosing linguistically-motivated features and independence assumptions can often result in a well-estimated, high-performance model. It is thus our hope that continued application of sound engineering practices will not only lead to more efficient, scalable and portable NLP software, but will also benefit the exploration and testing of linguistic theories.

7. REFERENCES

[1] Daniel M. Bikel. A statistical model for parsing and word-sense disambiguation. In *Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, Hong Kong, October 2000.

[2] Daniel M. Bikel and David Chiang. Two statistical parsing models applied to the Chinese Treebank. In Martha Palmer, Mitch Marcus, Aravind Joshi, and Fei Xia, editors, *Proceedings of the Second Chinese Language Processing Workshop*, pages 1–6, Hong Kong, 2000.

[3] Eugene Charniak. A maximum entropy–inspired parser. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 132–139, Seattle, Washington, April 29 to May 4 2000.

[4] David Chiang. Statistical parsing with an automatically-extracted tree adjoining grammar. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, 2000.

[5] Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of ACL-EACL '97*, pages 16–23, 1997.

[6] Michael Collins. Discriminative reranking for natural language parsing. In *International Conference on Machine Learning*, 2000. (to appear).

[7] Michael John Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

[8] D. Magerman. Statistical decision tree models for parsing. In *33rd Annual Meeting of the Association for Computational*

Linguistics, pages 276–283, Cambridge, Massachusetts, 1995. Morgan Kaufmann Publishers.

[9] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19:313–330, 1993.

[10] Scott Miller, Heidi Fox, Lance Ramshaw, and Ralph Weischedel. SIFT – Statistically-derived Information From Text. In *Seventh Message Understanding Conference (MUC-7)*, Washington, D.C., 1998.

[11] R. Weischedel, M. Meteer, R. Schwartz, L. Ramshaw, and J. Palmucci. Coping with ambiguity and unknown words through probabilistic methods. *Computational Linguistics*, 19(2):359–382, 1993.

[12] Fei Xia, Chung-hye Han, Martha Palmer, and Aravind Joshi. Comparing lexicalized treebank grammars extracted from Chinese, Korean, and English corpora. In Martha Palmer, Mitch Marcus, Aravind Joshi, and Fei Xia, editors, *Proceedings of the Second Chinese Language Processing Workshop*, pages 52–59, Hong Kong, 2000.

[13] Fei Xia, Martha Palmer, Nianwen Xue, Mary Ellen Okurowski, John Kovarik, Shizhe Huang, Tony Kroch, and Mitch Marcus. Developing Guidelines and Ensuring Consistency for Chinese Text Annotation. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC-2000)*, Athens, Greece, 2000.