Partial Parsing via Finite-State Cascades

Steven Abney
University of Tübingen

Steven Abney (1996). Partial Parsing via Finite-State Cascades. In *Proceedings of the ESSLLI '96 Robust Parsing Workshop.*

Many of my papers are available in PostScript on my web page:

http://www.sfs.nphil.uni-tuebingen.de/~abney/
abney@sfs.nphil.uni-tuebingen.de
Wilhelmstr. 113, 72074 Tübingen, Germany

Partial Parsing via Finite-State Cascades

Steven Abney
University of Tübingen

Abstract

Finite-state cascades represent an attractive architecture for parsing unrestricted text. Deterministic parsers specified by finite-state cascades are fast and reliable. They can be extended at modest cost to construct parse trees with finite feature structures. Finally, such deterministic parsers do not necessarily involve trading off accuracy against speed—they may in fact be more accurate than exhaustive-search stochastic context-free parsers.

1 Finite-State Cascades

Of current interest in corpus-oriented computational linguistics are techniques for bootstrapping broad-coverage parsers from text corpora. The work described here is a step along the way toward a bootstrapping scheme that involves inducing a tagger from word distributions, a lowlevel "chunk" parser from a tagged corpus, and lexical dependencies from a chunked corpus. In particular, I describe a chunk parsing technique based on what I will call a **finite-state cascade**. Though I shall not address the question of inducing such a parser from a corpus, the parsing technique has been implemented and is being used in a project for inducing lexical dependencies from corpora in English and German. The resulting parsers are robust and very fast.

A finite-state cascade consists of a sequence of **levels**. Phrases at one level are built on phrases at the previous level, and there is no recursion: phrases never contain same-level or higher-level phrases. Two levels of special importance are the level of **chunks** and the level of **simplex clauses** [2, 1]. Chunks are the non-recursive cores of "major" phrases, i.e., NP, VP, PP, AP, AdvP. Simplex clauses are clauses in which embedded clauses have been turned into siblings—tail recursion has been replaced with iteration, so to speak. To illustrate, (1) shows a parse tree represented as a sequence of levels.

(1)

L_3							S			S
T_3										
L_2		NP				PP	VP	NP		VP
T_2										
L_1		NP	P			NP	VP	NP		VP
T_1										
L_0	D	N	P	D	N	N	V-tns	Pron	Aux	V-ing
	$_0$ *the* $_1$	*woman* $_2$	*in* $_3$	*the* $_4$	*lab* $_5$	*coat* $_6$	*thought* $_7$	*you* $_8$	*were* $_9$	*sleeping*

Parsing consists of a series of finite transductions, represented by the T_i in (1). A number of researchers have applied finite transducers to natural-language

Steven Abney (1996). Partial Parsing via Finite-State Cascades. In *Proceedings of the ESSLLI '96 Robust Parsing Workshop.*

parsing [7, 8, 10]. Typically a transducer calculus is developed and syntactic analysis is accomplished by inserting syntactic labels into a word stream. By contrast, in a finite-state cascade, spans of input elements are reduced to single elements in each transduction, as in traditional parsing.

To illustrate, consider the regular cascade (2).

(2)
$$
\begin{aligned}
T_1 &: \left\{ \begin{array}{l} \text{NP} \rightarrow \text{D? N}* \text{ N} \\ \text{VP} \rightarrow \text{V-tns} \mid \text{Aux V-ing} \end{array} \right\} \\
T_2 &: \{\text{PP} \rightarrow \text{P NP}\} \\
T_3 &: \{\text{S PP}* \text{ NP PP}* \text{ VP PP}*\}
\end{aligned}
$$

Each transduction is defined by a set of patterns. A **pattern** consists of a category and a regular expression. The regular expression is translated into a finite-state automaton, and the union of the pattern automata yields a single, deterministic, finite-state **level recognizer** T_i in which each final state is associated with a unique pattern. The recognizer T_i is run with L_{i-1} as input, and it produces L_i as output. The recognizer starts in a distinguished start state, and input symbols determine state transitions in the usual way. If the recognizer enters a final state at more than one position in the input, only the longest match creates an output phrase. If the recognizer blocks without reaching a final state—and this is routinely the case, inasmuch as patterns only pick out easily recognizable constructions, rather than attempting to be exhaustive—then a single input element is "punted" to the output and recognition resumes at the following word.

For example, in (1), the recognizer T_1 begins at word 0 in level L_0. It reaches a final state associated with the NP pattern at position 2, and outputs an NP from 0 to 2 at L_1. The recognizer is then restarted at position 2. No transition is possible, so P is punted. Starting from position 3, final states associated with the NP pattern are reached at positions 5 and 6. Taking the longest match, the recognizer outputs an NP from 3 to 6 at L_1. Then recognition continues from position 6.

A finite-state cascade is very fast, being little more than a pipeline of Lex-style lexical analyzers [9]. Unlike traditional parsers, there is no global optimization. This contributes not only to speed, but also to robustness. Namely, a common problem with traditional parsers is that correct lowlevel phrases are often rejected because they do not fit into a global parse, due to the unavoidable incompleteness of the grammar. This type of fragility is avoided when lowlevel phrases are judged on their own merits.

If the speed of the parser is attributable to its architecture, its effectiveness is largely a function of the grammar. The grammar is viewed, not as a linguistic description, but as a programming language for recognizers. The goal is to write patterns that are reliable indicators of bits of syntactic structure, even if those bits of structure are "boundaries" or "kernels" rather than traditional phrases.

The reliability of patterns is key. The philosophy is **easy-first parsing**—we make the easy calls first, whittling away at the harder decisions in the process. By keeping pattern precision high, we can parse deterministically with acceptable error rates. Parsing proceeds by growing **islands of certainty** into larger and larger phrases. Easy-first parsing means that we do not build a parse-tree systematically from bottom to top, but rather recognize those features of structure that we can. Where reliable markers for high-level boundaries are recognized, uncertain intermediate-level structure can be skipped over with "ANY*" expressions to go straight to higher-level phrases. The result is **containment of ambiguity**. Containment of ambiguity plays a key role in the use of chunk-and-clause parsing in the bootstrapping of argument structures. PP's and the like whose attachment is uncertain are left unattached, but the possible attachment sites are limited by the containing simplex clause. Also within noun chunks, ambiguities like noun-noun modification are contained but not resolved.

2 Features and Internal Structure

In the application of the parser to argument-frame induction, syntactic features are necessary, particularly for case information in German. It is also at times convenient to be able to insert, after the fact, some of the internal structure that regular-expression patterns flatten out, or to insert "linguistic" phrases in cases where the pattern includes surrounding context or multiple traditional phrases. Regular cascades can be modified to compute feature inheritance and internal structure at a modest cost in efficiency, as follows.

We extend patterns to include **actions**. An example is the pattern (3):

(3) $\text{Subj} \rightarrow [_{\text{NP}}\ n{=}\ \mathbf{D}?\ n{=}\ [_{\text{N1}}\ \mathbf{A}^*\ n{=}\ \mathbf{N}\ ]\ ]\ \mathbf{V}$

The symbols "$[_{\text{NP}}$", "$n{=}$", "$[_{\text{N1}}$", and "]" represent actions. The pattern (3) translates to a finite transducer in which the boldface symbols ($\mathbf{D}$, $\mathbf{A}$, $\mathbf{N}$, $\mathbf{V}$) are input symbols, and the actions are output symbols. I hasten to emphasize the difference between this transducer and the transductions T_i that we discussed earlier. The transductions T_i are computed using level recognizers. Let us call transducers such as that generated by the extended expression (3) **internal transducers**. Internal transducers do *not* replace level recognizers. Extended patterns such as (3) are stripped of actions and compiled into automata whose union yields level recognizers, just as before. Level recognizer T_i is called with L_{i-1} as input, and creates L_i as output. *After* a phrase associated with pattern p is recognized, the internal transducer for pattern p is used to flesh out the phrase with features and internal structure.

Further, unlike in unification grammars, recognized phrases are *not* rejected because of unification failures. Indeed, there is no unification per se: features are represented as bit vectors, and feature assignment involves bitwise operations on those vectors.[1] For example, the case feature of the German word *der* might be represented as `1000 0110 0000 0100`, meaning that it is either masc. sg. nom., fem. sg. gen., fem. sg. dat., or pl. gen. Case features are combined by bitwise "and." If *Mann* is `1011 0000 0000 0000`, then *der Mann* ends up being `1000 0000 0000 0000`, i.e., unambiguously masc. sg. nom. If *Haus* is `0000 0000 1011 0000`, then *der Haus* ends up as all zeros. But it is not for that reason rejected as a phrase.

After a phrase is recognized, we run the internal transducer on it. That is, we associate states of the internal transducer with the positions in the input spanned by the recognized phrase, requiring of course that transitions respect input symbols and that a final state be reached at the end of the phrase. In general, the transducer is not deterministic. We do a backtracking search that postpones epsilon transitions whenever possible, and accepts the first path that successfully reaches a final state at the end of the recognized phrase.

By associating states of the internal transducer with input positions, we also associate actions with input positions. A right-bracket action at position i means "create a new current phrase ending at i". A left-bracket action at i means "the new current phrase begins at i". An assignment action f= at i means "copy feature f from the input symbol or old phrase at i to the current phrase".

Given these interpretations, actions cannot simply be executed left-to-right. Rather, we must create the innermost phrase first, do the feature assignments for which it is target, then drop it into the input and repeat with the next innermost phrase. Fortunately, the order in which actions are to be executed can be determined at compile time, so that we do not have to sort the actions at run time.

3 Evaluation

The speed of finite-state cascades has already been mentioned. The speed of the current implementation (Cass2) is quite sensitive to the number of levels: the parser is about 2/3 faster with a two-level grammar as with a nine-level grammar. With nine levels, the parser runs at about 1600 words/second (w/s) on a Sparcstation ELC, or an estimated 1300 w/s on a Sparcstation 1. By comparison, parser speeds reported in the literature range from less than one w/s to nearly 3000 w/s. I have attempted to adjust the following figures for hardware, taking a Sun4/Sparcstation 1 as standard.[2] Traditional chart parsers

[1] Those features with a fixed set of possible values could of course be folded into phrase categories, avoiding online computation. But folding features into categories often blows up the grammar size unacceptably, in which case online computation as described here is advisable.

[2] The estimates I used for hardware coefficients are as follows: Sparcstation 1: 1.0 (Fidditch, Scisor?, Clarit?—hardware is not specified in Scisor and Clarit citations; I have assumed Sparcstation 1), Siemens BS2000: 1.0 (Copsy—estimate is highly uncertain), Sparcstation

run at less than 1 w/s (Tacitus: 0.12? w/s [5]). "Skimming" parsers run at 20–50 w/s (Fastus: 23 w/s [3], Scisor: 30? w/s [6], Clarit: 50? w/s [4]). Deterministic parsers can be more than an order of magnitude faster (CG: 410 w/s [12], Fidditch: 1200 w/s (Hindle, p.c.), Cass2: 1300–2300 w/s, Copsy: 2700? w/s [11]). Cass2 is as fast as any parser in this class, with the possible exception of Copsy, for which the hardware adjustment is highly uncertain.

In measuring parser accuracy, there is a tension between evaluation with respect to some application (e.g., does parsing improve accuracy of speech recognition/information retrieval/etc.?) and evaluation with respect to a linguistic definition of "phrase." I think the resolution of this tension comes by distinguishing between the **accuracy** of the parser at performing the linguistic task it was designed for, and the **utility** of the linguistic task for particular applications.

In measuring accuracy, it is additionally important to distinguish between the task definition, which typically involves human linguistic judgments, and "test data," which is a record of a particular person's judgments. For example, a stylebook is a task definition, and a treebank is a record of judgments. Generally, a task definition leaves ample room for **interjudge variance**—though the degree to which interjudge variance compromises the validity of treebank-based evaluations is rarely appreciated. To reduce interjudge variance, we may make the task definition more detailed, but that also generally makes the task definition more arbitrary and less useful for comparison of parsers. To be broadly useful, it is in my opinion better to develop a collection of small, specialized "benchmarks" rather than a single large stylebook, so that evaluation participants can pick and choose those tasks that best match their systems' actual design criteria.

We have performed a preliminary evaluation of the parser described here. I wrote a brief stylebook defining "chunks", intended as a benchmark for one part of the output of a partial (or full-scale) parser. I hand-labelled a random sample of corpus positions with the category and end-point of the chunk (if any) starting at that corpus position. This permits us to estimate correlation between my judgments and the parser's "judgments" without creating a large treebank. A second human judge (Marc Light) performed the same task, to permit us to gauge interjudge reliability.

ELC: 1.25 (Cass2), Sparcstation 2: 1.7 (Tacitus?, Fastus), Sparcstation 10: 3.8 (CG).

(4)

		cass2	marc
sample size	N	1000	
answers[3] in common	X	921	934
chunks in tst	t	390	381
chunks in std	s	394	
chunks in common	x	343	348
per-word accuracy	X/N	$92.1 \pm 1.7\%$[4]	$93.4 \pm 1.5\%$
precision	x/t	$87.9 \pm 3.2\%$	$91.3 \pm 2.8\%$
recall	x/s	$87.1 \pm 3.3\%$	$88.3 \pm 3.2\%$

What is immediately striking is that the difference between parser and human performance on this test is barely significant. That fact would not have emerged if my mark-up had simply been accepted as the standard. Clearly, there is room for improvement in both the grammar (to improve parser accuracy) and stylebook (to reduce interjudge variance).

4 Not only Faster but also More Accurate

By way of closing remarks, I would like to address the motivation for partial parsing. The common view is that a parser such as that described here trades off accuracy for speed, compared to an exhaustive-search parser. But under certain reasonable assumptions about English, partial parsers—in particular, parsers using the longest-match rule—may be not only faster but also *more* accurate than exhaustive-search parsers—in particular, stochastic context-free parsers.

Consider the grammar

(5)
$$
\begin{array}{lll}
\mathrm{S} & \rightarrow & \mathrm{b\ A\ B} \mid \mathrm{c\ C\ A} \mid \mathrm{d\ B\ D} \\
\mathrm{A} & \rightarrow & \mathrm{a} \mid \mathrm{a\ a} \\
\mathrm{B} & \rightarrow & \mathrm{a} \mid \mathrm{a\ a} \\
\mathrm{C} & \rightarrow & \mathrm{a} \mid \mathrm{a\ a} \mid \mathrm{a\ a\ a} \\
\mathrm{D} & \rightarrow & \mathrm{a} \mid \mathrm{a\ a}
\end{array}
$$

Grammar (5) generates a finite language. Assume that each parse tree occurs with equal frequency, with the exception that a longest-match rule resolves ambiguities. That is, the parse-trees (6) are excluded, inasmuch as, for each parse in (6), there is an alternative parse in which the middle child covers more of the input.

[3] The sample consists of corpus positions; X is a random variable whose values (the "answers") are "chunk of category c and length k" or "no chunk". Per-word accuracy is the percentage of correct answers. Precision and recall consider only the subsample in which there is actually a chunk in the test or standard, respectively.

[4] The plus-minus figures represent a 95% confidence interval, using a normal approximation to the binomial.

(6) $[_S$ b $[_A$ a$]$ $[_B$ a a$]]$
$[_S$ c $[_C$ a$]$ $[_A$ a a$]]$
$[_S$ c $[_C$ a a$]$ $[_A$ a a$]]$
$[_S$ d $[_D$ a$]$ $[_B$ a a$]]$

If our training corpus contains each parse tree with equal frequency, excluding the parse trees (6), the maximum-likelihood estimate for rule probabilities is as follows:

(7)
$$\begin{array}{lcl}
S & \rightarrow & \text{b A B } (3/10) \mid \text{c C A } (2/5) \mid \text{d B D } (3/10) \\
A & \rightarrow & \text{a } (4/7) \mid \text{a a } (3/7) \\
B & \rightarrow & \text{a } (1/2) \mid \text{a a } (1/2) \\
C & \rightarrow & \text{a } (1/4) \mid \text{a a } (1/4) \mid \text{a a a } (1/2) \\
D & \rightarrow & \text{a } (2/3) \mid \text{a a } (1/3)
\end{array}$$

Now, because of the longest-match constraint that the language obeys, there is a deterministic longest-match parser that performs perfectly, whereas the best SCFG (7) parses the sentence *baaa* incorrectly. Namely, the parse tree $[_S$ b $[_A$ a$]$ $[_B$ a a$]]$ has probability 4/14 according to grammar (7), whereas the correct parse tree, $[_S$ b $[_A$ a a$]$ $[_B$ a$]]$, has probability 3/14.

Intuitively, this is a consequence of "longest match" being an essentially cross-derivational (equivalently: context-sensitive) notion. If English empirically observes a longest-match constraint, a deterministic longest-match parser can be more accurate than a stochastic CF parser. That English does adhere to a longest-match constraint is suggested by many garden path sentences. For example, *The emergency crews hate most is domestic violence* is a garden path because we strongly prefer the longest initial NP, *the emergency crews,* and overlook the alternative that is in this case correct, namely, *the emergency [which] crews hate most is domestic violence.*

5 Conclusion

I have presented a technique, finite-state cascades, for producing fast, robust parsers for unrestricted text. The technique has been applied to English and German, and is being used in a project for inducing subcategorization frames and selectional restrictions in these languages. The parser consists of a pipeline of finite-state recognizers. Key concepts are easy-first parsing, islands of certainty, and containment of ambiguity. Finite-state cascades can be extended to include feature assignment and output of "linguistic" structure at little cost in efficiency.

I have also drawn some distinctions that I believe are important for evaluation, though not widely appreciated—in particular, the distinction between accuracy and utility, and the distinction between task specification and test data, along with the importance of measuring and controlling interjudge variance. To the latter end, I propose using a collection of benchmarks instead of a single stylebook in order to control interjudge variance while maintaining broadness of relevancy.

References

[1] Steven Abney. Rapid incremental parsing with repair. In *Proceedings of the 6th New OED Conference: Electronic Text Research*, pages 1–9, Waterloo, Ontario, October 1990. University of Waterloo.

[2] Steven Abney. Parsing by chunks. In Robert Berwick, Steven Abney, and Carol Tenny, editors, *Principle-Based Parsing*. Kluwer Academic Publishers, 1991.

[3] Douglas E. Appelt et al. SRI international FASTUS system MUC-4 test results and analysis. In *Proceedings, Fourth Message Understanding Conference (MUC-4)*, pages 143–147, San Mateo, CA, 1992. Morgan Kaufmann.

[4] D. Evans, K. Ginther-Webster, M. Hart, R. Lefferts, and I. Monarch. Automatic indexing using selective nlp and first-order thesauri. In *Proc. of RIAO 91 (Barcelona)*, pages 624–643, 1991.

[5] Jerry R. Hobbs et al. SRI International: Description of the FASTUS system used for MUC-4. In *Proceedings, Fourth Message Understanding Conference (MUC-4)*, pages 268–275, San Mateo, CA, 1992. Morgan Kaufmann.

[6] Paul S. Jacobs. To parse or not to parse: Relation-driven text skimming. In *COLING 90, vol. 2*, pages pp. 194–198, 1990.

[7] Kimmo Koskenniemi. Finite-state parsing and disambiguation. In *COLING-90*, pages 229–232, 1990.

[8] Kimmo Koskenniemi, Pasi Tapanainen, and Atro Voutilainen. Compiling and using finite-state syntactic rules. In *COLING-92*, pages 156–162, 1992.

[9] Michael Lesk. Lex: a lexical analysis program generator (?). In *UNIX Programming Utilities and Libraries*. Publisher unknown, 1978?

[10] Emmanuel Roche. *Analyse Syntaxique Transformationnelle du Francais par Transducteurs et Lexique-Grammaire*. PhD thesis, Université Paris 7, 1993.

[11] Christoph Schwarz. Automatic syntactic analysis of free text. *JASIS*, 41(6):408–417, 1990.

[12] Atro Voutilainen. NPtool, a detector of English noun phrases. In *Proceedings of the Workshop on Very Large Corpora*, pages 48–57, 1993.