Parsing with Treebank Grammars: Empirical Bounds, Theoretical Models, and the Structure of the Penn Treebank

Dan Klein and **Christopher D. Manning**

Computer Science Department
Stanford University
Stanford, CA 94305-9040
{klein, manning}@cs.stanford.edu

Abstract

This paper presents empirical studies and closely corresponding theoretical models of the performance of a chart parser exhaustively parsing the Penn Treebank, using the Treebank's own CFG grammar. We show how performance is dramatically affected by rule representation choices, and analyze the value of top-down vs. bottom-up strategies. We discuss grammatical saturation, including analysis of the strongly connected components of the phrasal nonterminals in the Treebank, and present evidence that as sentence length increases, the effective grammar rule size increases as regions of the grammar are unlocked, yielding super-cubic observed time behavior in some configurations.

1 Introduction

This paper began from examining the practical performance of an exhaustive chart parser using an untransformed treebank grammar (Charniak, 1996) on the Penn Treebank. Our initial experiments with such a setup yielded quite surprising results. The parser operates with grammar rules implemented as finite state automata (FSA), and for greater efficiency, we had encoded the expansions of different categories via a trie FSA. While the worst-case time bound of the parser is $O(n^3)$ in the sentence length n, we observed that the actual time performance with this grammar was super-cubic (over the range of sentence lengths present in the Penn treebank). It initally seemed plausible that as span sizes grew, the number of passive edges over those spans would increase because longer spans of words would be more ambiguous. However, the number of passive edges over a span is almost constant (and so the number of passive edges grows roughly quadratically in n with the number of spans). This led us to look more closely at the structure of treebank grammars derived from the Penn Treebank. The resulting analysis builds on the presentation of Charniak (1996), but extends it by elucidating the structure of non-terminal interrelationships in the Penn treebank grammar. On the basis of these studies, we build simple theoretical models which closely predict the observed performance for several parameters, and, in particular, explain the originally observed super-cubic behavior.

2 Background

The sentences were parsed using an implementation of the chart-parsing algorithm presented in (Self, submitted). In that paper, we present a theoretical analysis showing an $O(n^3)$ worst-case time bound for arbitrary probabilistic context-free grammars. In what follows, we do not make use of the probabilistic aspects of the grammar or parser; all of the results are for exhaustive parsing and apply simply to the treebank CFG. The hardware was an Intel Pentium III box with a 700MHz processor, and we used up to 2GB of RAM for very long sentences or very poor parameters.

We used treebank grammars induced directly from the local trees of the Penn Treebank (Marcus et al., 1993) (release 3). The entire WSJ section was used for training. For each length and parameter setting, 25 sentences evenly distributed through the treebank were parsed. Since we were parsing sentences from among those from which our grammar was derived, coverage was never an issue. Every sentence parsed had at least one parse – the parse with which it was originally observed.[1]

3 Parameters

The parameters we varied were:

- Grammar Transforms: **NO-TRANSFORM** or NO-EMPTIES
- Grammar Rule Encoding: **TRIES** or LISTS
- Rule Introduction Stratgies: TOP-DOWN or **BOTTOM-UP**

The default settings are shown above in bold face.

[1]Effectively "testing on the training set" like this would be invalid if we wished to present performance results such as precision and recall, but it is in no way a problem for the present experiments, which focused solely on the parser load and grammar structure.

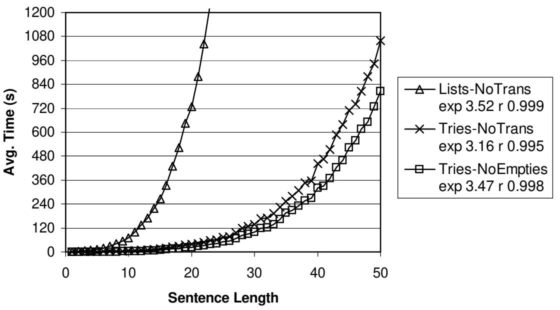

Figure 1: Time to parse sentences using various parameters.

3.1 Grammar Transforms

In all cases, the grammar was directly induced from Penn treebank trees. For the parameter setting NO-TRANSFORM, no modification to the trees was made (except for stripping functional tags, and crossreferencing annotations). In particular, empty nodes (represented as -NONE- in the treebank) were turned into rules that generated nothing in the grammar, and there was no collapsing of categories (such as PRT and ADVP), as is often done in parsing work (Collins, 1997, etc.). For the setting NO-EMPTIES, empties were removed by pruning nonterminals above them which cover no words from trees, and then using the resulting local trees to induce the grammar. Unary rules play an important role in our results, and so one might want to also investigate a second transformation in which one removed all unary rules, but we do not investigate such a grammar here.[2]

3.2 Grammar Rule Encoding

The parser operates on FSA grammar representations. We compiled grammar rules into FSAs in two ways: LISTS or TRIES. For LIST encodings, each local tree type was encoded in its own, linearly structured FSA, corresponding to Earley (1970)-style dotted rules. For TRIES, there was one FSA per category, encoding all rule types with that category as an LHS as described in, for example, (Allen, 1995). The FSAs used by the parser are weighted, but this is not relevant to the present work.

3.3 Settings Used

We do not report results for all possible combinations of these settings. Rather, we take the exhaustive bottom-up parser using an untransformed grammar with trie rule encodings to be the basic form of the parser. Except where noted, we will discuss how each factor affects this baseline, as most of the effects are orthogonal. When we name a setting, any omitted parameters are assumed to be the defaults.

[2]Such a grammar would greatly reduce the chart size, for the reasons discussed below, but is unlikely to be easy to use or evaluate in linguistic applications. One loses many intermediate nodes, and either has to allow phrasal categories to rewrite as words, if one folds unary rules downwards, or to have 'mixed' rules like S $\rightarrow$ DT VP, if one folds unary rules upwards.

4 Observed Performance

In this section, we describe the observed performance of the parser for various settings. We frequently describe things in terms of the following:

- **span:** a range in the chart, e.g., [4,17]
- **edge:** a category over a span, e.g., NP:[4,17]
- **traversal:** a way of making an edge from another active and passive edge, e.g., NP:[4,17] $\Leftarrow$ (NP$\rightarrow$DT.NN:[4,12] + NN:[12,17])

4.1 Time

The parser has an $O(SCn^3)$ theoretical time bound, where n is the number of words in the sentence to be parsed, C is the number of nonterminal categories and S is the number of (active) states in the FSA encoding of the grammar. The time bound is derived from the number of traversals processed by the parser, each taking $O(1)$ time.

In figure 1 we see the average time taken per sentence length for the several settings, with the empirical exponent (and correlation r-value) to the right. In figure 2, we see the average number of traversals processed. Notice that the LIST setting is dramatically more expensive in both time and traversals, and that all of them show time growth greater than $O(n^3)$.

Although, of course, $O(n^3)$ is simply an asymptotic bound, we will see that there are good explanations for the observed behavior, both when it is super-cubic and when it is not. It turns out that there are two primary causes for the super-cubic time values. The first is theoretically uninteresting, though pratically very real. The parser we use is implemented in Java, which uses garbage collection for memory management. Even when there is plenty of memory for a parse's primary data structures, "garbage collection thrashing" can occur as temporary objects cause increasingly frequent reclamation. This accounts for the deviation of the processing time from the traversal counts, and we will henceforth confine our investigation to the latter.

The more interesting reason for the widely varying exponential behavior is that the actual number of traversals to process can grow much faster (or slower) than the predicted bounds, depending on the parameters. In figure 2, the number of traversals is itself growing at around $O(n^{3.3})$ in this range for the TRIE NO-EMPTIES case. The general reason for such rapid traversal growth is that the S term, representing the size of the grammar, effectively grows with n, as we will describe in section 7. Even for settings which do not exhibit super-cubic traversal growth, the effective S term still grows with n, however the traversal exponents are drawn below 3 by other factors (see sections 6 and 7).

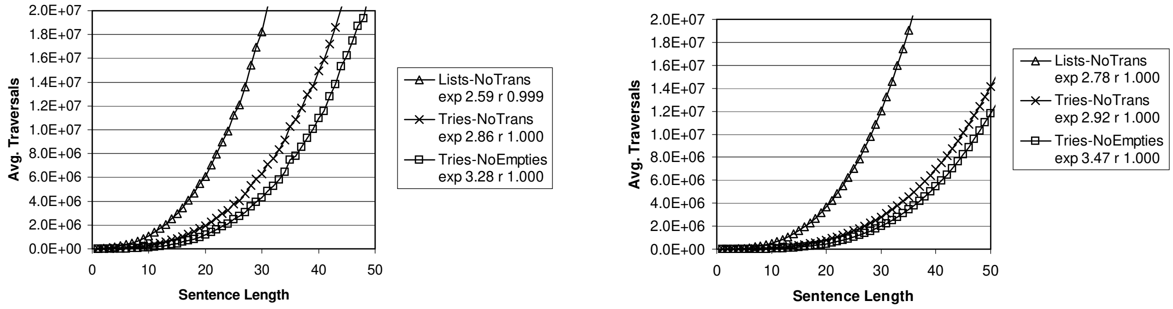

Figure 2: The average number of traversals for sentences of a given length as observed in practice (left), and as predicted by our models (right).

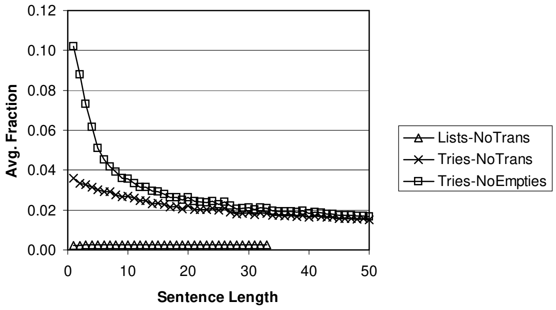

Figure 3: Fraction of edges which were passive for a given sentence length.

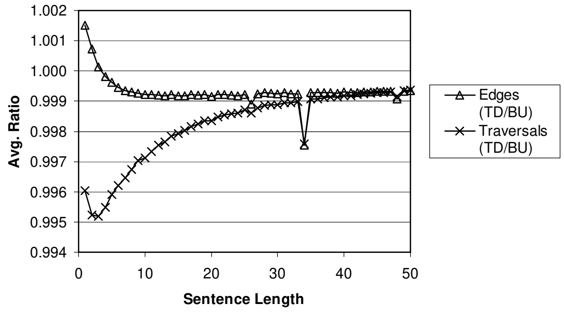

Figure 4: For the given data types, the ratio of the quantity produced under a top-down strategy over the quantity produced under a bottom-up strategy. (Shown for TRIE NO-TRANSFORM, others are similar.)

4.2 Memory

The memory bound of the parser is $O(Sn^2)$. Since the parser is running in a garbage-collected environment, it is hard to distinguish required memory from utilized memory. However, unlike time and traversals which in practice diverge greatly, memory requirements match the number of edges in the chart almost exactly, since the large maintained data structures are all proportional in size to the number of edges.[3]

Almost all edges stored are active edges (see figure 3), of which there can be $O(Sn^2)$: one for every grammar state and span. Passive edges, of which there can be $O(Cn^2)$, one for every category and span, are the minority. This is because, while C is bounded above by 27 in the treebank[4] (for spans ≥ 2), S numbers in the thousands, even in practice (see figure 11).

Since the total number of edges is roughly equal to the total number of active edges, the required memory growth rates will be implicitly modeled in section 7 when we model active edge counts.

4.3 Top-Down vs. Bottom-Up

Figure 4 shows the effect of using top-down and bottom-up strategies on total edges and traversals. There is some extremely minimal savings in traversals due to top-down filtering effects, but there is a corresponding penalty in edges as rules whose left-corner cannot be built are introduced. Given the extremely unrestrictive nature of the treebank grammar, it is not very surprising that top-down filtering provides extremely little benefit. However, this is a useful observation about real world parsing performance. The advantages of top-down chart parsing in providing grammar-driven prediction are often advanced (e.g., Allen 1995:66), but in practice we find almost no value in this. While in some part this is doubtless due to errors in the treebank, a large part of this just reflects the true nature of broad coverage grammars: e.g., once you do such things as allowing adverbial phrases almost anywhere, and allowing PPs, (participial) VPs, and (temporal) NPs to be adverbial phrases, along with phrases headed by adverbs, then there is very little useful top-down control left. With such a permissive grammar, the only real constraints to be found, as we argue later, are in the POS tags which anchor the local trees. For the remainder of the paper, therefore, we focus on bottom-up settings.

[3]Note that a standard chart parser might conceivably require storing more than $O(E)$ traversals on its agenda, but ours provably never does.

[4]This count is the number of phrasal categories with the introduction of a ROOT label for the unlabeled top treebank nodes.

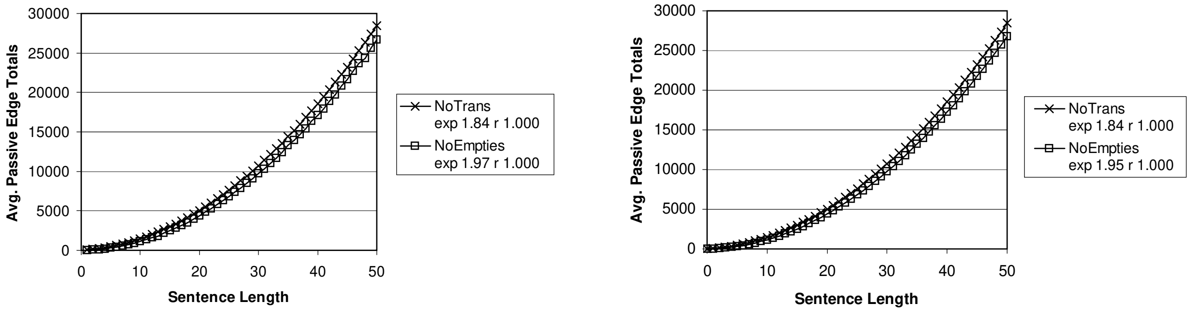

Figure 5: The number of passive edges processed in practice (left), and predicted by our models (right).

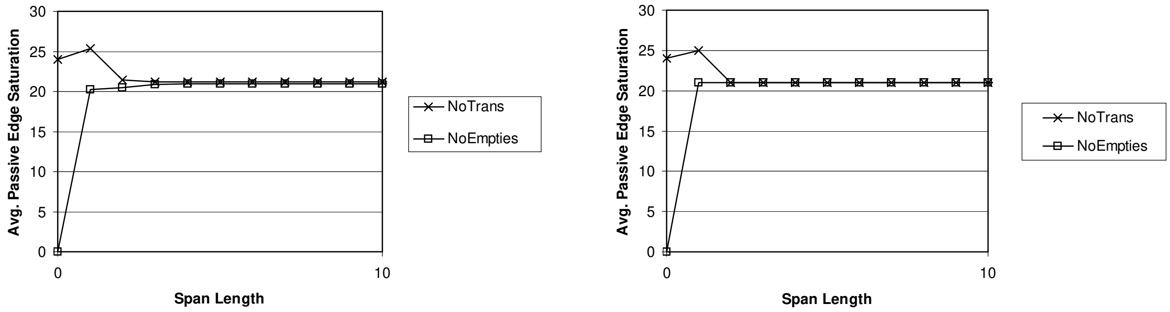

Figure 6: The average number of passive edges for a span of a given length as processed in practice (left), and as predicted by our models (right).

5 Saturation

The number of spans for a given sentence length is fixed and trivially countable. Thus, to be able to evaluate and model the total edge counts, we look to the number of edges over a given span.

Definition 1 *The* passive *(or* active*) saturation of a given span is the observed number of passive (or active) edges over that span.*

An interesting fact is that the saturation of a span is, for the treebank grammar and sentences, essentially independent of what size sentence the span is from and where in the sentence the span begins. Thus, for a given span size, we report the average over all spans of that size occuring anywhere in any sentence parsed.

6 Passive Edges

The total growth and saturation of passive edges is relatively easy to describe. Figure 5 shows the total number of passive edges by sentence length and figure 6 shows the saturation as a function of span size. Note that rule encoding methods cannot change which passive edges will occur for a given span.

The most interesting aspect of figure 6 is not that it is relatively constant, at least for spans greater than size 2, but that the saturation values are so *large*[5] (see section 6.1).

For each transformation setting, one can see that the passive counts can be summarized by three saturation numbers: zero-spans (empties) sat_0, one-spans (words) sat_1, and all larger spans (categories) sat_2. Taking averages directly from the data, we have our first model, shown on the right in figure 6. While it is not a particularly explanatory model yet, it describes the difference in passive totals quite well (shown to the right in figure 5. The totals are given by

$$psat(n) = sat_2\frac{(n-1)n}{2} + sat_1 n + sat_0(n+1)$$

We correctly predict that the passive edge total exponents will be slighly less than 2.0 in both cases, and lower for NO-TRANSFORM than for NO-EMPTIES. This is because the linear term in the equation is significant over these sentence length and drags down the exponent. The linear term is larger for NO-TRANSFORM and therefore drags the exponent down more.[6]

6.1 Treebank Grammar Structure

Although we have now reproduced the passive totals, we have not addressed the key issues surrounding passive saturation. Why do the counts for $size \geq 2$ not depend on $size$, and why are they so large? The reason is

[5] The maximum possible passive saturation for any span greater than one is equal to the number of phrasal categories in the treebank grammar: 27. However, empty and size-one spans can additionally be covered by POS tag edges.

[6] Note that, over these values of n, even the simple sum $\sum n$ has a best-fit exponent of only 1.95.

ADJP ADVP FRAG INTJ NAC
NP NX PP PRN QP ROOT RRC
S SBAR SBARQ SINV SQ UCP
VP WHADVP WHNP WHPP X

Figure 7: The empty-reachable set for the NO-TRANSFORM grammar.

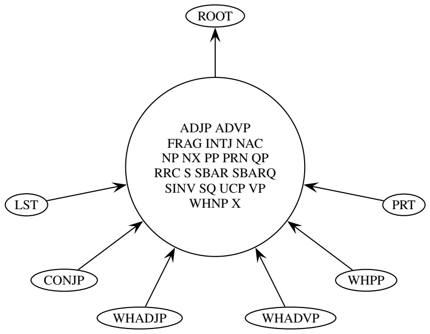

Figure 8: A same-span reachability graph for the NO-TRANSFORM grammar.

because, even with the transformed grammar, most categories are reachable from most other categories using rules which can be applied over a single span. Once you get one of these kinds of passive edges over a span, you will get the rest.

We now formalize this.

Definition 2 *A category X is* empty-reachable *in a grammar G if X can be built using only empty terminals.*

Definition 3 *A category Y is* same-span-reachable *from a category X in a grammar G if Y can be built from X using a parse tree in which, aside from at most one instance of X, every node not dominating that instance is an instance of an empty-reachable category.*

The empty-reachable set for the untransformed grammar is shown in figure 7.[7] This set has size 23. These 23 categories, plus the tag -NONE-, are the source of the saturation of 24 for zero-spans.

The same-span-reachability relation induces a graph over the 27 non-terminal categories. The strongly-connected component (SCC) reduction[8] of that graph is shown in figures 8 and 9. Unsurprisingly, the largest SCC, which contains most "common" categories (S, NP, VP, PP, etc) is slightly larger for the NO-TRANSFORM grammar, since the empty-reachable set is non-empty. However, note that even for NO-TRANSFORM, the largest SCC is smaller than the empty-reachable set since empties provide direct entry into some of the lower SCCs, in particular because of WH-gaps. The large SCCs also explain the relative independence of passive saturation on span size. Once any category in the SCC is found, all will be found, as well as all categories reachable from that SCC.

[7]The set of phrasal categories used in the Penn Treebank is documented in Marcus et al. (1993) or Manning and Schütze (1999, 413).

[8]Implied arcs have been removed for clarity. The relation is in fact the transitive closure over this graph.

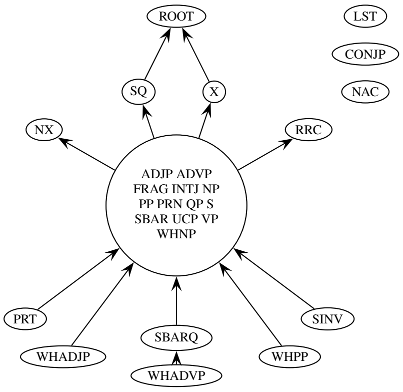

Figure 9: The same-span-reachable set for the NO-EMPTIES grammar.

	NO-TRANSFORM		NO-EMPTIES	
TRIES	Rules:	27	Rules:	27
	States:	17298	States:	17011
LISTS	Rules:	17357	Rules:	17032
	States:	80120	States:	78233

Figure 11: Grammar sizes.

7 Active Edges

Active edges are the vast majority of edges and essentially determine (non-transient) memory requirements. Figure 10 shows the total active edges by sentence size for various settings. This growth is sub-quadratic for list grammars, but nearly $O(n^{2.4})$ for trie grammars.[9]

The maximum active saturation for any span is equal to the number of active grammar states – the number of states in the grammar FSAs which can give rise to active edges. For list grammars, this number is the sum of the lengths of all rules in the grammar. For trie grammars, it is the number of unique rule prefixes (including the LHS) in the grammar.

For the grammars used, figure 11 gives the number of rules and active states. Note that the maximum number of active states is dramatically larger for list representations since common rule prefixes are duplicated many times. Since (see section 7.1) states which are earlier in

[9]Charniak et al. (1998, 127) cite the figure that "exhaustively parsing maximum-40-word sentences from the Penn II Treebank requires an average of 1.2 million edges per sentence." Given that the average sentence will be around 20 words, this number seems to sit slightly oddly between our list- and trie-grammar curves. This may reflect use of a pruned list grammar or the fact that in a probabilistic chart parser most edges which are active/passive for a categorical chart parser must be split into two separate edges.

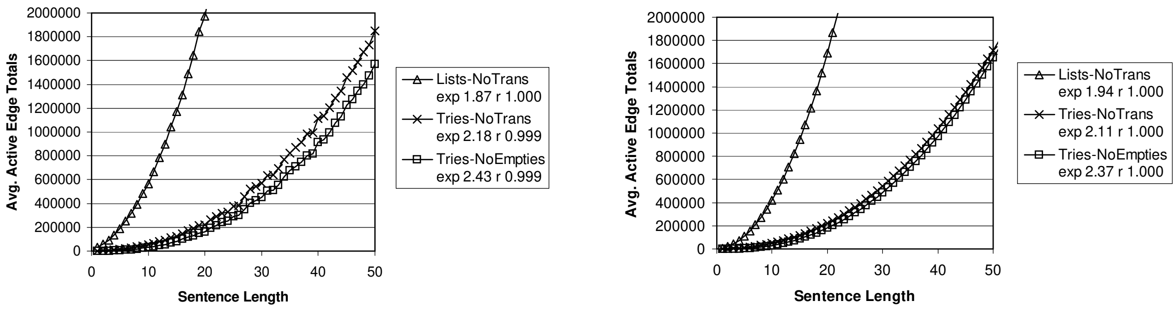

Figure 10: The average number of active edges for sentences of a given length as observed in practice (left), and as predicted by our models (right).

a rule are much more likely to match a span, the list representation is redundant in a highly inefficient way.

The most interesting puzzle is the active saturation curves shown in figure 12, from which the active total curves can be directly derived. Unlike passive saturation, which was relatively constant in span size, active saturation quite clearly grows with span size. As spans become larger, saturation additively increases at roughly the same rate for both lists and tries, however the initial values for lists are much higher. In the next section, we model these curves.

7.1 Active Saturation

In the Treebank grammar, active states (which correspond to a prefix of a rule) are a mix of POS tags and phrasal categories. Given the large mutual reachability regions seen in section 6.1, phrasal categories, to a first approximation, might as well be wildcards, especially if empties are present. However, POS tags over words are, in comparison, very restricted. Tags in rules must actually match a word in the span.

More precisely, consider an active state a in the grammar and a span s. In our encodings, there is some, possibly empty, list L of labels that must be matched over s before an active edge with this state can be constructed over that span. Assume that the categories in L can match any span (or any non-zero span if NO-EMPTIES. Therefore, they do not constrain whether a can match s. The real issue is whether the tags in L will match words in s. Assume therefore, that a random tag matches a random word with a fixed probabiliy p, independently of where the tag is in the rule and where the word is in the sentence.[10] Assume further, that although tags occur more often than categories in rules (63.9% of rule items are tags), given a fixed number of tags and categories, all permutations of them are equally likely to appear as rules.[11]

[10]This is of course false; in particular, tags at the end of rules have much lower entropy, disproportionately tending to be punctuation tags.

[11]This is also false; tags occur slightly more often at the beginnings of rules and less often at the ends.

Under these assumptions, the probability that a particular rule or active state is in the treebank grammar will not depend at all on the individual tags and categories in the rule, nor even on their order, but only on the num t of tags and c of categories in that rule or state. For an active state a, call this pair $\sigma(a) = (t, c)$ the *signature* of a. For a given signature σ, we count the number of active state types in the grammar which have that signature and call that number $count(\sigma)$.

Now, take a state a of signature (t, c) and a span s. If we align the tags in a with words in s and align the categories in a with spans of words in s, then provided the categories align with a non-empty span (for NO-EMPTIES) or any span at all (for NO-TRANSFORM), then the question of whether this alignment of a with s matches is determined entirely by the t tags. However, with our assumptions, the probabilty that a randomly chosen set of t tags matches a randomly chosen set of t words is simply $p^{|t|}$.

We then have an expression for the chance of matching a specific alignment of an active state to a specific span. Clearly, there can be many alignments which differ only in the spans of the categories, but line up the same tags with the same words. However, there will be a certain number of unique ways in which the words and tags can be lined up between a and s. If we know this number, we can calculate the total probabilty that there is some alignment which matches. For example, consider the state NP $\rightarrow$ NP CC NP . NP (which has signature (1,2)) over a span of length n with empties available. The NPs can match any span, so there are n alignments which are distinct from the standpoint of the CC tag – it can be in any position. The chance that some alignment will match is therefore $1 - (1 - p)^n$, which, for small p is roughly linear in n. It should be clear that for an active state like this, the longer the span, the more likely it is that this state will be found over that span.

It is unfortunately not the case that all states with the same signature will match a span length with the same probability. For example, the state NP $\rightarrow$ NP NP CC . NP has the same signature, but *must* align the CC with the final element of the span. A state like this will not become more likely with span increases (at least in our model).

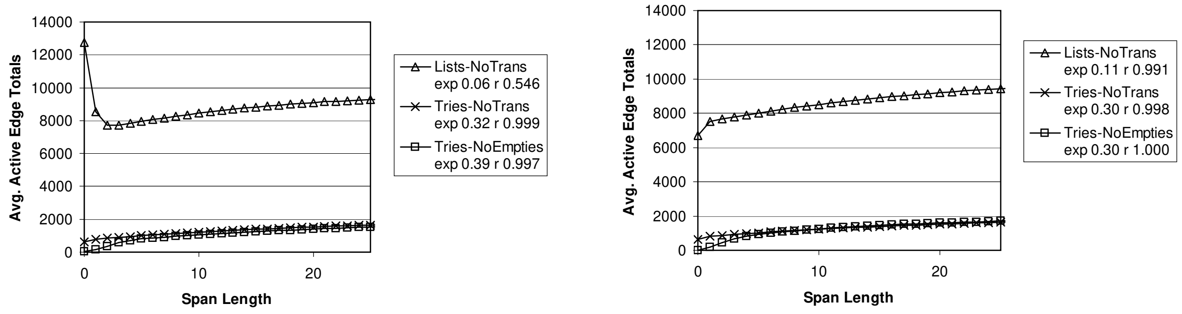

Figure 12: The average number of active edges for a span of a given length as processed in practice (left), and as predicted by our models (right).

With some straightforward but space-consuming combinatorics, we can, however, calculate the expected chance that a random rule of a given signature will match a given span length. Since we know how many states have a given signature, we can calculate the total active saturation $asat(n)$ as

$asat(n) = \sum_{\sigma} count(\sigma) E_{a \in \sigma}[P(match(a, n))]$

This model has two parameters. First, there is p which we estimate directly by looking at the expected match between the distribution of tags in rules and the distribution of tags in the Treebank text (which is around 1/17.65). No factor for POS tag ambiguity was used, another simplification.[12] Second, there is the map *count* from signatures to a number of active states, which was read directly from the compiled grammars.[13]

We used this model to predict active saturation for span sizes as shown in figure 12. Note that the model, though not perfect, exhibits the qualitative differences between the settings.[14] In particular:

- In TRIES NO-EMPTIES the initial (low-span) saturation is lower than in TRIES NO-TRANSFORM since short spans in the former case can match only signatures which have both t and c small, while in the latter only t needs to be small. Therefore, the several hundred states which are reachable only via categories all match every span starting from size 0 for NO-TRANSFORM, but are accessed only gradually for NO-EMPTIES. However, for larger spans, the behavior converges.
- In LISTS, the initial number of available states is huge, due to the fact that most of the states which are available early for trie grammars are precisely the ones duplicated up to thousands of times in the list grammars. However, the additive gain over the initial states is roughly the same for both, as after a few items are specified, the tries become sparse.
- The trie exponents are around 0.3 while the list exponents are around 0.1.[15]
- The actual magnitudes of the saturations are surprisingly well predicted, suggesting that in fact this model captures the essential behavior.

One noticeable missing prediction is the spike at span 0 for LISTS. This is because not all categories actually match all spans, and there are significantly more categories which do match empty spans.

8 Traversals

Now that we have models for active and passive edges, we model for traversal counts as well. We assume that the chance for passives and actives to combine into a traversal is a constant representing how likely an arbitrary active state is to have a continuation with a label matching an arbitrary passive state. List rule states have only one continuation, while trie rule states have an average of about 1.7.[16] Making another uniformity assumption, we assume that the chance is the average continuation degree divided by the total number of passive labels, categorical or tag (73).

In figure 2 we gave graphs and exponents for the various parameters, both observed and predicted. Our model correctly predicts the approximate values and major qualitative facts, including:

- For LISTS, the observed exponent is lower than for TRIES, though the total number of traversals is dramatically higher.
- Only TRIES NO-TRANSFORM exhibits super-cubic traversal totals. Despite their similar active and passive exponents, TRIES NO-EMPTIES and TRIES NO-TRANSFORM vary substantially in traversal growth due to the "early burst" of active edges which gives TRIES NO-TRANSFORM significantly more edges over short spans than its power law would predict.

[12] In general, the p we used was lower for not having modeled tagging ambiguitiy, but higher for not having modeled the fact that the SCCs are not of size 27.

[13] Note that the primary effect on the saturation model of removing empties is to change the number of possible ways in which a state can align with a span, not to change the rule counts substantially.

[14] And does so without any "tweakable" parameters.

[15] Note that the list curves do not compellingly suggest a pure power law model.

[16] This is obviously a simplification as well, since the shorter prefixes that tend to have higher continuation degrees are on average also a larger fraction of the active edges.

This excess leads to a sizeable quadratic addend in the number of transitions, causing the average best-fit exponent to drop.

9 Conclusions

We built simple but accurate models on the basis of two observations. First, passive edge saturation is relatively constant in span size, but large due to large same-span-reachability SCCs among phrasal categories in the grammar. Second, active saturation grows with n because as spans increase, the tags in a given active edge are more likely to find a matching arrangement over a span. Combining these models, we demonstrated that a wide rangle of empirical qualitative and quantitative behaviors of an exhaustive parser could be derived, including the potential super-cubic traversal growth over sentence lengths of interest.

References

James Allen. 1995. *Natural Language Understanding*. Benjamin Cummings, Redwood City, CA.

Eugene Charniak, Sharon Goldwater, and Mark Johnson. 1998. Edge-based best-first chart parsing. In *Proceedings of the Sixth Workshop on Very Large Corpora*, pages 127–133.

Eugene Charniak. 1996. Tree-bank grammars. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI '96)*, pages 1031–1036.

Michael John Collins. 1997. Three generative, lexicalised models for statistical parsing. In *ACL 35/EACL 8*, pages 16–23.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 6(8):451–455.

Christopher D. Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press, Boston, MA.

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

Self. submitted. xxx. In *yyy*.