An Empirical Evaluation of HHMM Parsing Time*

Tim Miller
University of Minnesota
tmill@cs.umn.edu

William Schuler
University of Minnesota
schuler@cs.umn.edu

1 Introduction

Current state of the art speech recognition systems use very little structural linguistic information while doing word recognition. Some systems attempt to apply syntactic and semantic analysis to speech, but this is typically done in a pipelined approach, where there is thresholding done in between each stage. It would be advantageous to make use of information about higher level linguistic structure before doing any thresholding, so that uncertainty at different levels (acoustic, word level, syntax, semantics) can all be weighed simultaneously to recover the most likely global outcome.

However, the standard CYK parsing algorithm has cubic run-time complexity, which makes it difficult to use in streaming speech applications with unsegmented utterances. Some have proposed frameworks for parsing using time-series models such as hierarchical hidden Markov models (HHMMs), with an architecture similar to that of Murphy and Paskin (2001). These models have been proposed because there are algorithms for recovering a most likely sequence in linear time. However, transforming a grammar into a format usable by time series models like HHMMs increases the constants in the run-time complexity, and the practical effect of these transformations on run-time complexity has not been evaluated thoroughly.

This paper describes a system that does parsing using an HHMM, and shows preliminary experimental results on run-time for utterances of varying lengths with this system compared to a baseline PCFG parser.

*The material in this article is based upon work supported by NASA under award NNX08AC36A. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Aeronautics and Space Administration.

2 HHMM Parsing

HHMM parsing makes use of a graphical model architecture such that, for each word (time step), there is a 'stack' of random variables, which essentially are the stack for a non-deterministic shift-reduce parser.

Since the topology of the network cannot change over time, the HHMM requires a fixed size stack. The naive solution to determining a stack size would be to find the sentence in the training data with the largest stack requirements, and simply make the topology large enough to handle recursion of that depth. However, this highlights two problems: First, most of the stack depth would be unused at each step, although the algorithm would still have to spend time exploring potential uses of the stack. This leads to the second problem, which is that the algorithm for HHMM inference has a run-time that is exponential on the depth of the network (see Section 3), which means that it is in our best interests to limit the depth of the HHMM topology.

The following sections will describe both how an HHMM can be trained by a set of syntactically annotated trees to do parsing on unseen sentences, and how those trees can be transformed to minimize the depth required in the HHMM topology.

2.1 Right-corner transform

Input trees are first binarized as in Miller and Schuler (2008), and then transformed into *right-corner* trees using transform rules similar to those described by Johnson (1998). This right-corner

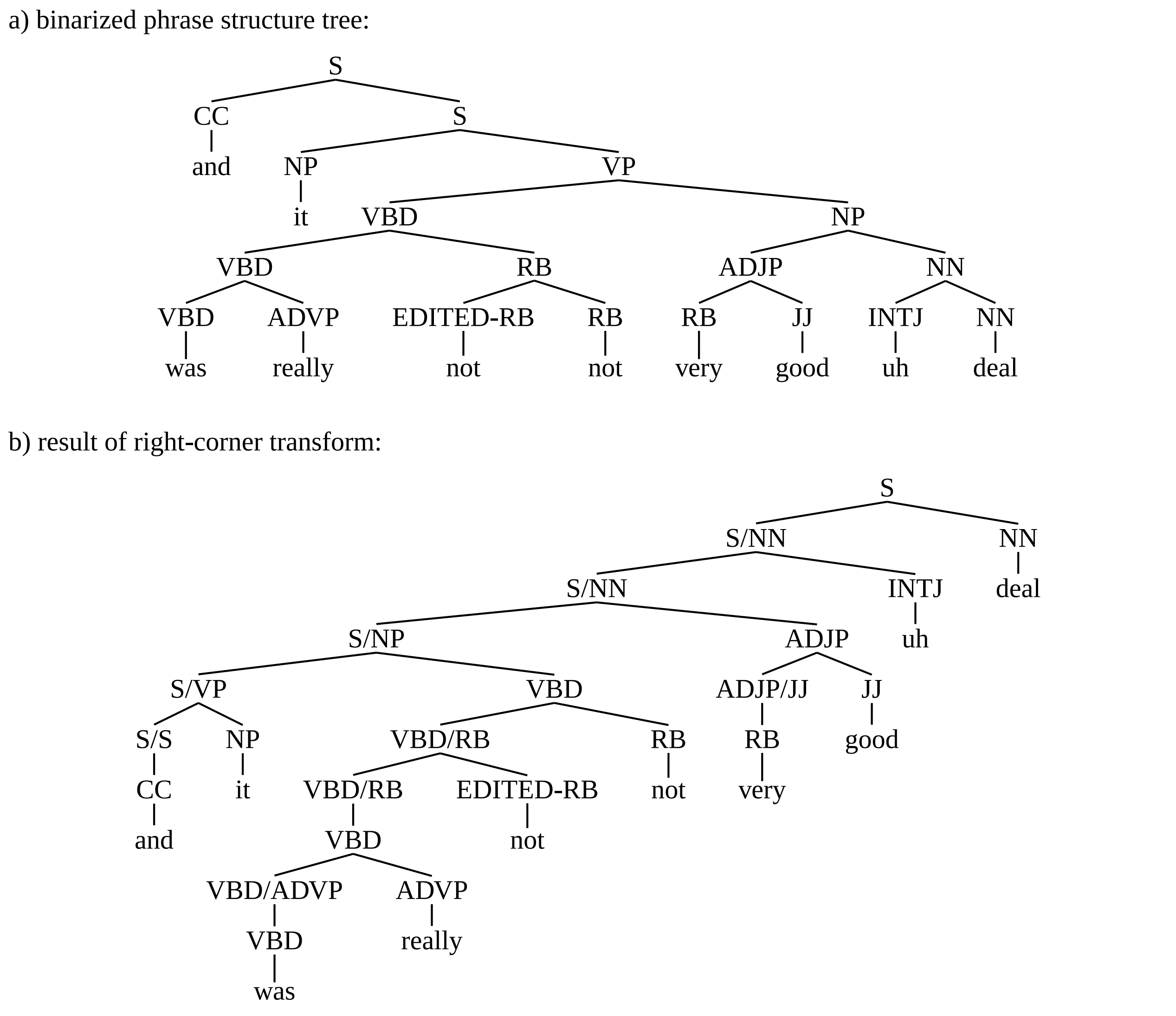
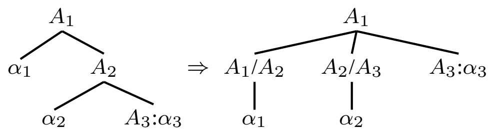
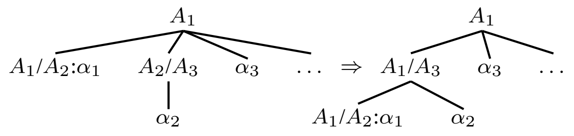


Figure 1: This figure shows a) a binarized phrase structure tree which serves as the input to the right corner transform, and b) a right-corner transform of this binarized tree.

transform is simply the left-right dual of a left-corner transform. It transforms all right recursive sequences in each tree into left recursive sequences of symbols of the form A_1/A_2, denoting an incomplete instance of category A_1 lacking an instance of category A_2 to the right.

Rewrite rules for the right-corner transform are shown below:[1]





Here, the first rewrite rule is applied iteratively (bottom-up on the tree) to flatten all right recursion, using incomplete constituents to record the original nonterminal ordering. The second rule is then applied to generate left recursive structure, preserving this ordering. An example of this transform applied to a binarized tree is shown in Figure 1.

This transform has the property of turning all right recursion into left recursion, while leaving left recursion intact. This is useful because left recursion

[1]All A_i denote nonterminal symbols, and all α_i denote subtrees; the notation $A_1{:}\alpha_1$ indicates a subtree α_1 with label A_1; and all rewrites are applied recursively, from leaves to root.

does not require stack space, since each word taken as input combines with an element that is popped off the stack to form a new constituent that becomes the new top of the stack. Center recursion, which notoriously causes problems in human processing as sentences become more deeply recursive, is also left intact. Contrary to left recursion, center recursion does take up space on the stack. Thus, this transform adheres to psycholinguistic models in which center recursion requires memory during processing while other types of recursion do not.

By transforming trees from the training set in this manner, we minimize the amount of stack space needed to represent utterances that humans realistically generate and understand.

2.2 Hierarchical HMMs

Right-corner transformed trees from the training set are then mapped to random variable positions in a hierarchical hidden Markov model, essentially a hidden Markov model (HMM) factored into some fixed number of stack levels at each time step.

HMMs characterize speech or text as a sequence of hidden states q_t (which may consist of speech sounds, words, or other hypothesized syntactic or semantic information), and observed states o_t at corresponding time steps t (typically short, overlapping frames of an audio signal, or words or characters in a text processing application). A most likely sequence of hidden states $\hat{q}_{1..T}$ can then be hypothesized given any sequence of observed states $o_{1..T}$, using Bayes' Law (Equation 2) and Markov independence assumptions (Equation 3) to define a full $\mathsf{P}(q_{1..T} \mid o_{1..T})$ probability as the product of a *Language Model (*Θ_L*)* prior probability $\mathsf{P}(q_{1..T}) \overset{\mathrm{def}}{=} \prod_t \mathsf{P}_{\Theta_\mathrm{L}}(q_t \mid q_{t-1})$ and an *Observation Model (*Θ_O*)* likelihood probability $\mathsf{P}(o_{1..T} \mid q_{1..T}) \overset{\mathrm{def}}{=} \prod_t \mathsf{P}_{\Theta_\mathrm{O}}(o_t \mid q_t)$:

$$\hat{q}_{1..T} = \underset{q_{1..T}}{\operatorname{argmax}}\, \mathsf{P}(q_{1..T} \mid o_{1..T}) \tag{1}$$

$$= \underset{q_{1..T}}{\operatorname{argmax}}\, \mathsf{P}(q_{1..T}) \cdot \mathsf{P}(o_{1..T} \mid q_{1..T}) \tag{2}$$

$$\overset{\mathrm{def}}{=} \underset{q_{1..T}}{\operatorname{argmax}} \prod_{t=1}^{T} \mathsf{P}_{\Theta_\mathrm{L}}(q_t \mid q_{t-1}) \cdot \mathsf{P}_{\Theta_\mathrm{O}}(o_t \mid q_t) \tag{3}$$

Language model transitions $\mathsf{P}_{\Theta_\mathrm{L}}(q_t \mid q_{t-1})$ over complex hidden states q_t can be modeled using synchronized levels of stacked-up component HMMs in an HHMM as in Murphy and Paskin (2001). HHMM transition probabilities are calculated in two phases: a 'reduce' phase (resulting in an intermediate, marginalized state f_t), in which component HMMs may terminate; and a 'shift' phase (resulting in a modeled state q_t), in which unterminated HMMs transition, and terminated HMMs are re-initialized from their parent HMMs. Variables over intermediate f_t and modeled q_t states are factored into sequences of depth-specific variables – one for each of D levels in the HMM hierarchy:

$$f_t = \langle f_t^1 \dots f_t^D \rangle \tag{4}$$

$$q_t = \langle q_t^1 \dots q_t^D \rangle \tag{5}$$

Transition probabilities are then calculated as a product of transition probabilities at each level, using level-specific 'reduce' Θ_F and 'shift' Θ_Q models:

$$\mathsf{P}(q_t \mid q_{t-1}) = \sum_{f_t} \mathsf{P}(f_t \mid q_{t-1}) \cdot \mathsf{P}(q_t \mid f_t\; q_{t-1}) \tag{6}$$

$$\overset{\mathrm{def}}{=} \sum_{f_t^1..f_t^D} \prod_{d=1}^{D} \mathsf{P}_{\Theta_\mathrm{F}}(f_t^d \mid f_t^{d+1} q_{t-1}^d q_{t-1}^{d-1}) \cdot \prod_{d=1}^{D} \mathsf{P}_{\Theta_\mathrm{Q}}(q_t^d \mid f_t^{d+1} f_t^d\; q_{t-1}^d q_t^{d-1}) \tag{7}$$

with f_t^{D+1} and q_t^0 defined as constants.

Shift and reduce probabilities are now defined in terms of finitely recursive FSAs with probability distributions over transition, recursive expansion, and final-state status of states at each hierarchy level. In simple HHMMs, each intermediate state variable is a boolean switching variable $f_t^d \in \{\mathbf{0}, \mathbf{1}\}$ and each modeled state variable is a syntactic, lexical, or phonetic state q_t^d. The intermediate variable f_t^d is true (equal to $\mathbf{1}$) with probability 1 if there is a transition at the level immediately below d and the stack element q_{t-1}^d is a final state, and false (equal to $\mathbf{0}$) with

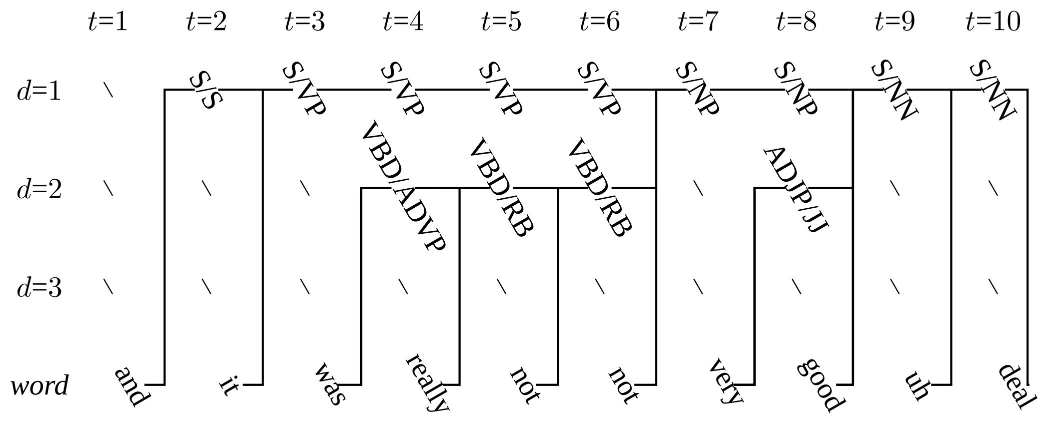


Figure 2: Sample tree from Figure 1 mapped to q_t^d variable positions of an HHMM at each stack depth d (vertical) and time step t (horizontal). Values for final-state variables f_t^d are not shown. Note that some nonterminal labels have been omitted; labels for these nodes can be reconstructed from their children.

probability 1 otherwise:[2]

$$\mathsf{P}_{\Theta_{\text{F}}}(f_t^d \mid f_t^{d+1} q_{t-1}^d q_{t-1}^{d-1}) \overset{\text{def}}{=} \begin{cases} \text{if } f_t^{d+1}{=}\,\mathbf{0} : [f_t^d{=}\,\mathbf{0}] \\ \text{if } f_t^{d+1}{=}\,\mathbf{1} : \mathsf{P}_{\Theta_{\text{F-Reduce}}}(f_t^d \mid q_{t-1}^d, q_{t-1}^{d-1}) \end{cases} \tag{8}$$

where $f^{D+1} = \mathbf{1}$ and $q_t^0 = \mathbf{ROOT}$.

Shift probabilities at each level are defined using level-specific transition $\Theta_{\text{Q-Trans}}$ and expansion $\Theta_{\text{Q-Expand}}$ models:

$$\mathsf{P}_{\Theta_{\text{Q}}}(q_t^d \mid f_t^{d+1} f_t^d\, q_{t-1}^d q_t^{d-1}) \overset{\text{def}}{=} \begin{cases} \text{if } f_t^{d+1}{=}\,\mathbf{0},\ f_t^d{=}\,\mathbf{0} : [q_t^d{=}\,q_{t-1}^d] \\ \text{if } f_t^{d+1}{=}\,\mathbf{1},\ f_t^d{=}\,\mathbf{0} : \mathsf{P}_{\Theta_{\text{Q-Trans}}}(q_t^d \mid q_{t-1}^d q_t^{d-1}) \\ \text{if } f_t^{d+1}{=}\,\mathbf{1},\ f_t^d{=}\,\mathbf{1} : \mathsf{P}_{\Theta_{\text{Q-Expand}}}(q_t^d \mid q_t^{d-1}) \end{cases} \tag{9}$$

where $f^{D+1} = \mathbf{1}$ and $q_t^0 = \mathbf{ROOT}$. This model is conditioned on final-state switching variables at and immediately below the current HHMM level. If there is no final state immediately below the current level (the first case above), it deterministically copies the current HHMM state forward to the next time step. If there is a final state immediately below the current level (the second case above), it transitions the HHMM state at the current level, according to the distribution $\Theta_{\text{Q-Trans}}$. And if the state at the current level is final (the third case above), it re-initializes this state given the state at the level above, according to the distribution $\Theta_{\text{Q-Expand}}$. The overall effect is that higher-level HMMs are allowed to transition only when lower-level HMMs terminate. An HHMM therefore behaves like a probabilistic implementation of a pushdown automaton (or 'shift-reduce' parser) with a finite stack, where the maximum stack depth is equal to the number of levels in the HHMM hierarchy.

2.3 Mapping trees to HHMM derivations

Any tree can now be mapped to an HHMM derivation by aligning the nonterminals with q_t^d categories. First, it is necessary to define rightward depth d, right index position t, and final (right) child status f, for every nonterminal node A in a tree, where:

- d is defined to be the number of right branches between node A and the root,
- t is defined to be the number of words beneath or to the left of node A, and
- f is defined to be **0** if node A is a left (or unary) child, **1** otherwise.

Any binary-branching tree can then be annotated with these values and rewritten to define labels and final-state values for every combination of d and t covered by the tree, using the rewrite rule: This rewrite simply copies stacked up constituents over multiple time steps, while other constituents are being recognized. Coordinates $d, t \leq D, T$ that are

[2]Here $[\cdot]$ is an indicator function: $[\phi] = 1$ if ϕ is true, 0 otherwise.

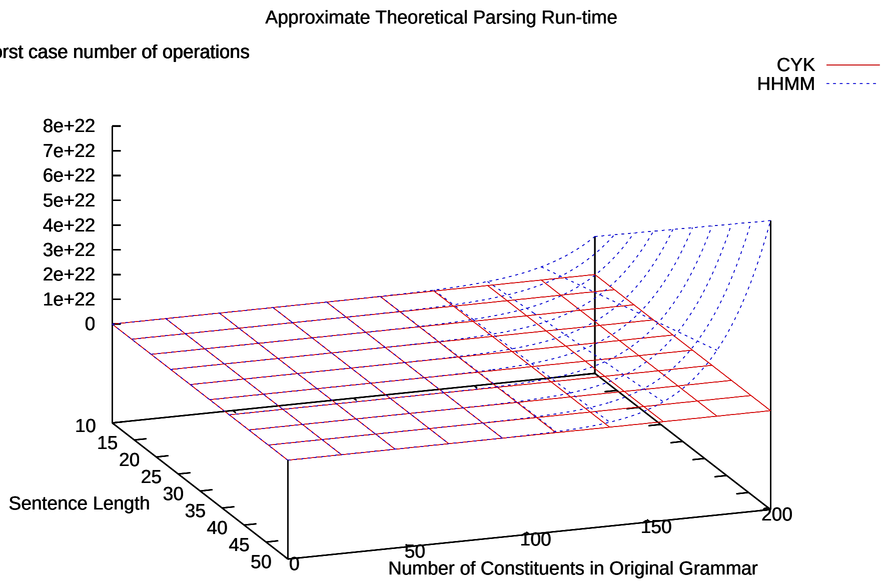


Figure 3: This plot shows the estimated run-time if we approximate the constants in the CYK and HHMM algorithms as described in Section 3.

not covered by the tree are assigned label '$-$', and $f = \mathbf{1}$. The resulting label and final-state values at each node now define a value of q_t^d and f_{t+1}^d for each depth d and time step t of the HHMM (see Figure 2). Probabilities for HHMM models $\Theta_{\text{Q-Expand}}$, $\Theta_{\text{Q-Trans}}$, and $\Theta_{\text{F-Reduce}}$ can then be estimated from these values directly. Like the right-corner transform, this mapping is reversible, so q and f values can be taken from a hypothesized most likely sequence and mapped back to trees (which can then undergo the reverse of the right-corner transform to become ordinary phrase structure trees).

3 Run-time analysis

The CYK algorithm is widely known and frequently used, with run-time complexity of $\mathcal{O}(n^3)$, where n is the number of terminal symbols in the string. This complexity is derived from the fact that the algorithm contains three nested iterations across starting point, end point, and split point while it is parsing. In parsing applications, however, the worst case asymptotic complexity may not provide enough information to distinguish between the speed of two algorithms, since the vast majority of human generated sentences tend to be shorter than 50 words. For our purposes, then, it is useful to point out that a closer approximation to the number of required operations for the CYK algorithm is actually $\mathcal{O}(n^3 Q^3)$, where Q is the number of constituent labels in the grammar. This results from the innermost part of the main loop, where the algorithm contains three more nested loops that iterate over the two possible symbols on the right hand side of the rule and the possible symbol on the left hand side.

HHMM parsers, on the other hand, are not as widely known, but the time complexity properties of HHMMs are well studied. Murphy and Paskin (2001) showed that by casting an HHMM as a dynamic Bayes network, the algorithms for doing inference on that HHMM can be made linear on the length of the input string, which is just the word input sequence. Taking a closer look, they show that a better approximation to the number of operations that an HHMM parser must use is $\mathcal{O}(nDQ'^{\lceil 1.5D \rceil})$, where n is again the length of input, D is the depth of the HHMM, and Q' is the number of possible values of a given random variable.

In the HHMM parser presented here, $D = 3$, since our trees have been transformed as above, such

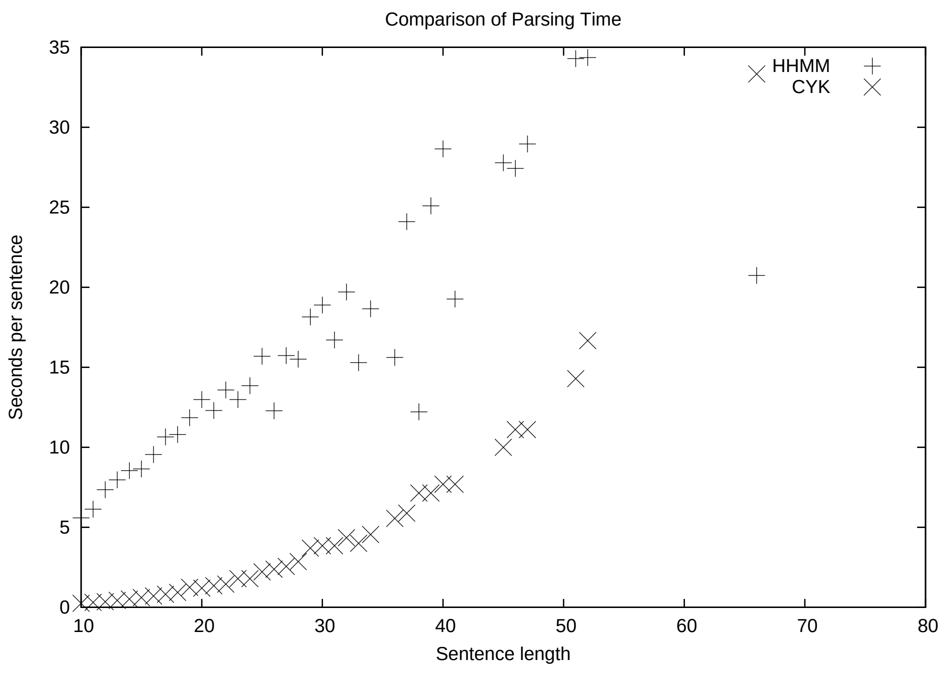


Figure 4: This graph shows the results for test runs of the HHMM parser as well as a standard CYK-style parser. The x-axis shows sentence length, while the y-axis shows number of seconds per sentence. Gaps in the chart indicate that the test data did not contain sentences of that length.

that almost all trees in our corpus fit into that depth. Q' corresponds to the number of constituent types, but in this case we have to account for all the incomplete constituents (slash categories) created by the right corner transform. To a first approximation, this value is Q^2, where Q is the number of constituent types in the original grammar (the same value as for the CYK parser's run-time approximation), since each constituent can be of the form α/β, for α and β each being a constituent in the original grammar. [3]

We plotted a comparison of a CYK parser's expected runtime with a HHMM parser's expected runtime, estimated as above, in Figure 3. The two independent variables shown are sentence length and number of rules in the original grammar, while the dependent variable (z-axis) shows the approximate worst case number of operations for the parser. The notable thing in this graph is that for any reasonable sentence size (and even well beyond), the relevant variables in determining run-times for the respective parsers are in the range where the constant for grammar size is dominant over the term for sentence length.

[3]In fact, this is a conservative estimate because the binarization preprocess to the right corner transform creates additional categories.

4 Evaluation

The CYK and HHMM parsers both have the same input and output, but they work in entirely different ways internally. The HHMM parser contains a much richer syntactic representation (due to the binarization and right corner transform), operates in a top-down manner, and has different dependencies than a CYK parser. If run unconstrained, this architecture can potentially be very accurate, but very slow. We wanted to see what sort of time performance this system could achieve if we use a beam search to constrain the set of hypotheses explored by the HHMM during processing, with the beam width set such that the accuracy results are comparable to the CYK parser.

To evaluate the actual run-time properties of the HHMM parser we performed experiments using sentences of varying length, and compared total runtime between this parser and an off the shelf PCFG parser using the CYK algorithm. The sentences for both training and testing were taken from the Switchboard corpus (Godfrey et al., 1992), a corpus of spontaneous conversations between pairs of individuals segmented by annotators into sentence-like units. In addition, to simulate realistic speech parsing conditions, input to the parsers was stripped of punctuation for both training and testing.

For testing, the data was arranged by length, starting with utterances of 10 words, extending to utterances of 66 words. Both the HHMM parser and CYK parser were run on all sentences of each length, and average parsing time was calculated across each sentence length. In both parsers, the timer was started after the grammar was loaded into memory, so the times measured do not include the overhead, only parsing time.

Figure 4 shows the results of these experiments. This plot shows that in the range of sentence lengths encountered in this corpus, the CYK parser is indeed faster than the HHMM parser. Despite what might seem to be prohibitively large constants in the theoretical run-time of the HHMM parser, through most of this range of input length the HHMM parser is only about twice as slow as the CYK parser.

5 Discussion

These results suggest that a CYK parser may be preferred in standard applications parsing segmented text when speed is an issue, while an HHMM parser will be most useful when its unique properties are able to be exploited. For example, in streaming text input, say from a speech recognizer, sentence boundaries are not given, and the input will need to be segmented. Finding sentence-like units (SUs) in speech is a difficult problem, as shown by error rates of about 26% in state of the art systems (Roark et al., 2006). That system then uses a discriminative reranker to choose from among the n-best outputs of the SU detector. As a result, even though the CYK parser may run fast on the resulting SUs, obtaining that segmentation as a preprocess takes some amount of time. Further, it may be necessary to parse several of the best segmentations and choose the best of those to obtain the best results.

On the the other hand, an HHMM parser can be run on streaming input, and do segmenting as it parses. This has the potential advantage of considering multiple sources of knowledge and preserving uncertainty among different tasks involved, parsing and boundary detection to find the best overall analysis. Future work for this parser will take the Switchboard corpus as input, ignoring sentence breaks, and parse the entire stream as one unit. We expect that this approach will not only do well at doing parsing and segmenting, but it will be very efficient as well, because it does segmentation automatically, as part of the parsing process.

References

John J. Godfrey, Edward C. Holliman, and Jane McDaniel. 1992. Switchboard: Telephone speech corpus for research and development. In *Proc. ICASSP*, pages 517–520.

Mark Johnson. 1998. Finite state approximation of constraint-based grammars using left-corner grammar transforms. In *Proceedings of COLING/ACL*, pages 619–623.

Tim Miller and William Schuler. 2008. A unified syntactic model for parsing fluent and disfluent speech. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL '08)*.

Kevin P. Murphy and Mark A. Paskin. 2001. Linear time inference in hierarchical HMMs. In *Proc. NIPS*, pages 833–840.

Brian Roark, Yang Liu, Mary Harper, Robin Stewart, Matthew Lease, Matthew Snover, Izhak Safran, Bonnie Dorr, John Hale, Anna Krasnyanskaya, and Lisa Yung. 2006. Reranking for sentence boundary detection in conversational speech. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP'06)*.