Examining Working Memory during Sentence Construction with an ACT-R Model of Grammatical Encoding

Jeremy R. Cole
The Pennsylvania State University
University Park, PA
jrcole@psu.edu

David Reitter
The Pennsylvania State University
University Park, PA
reitter@psu.edu

Abstract

We examine working memory use and incrementality using a cognitive model of grammatical encoding. Our model combines an empirically validated framework, ACT-R, with a linguistic theory, Combinatory Categorial Grammar, to target that phase of language production. By building the model with the Switchboard corpus, it can attempt to realize a larger set of sentences. With this methodology, different strategies may be compared according to the similarity of the model's sentences to the test sentences. In this way, the model can still be evaluated by its fit to human data, without overfitting to individual experiments. The results show that while having more working memory available improves performance, using less working memory during realization is correlated with a closer fit, even after controlling for sentence complexity. Further, sentences realized with a more incremental strategy are also more similar to the corpus sentences as measured by edit distance. As high incrementality is correlated with low working memory usage, this study offers a possible mechanism by which incrementality can be explained.

Introduction

Working memory has long been thought to play an important role in language processing (e.g., Gibson, 1998). In language production, one important question concerns *grammatical encoding*, the process by which words are combined into sentences. In this paper, we make progress in understanding the interaction of working memory with strategies and representations that are needed for grammatical encoding.

One strategic decision that is crucial to grammatical encoding is how incremental the process is: are words planned in the exact order they are output, or are other mechanisms at play? There are, obviously, opportunities to leverage representational insights from computational linguistics and algorithmic choices known in natural-language generation. Models of grammatical encoding thus far have been either too general or too specific to both make clear predictions that are testable with behavioral methods or against known effects.

Computational implementations of linguistic models (e.g., Steedman, 2000) are geared towards performance, not explanatory value, which makes it difficult to evaluate their demand for limited cognitive resources and to determine their interactions with general cognition. On the other hand, connectionist models often aim to be more agnostic to the linguistic task (e.g., Dell et al., 1999), making them challenging to interpret once they are trained on data. In short, the general motivations to engage in a combination of cognitive modeling and computational linguistics apply to grammatical encoding.

With the availability of large-scale data, language models should strive to explain as much data as possible. Reusing syntactic alternatives, such as the difference between double object and prepositional object, can only take the field so far in explaining the richness of human discourse. Conversely, if our models are high coverage but not cognitively plausible, the performance of the model may be good from an engineering perspective, but the model cannot be said to have explained any of the data from a scientific perspective.

Big data, used appropriately, also allows us to contrast different model variants in terms of their explanatory power. This can lead to incremental improvements. In short, we call for a new type of cognitive modeling where it is possible: instead of modeling a relatively small number of experiments surrounding a phenomenon, we model a large amount of the raw data produced by the phenomenon itself. For our task, we evaluate against a corpus.

In this paper, we advance towards such a computational cognitive model of grammatical encoding. The implemented model that we will discuss has clear, interpretable representations in the form of Combinatory Categorial Grammar (CCG, Steedman and Baldridge, 2011). It is cognitively plausible and implemented in an empirically validated framework, ACT-R (Anderson et al., 2004). It is empirically testable, as the model can produce output for any target sentence, allowing competition among alternative models. Our model in particular examines how incrementality in syntax, working memory availability, and working memory usage can improve or worsen the model's fit to linguistic data.

Related Work

This paper builds on a rich body of work from both psychology and linguistics attempting to characterize the language production process.

Grammatical Encoding

There are many ways to discretize the steps of the full process of language production. For instance, we could say after an idea is formulated, it is grammatically encoded and then phonologically encoded (Bock and Levelt, 2002). In turn, grammatical encoding could consist of lexical selection, function assignment, and constituent assembly. The first stage maps ideas to words; the second stage maps words to parts of speech, and the last stage combines these lexical-syntactic units (hereafter lexsyns) into constituents. As syntactic trees are formed by recursively combining constituents, this process eventually leads to a sentence. Thus, the full process of grammatical encoding transforms ideas, or semantics, into a realized sentence.

Working Memory and Language Production

The precise effect of memory on syntactic processing has not been the focus of previous studies. Nonetheless, some constraints have been proposed on sentence formulation, including showing that a higher span can decrease certain types of grammatical errors (Hartsuiker and Barkhuysen, 2006; Badecker and Kuminiak, 2007). Further, Slevc (2011) suggests working memory load can affect the incrementality of a sentence. However, the discussion of representations in working memory for the function assignment and constituent assembly process is a matter of linguistic theory. We turn to Combinatory Categorial Grammar (Steedman and Baldridge, 2011), which provides a possible representation, and we show how it could map to the psychological architecture of attention, processing and memory.

Theories of Incrementality

V. S. Ferreira (1996) makes an argument for incrementality, based on the observation that competitive syntactic alternatives facilitate production rather than making it more difficult. An incremental account of sentence realization would predict such an effect, as syntactic "flexibility" introduced by the alternatives makes it easier to find a workable syntactic decision. By contrast, without incremental commitment to each structure, competing material slows down the process, because it would lead to a combinatory explosion. Further results, however, relativize this account when it comes to the syntax-phonology interface (F. Ferreira and Swets, 2002). Incremental production is possible, but it is "under strategic control"; it depends on semantic information, and it could be modulated by external factors, such as stress.

Based on the literature, we assume that incrementality in grammatical encoding may be graded: the degree to which a sentence is realized incrementally may vary based on certain cognitive factors. However, the literature has yet to address how speakers (or comprehenders) might use the limited memory resources available to guide the attendant strategic choices surrounding incremental realization. Our corpus-driven model has the potential to explain this by contrasting models with different available working memory, and by examining actual working memory use, as well as by measuring the activation and availability of linguistic structures.

Background

Our model relies on the unification between a linguistic theory (CCG) and a cognitive framework (ACT-R), which will be explained in turn in the following section.

Linguistic Theory

Combinatory Categorial Grammar (CCG) is a grammar formalism (Steedman and Baldridge, 2011). While it was not conceived as a purely psycholinguistic theory, interpreting it as such has a few important consequences. Most importantly, after a syntactic operation, the representations are simplified. This is as opposed to other grammar formalisms, where combination always results in a more complicated representation, e.g. Tree-Adjoining Grammar (Joshi and Schabes, 1997). In general, grammar formalisms operate based on *types*, such as noun phrase, and *rules*, which are methods for combining the types into a sentence. See Table 1 for a demonstration of CCG's combinatory rules. See Figure 1 for an example of how these rules can create sentences.

Table 1: The four basic rules of CCG, which specify how syntactic types can be combined. They are the rules by which types can be systematically combined into a sentence. The left-hand side specifies two types, each of which are recursively composed of one or two types (e.g. X/Y is one type). The right-hand side specifies the resulting type of the operation on the left.

Forward Application (>):	$X/Y > Y = X$
Backward Application (<):	$Y < X \setminus Y = X$
Forward Composition (>>):	$X/Y >> Y/Z = X/Z$
Backward Composition (<<):	$Y \setminus Z << X \setminus Y = X \setminus Z$

ACT-R

ACT-R is a general theory of cognition (Anderson et al., 2004). ACT-R, combined with a linguistic theory like CCG, can provide a unification of computational modeling, cognitive science, and linguistics. ACT-R's basic system for writing models involves chunks and production rules, where chunks represent declarative memory and production rules represent procedural memory. In the following sections, we discuss how we infer the chunks from a corpus.

Methods

We create a model that is automatically derived from the syntactic and lexical information present in $1,200$ sentences sampled from the *Switchboard* corpus (Godfrey et al., 1992). Switchboard is a spoken language corpus of two strangers having a phone conservation about a provided topic. We then run this model with no interruptions or constraints, using the unordered bag-of-words from the corpus sentences as input as an approximation of the meanings (one sentence at a time), and expecting sentences or sentence fragments as

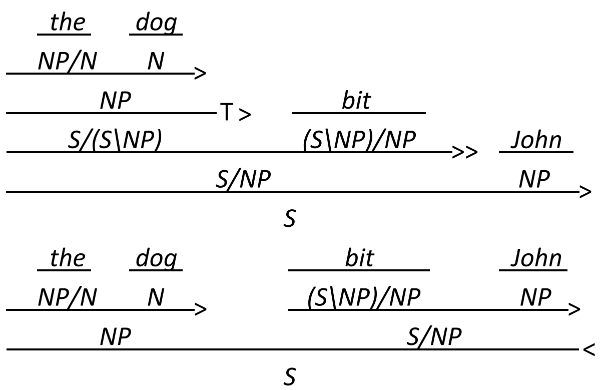

Figure 1: Two contrasting CCG derivations: The top is more incremental (right-branching) than the bottom. Note that the $T>$ is normally used to mark non-standard derivations, which are usually more incremental.

output. The model's process is recorded in the form of syntax trees ("derivations") for further analysis, as these derivations reflect the strategy applied by the model to produce the sentences. The model's performance under different working memory conditions will be evaluated by comparing to each original sentence. Thus, the core task of the model is to recover the original ordering of the words in each sentence.[1]

Model

Our model is implemented in *jACT-R* (Harrison, 2005), a full Java implementation of the ACT-R theory (Anderson et al., 2004). This was primarily due to convenience, portability, and scalability, rather than any difference in theoretical predictions between jACT-R and the core Lisp ACT-R.

The model's combinatory mechanism is based on CCG. As CCG specifies clear symbolic and procedural components, it maps naturally to chunks and production rules. The exact mapping will be discussed in the following sections. As discussed earlier, using CCG as the combinatory mechanism of the cognitive model means that combinations will reduce the current memory use. We acknowledge that such predictions should match data on working memory, though we don't see such a prediction as out of line with current ideas about chunking (Conway et al., 2005).

The model is generated from a corpus. The model's goal is to encode all of the sentences found in the corpus into declarative memory and production rules. A wide range of models can be created in this way; however, our empirical evaluation is based on a model learned from a subset of the Switchboard corpus. The chosen sentences use more frequent syntactic types and are of shorter length. Then, the model learns the words and potential syntactic types from the raw text and CCG annotations. These learned syntactic types serve as possible function assignments for the words. Importantly, this is all the model learns: the production rules are encoded with no knowledge of the sentences.

In short, the current model forms a sentence by combining lexical-syntactic chunks together. Out of simplicity, it chooses what to combine greedily. Importantly, it treats no words, types, or rules as special, and it has no knowledge of what words or types should go together beyond the constraints of CCG. Nonetheless, simply following CCG rules can lead to unidiomatic sentences and potentially even ungrammatical sentences by violating certain thematic constraints. This is true of the presented model. However, the full expressive range of syntactic and lexical constructions found in a corpus requires substantial learning, which is out of scope for the present paper. Thus, while many constructions of our model are unidiomatic, we provide a baseline for future work to be evaluated against. Randomly selected example constructions by the model can be found in Table 2.

[1]Due to our lack of test set, the careful reader may note that it would be possible to overfit. However, our model does not learn word orderings directly from the corpus, instead only learning syntactic types: indeed, we are much more interested in the effects of working memory and grammatical encoding strategy.

Declarative Memory

Declarative Memory (DM) is composed of a few simple chunk types, described below. The basic organizational scheme has Sentences composed of Lexsyns, and Lexsyns composed of a single Word and several Types that the word can be in a given context.**Words** simply have a name, which corresponds to its lexical information (e.g. family).

Types are an arbitrarily complicated CCG type. The types that exist in DM are the types that are used in Switchboard CCG derivations of our chosen sentences.

Lexsyns associate a Word with some number of Types. These associated types are taken from the function assignments of each word in the Switchboard CCG derivations of our chosen sentences. The types are ordered from most common to least common, which would mean more common types would be selected if all else is equal.

Sentences are normally in the goal buffer. Thus, the sentence contains the current state of grammatical encoding. If we think of the goal buffer as working memory, then differing the slots available to realize a sentence corresponds to different predictions about working memory availability. Additionally, the sentence chunk also contains the input for the task. However, this is more of a limitation than a theoretical commitment: due to our focus on grammatical encoding, we had to assume the previous tasks of idea generation and lexical selection were complete. In reality, it is likely that all three tasks overlap to some extent.

Table 2: Example sentences produced by the model. The 'target' is the actual sentence from the corpus, while the realization is what the model produced. The quality of the model's output varies.

Realization	Target
downhill going like everybody	but then they started going downhill like everybody else
you fire never something unless anybody 're caught	they never fire anybody unless you 're caught doing something illegally
still taxes raise probably and	i think he can probably raise taxes and still get elected
i then and decided i like author this	and then i decided i like this author
are school working you	are you working anywhere while you are going to school

Production Rules

We define a small set of about ten production rules (which are compiled into several thousand production rules through an automatic process, which we will not describe in detail here). Depending on the production, the architecture will choose an appropriate rule; there is no predefined algorithmic flow. The model's production rules fall into three basic categories.

1. **Syntax Rule Application** This production rule may fire if Working Memory contains at least two Lexsyns whose types would follow the constraints of at least one CCG rule.

If so, it initiates Rule (3) to determine the result of the rule application.

2. **Goal buffer modification** (A) Move word from Input to Working Memory: This production rule can only fire if there is space in the goal buffer for it to be added. It simply deletes the word from the input, and initiates Rule (3) to retrieve its function assignment. (B) Flush: If no other rules apply, the model will flush, clearing its retrieval buffer or working memory to try again. (C) Resolve Syntax Rule: This deletes the unnecessary entries in working memory after the resulting type is known, as the two entries are combined into the single CCG entry representing the result of applying the rule.

3. **Lexical Retrieval from DM** Retrieve the possible function assignments of a word (its lexsyn), or retrieve the type resulting from the application of a syntax rule.

Input

Due to our focus on grammatical encoding, the input to the model is a bag of words generated from the target corpus sentence. To be clear, the model does not order the words its given as input, instead it only combines two words if its possible under CCG using their current function assignments. Thus, not every output sentence uses every word in the input.

Experiment

In our experiment, we use the model to ask how the size of verbal working memory relates to the fidelity of the produced sentences, and how this interacts with the strategy for grammatical encoding.

Conditions

Working Memory (WM) We contrast two basic versions of the model with 3 and 5 working memory slots, respectively. We consider these values as realistic lower and upper bounds of working memory capacity as found in language tasks (Daneman and Carpenter, 1980). This is implemented simply by limiting the number of slots in the Sentence chunk, so the model has less available working memory to use to combine Lexsyns. We distinguish working memory span (controlled) from actual working memory usage (observed).

Dependent Variables

Branching Factor We see grammatical encoding as a process that is quite flexible: the set of production rules, and the absence of a fixed algorithm (and order in which they are applied) is commensurate with that (as well as with ACT-R as a cognitive architecture framing the model). Strategies emerge as a result of the available cognitive resources, such as WM, and, ultimately (not modeled) the success of rule sets. We measure an important aspect of the strategy: incrementality, as determined by branching factor: The more right-branching a syntax tree is, the more incrementally it was realized.

We define two basic metrics for measuring branching factor. The unweighted branching factor (UBF) is the number of right-branching decisions compared to the number of total decisions. The weighted branching factor (WBF) takes into account how far up the syntax tree the decision was made; it short, it sums all of the subtrees rather than simply comparing the decisions. An example tree and computation can be found in Figure 2, which is an syntax tree created from the model's syntactic decisions. Alternatively, to reference the CCG derivations from earlier in Figure 1, the top derivation has a WBF of 3 and a UBF of 7, while the bottom derivation has a WBF of 1.0 and a UBF of 1.0. These values are not on the same scale: 1.0 is the mean for WBF, but 0.2 is the mean of UBF. Both metrics correlate with each other and higher values represent more incremental constructions.

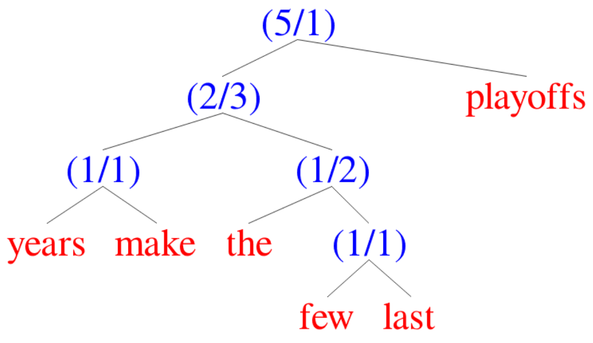

Figure 2: An example of the syntax tree of actual output from the model. To compute the weighted branching factor (WBF), sum the numbers in parentheses for the left and the right, then divide the sum of the left numbers by the sum of the right numbers. The numbers indicate the number of leaves in the right and left subtrees. This computes to 10/8, or 1.25. The unweighted branching factor (UBF) divides the total number of left leaf nodes (in this tree, 3) by right leaf nodes (in this tree, 3), getting a branching factor of 1.0.

Working Memory Usage (WMU) This is based on the maximum amount of slots the model used while realizing a sentence. This includes slots for retrieval and all lexsyns stored in the working memory portion of the goal buffer. We additionally compute the adjusted working memory usage (AWMU), which takes into account the length of the sentence, as longer sentences could possibly require additional working memory, especially if constructions tend to be less incremental.

Edit Distance This measure evaluates fidelity of the model output, i.e., match between the result and the input sentence is computed using Levenshtein distance (Levenshtein, 1966). An edit distance in general is the number of changes (additions, swaps, and deletions) to transform one list into another one: in this case, a sentence is treated as a list of words. Thus, it is a measurement for how dissimilar two sentences are from each other. If the model produces multiple fragments rather than a single utterance, the distances are averaged. We chose edit distance as a metric to ensure the model's trace of syntax was being measured, rather than simply the meaning. It correlates well with metrics used to evaluate natural language processing tasks (e.g., Lin, 2004). We define the edit distance between the two sentences as the model fit.

Results

We examined the correlations between the branching factor, working memory usage, and fit, as measured by edit distance between the realization and the target sentences. We analyze the influence of observed branching factor, available WM and observed WM usage separately.

Both branching factor metrics (UBF and WBF) were found to be significant with a negative effect on edit distance, implying more incremental constructions produce realizations more similar to the initial sentences ($p < 0.001$). Conversely, working memory use (WMU) was found to have a positive effect, implying increased working memory usage decreased fit ($p < 0.001$). This had an even larger effect for adjusted working memory use (AWMU), implying minimizing working memory usage was especially important for longer sentences. Branching Factor and Working Memory usage (all metrics) were also significantly correlated ($p < 0.001$).

Table 3: Individual linear models correlating predictors with edit distance in the WM=3 condition.

	WBF	UBF	WMU	AWMU
p-value	< 0.0001	< 0.0001	0.007	< 0.0001
effect	-0.221	-0.168	0.057	0.074
Intercept	0.873	0.763	0.507	0.610
r^2	0.056	0.025	0.162	0.065
df	1038	1038	1038	1038

Because increased working memory usage is correlated with decreased fit, this begs the question of whether that is because sentences with lower working memory usage requirements are easier, or whether using more working memory directly decreases fit. As sentences have different working memory requirements, the ones with lower requirements could just be easier to realize incrementally, possibly reducing production errors. The five-slot model helps elucidate this.

In the five-slot condition, the model performs slightly better, even though it uses more working memory on average by both metrics. However, it is also more right-branching than the other model by both metrics.

Table 4: Individual linear models correlating predictors with edit distance in the WM=5 condition.

	WBF	UBF	WMU	AWMU
p-value	< 0.0001	< 0.0001	< 0.0001	< 0.0001
effect	-0.162	-0.228	0.120	0.041
Intercept	0.915	0.773	0.427	0.685
r^2	0.079	0.536	0.092	0.015
df	1038	1038	1038	1038

Discussion

The most interesting take-away from this is that higher working memory usage, which was previously associated with fewer speech errors, is associated with worse fit to the corpus data in our model. This could be because increased working memory usage, rather than alleviating stress caused by low-resources, causes the realizer to garden-path itself. By allowing itself to work breadth-first, it can potentially make syntactic choices that won't eventually lead to a good utterance. The branching factor could partially be a result of this: having a higher right-branching factor should lead to lower working memory use, as new elements are added to the current state, rather than built up in another way. However, it could also be a simple consequence of the fact that since language is outputted in order, it's easier to combine it in order, thus allowing earlier outputs. Importantly, this result indicates the effect of *using* less working memory when all else is equal. It does not indicate the effect of *having* less working memory available. An important caveat then, is that this effect could simply be explained with the observation that easier sentences use less working memory.

Having working memory available when needed clearly improves fit, even though in general, using more working memory worsens fit. Varying WM capacity does not change the general strategy of grammatical encoding, which prefers to use less working memory and more right-branching constructions. Still, the model with less working memory was less right-branching. This could perhaps be because without additional working memory available, it sometimes had to settle for an inferior strategy, perhaps explaining its fit decline, in line with work such as Slevc (2011). We consider the lower fit of the lower working memory model to be in line with previous research, which leaves open as a possible avenue for future experimentation the correlation of lower working memory usage to higher fit.

We consider both of these results to be compatible with the hypothesis of strategic incrementality. More incremental processes require less working memory. This is because lexsyns can be combined and outputted, freeing space. Moreover, reducing working memory usage is normally used as a possible argument for why incremental strategies might be preferred. That still leaves two basic possibilities: (1) Speakers prefer to use constructions that are possible to realize more incrementally, or (2) speakers attempt to realize all constructions as incrementally as possible. We have reason to believe, from F. Ferreira and Swets (2002), that (2) is not the case, unless the speakers are under some stress to speak as quickly as possible. Possibly, (1) can be fairly easily examined from frequency rates, though we are unaware of work doing so.

Table 5: Paired t-tests between WM=3 and WM=5 conditions.

	WBF	UBF	WMU	AWMU	dist
Cond1-Mean	1.050	0.132	3.156	1.721	0.743
Cond2-Mean	1.023	0.105	2.703	1.575	0.748
p-value	< 0.0001	< 0.0001	< 0.0001	< 0.0001	< 0.0001

By limiting working memory directly, we are able to demonstrate through the model that working memory is critical to grammatical encoding: limiting it or having less of it increases errors experimentally and reduces the fit of our model. While our task was naturally not perfectly analogous to experimental work, it does provide converging evidence in this discussion (Hartsuiker and Barkhuysen, 2006; Badecker and Kuminiak, 2007).

However, actually using less working memory on any sentence is correlated with increased model fit, even after controlling for effects of sentence length or complexity. There are several possible explanations for this. For instance, pushing working memory to capacity could be more likely to cause errors, as speakers retrieve too many lexsyns that can't be combined, forcing themselves to flush, thereby losing track of part of the sentence. Conversely, each sentence may dictate an minimum amount of working memory needed to realize a sentence even in an incremental fashion. In that case, the model predicts, testably, sentences with a lower minimum work memory requirement will have fewer production errors than others.

Based on our modeling simulations, we argue that there is a specific amount of modality-specific working memory available to speakers for grammatical encoding, and that speakers generally do not maximize working memory use. Importantly, our conclusions require researchers to take our model for granted, though we do provide metrics by which future models can be compared.

Conclusion

In this paper, we created a model of grammatical encoding (specifically function assignment and constituent assembly) by combining linguistic theory and computational cognitive modeling. We examined working memory's role during this stage of language production, along with additional data on incrementality, finding the model's fit increases with higher incrementality and lower working memory usage, but that having additional working memory available improves overall fit. Lastly, we present the first cognitive model of language production that is evaluated on a corpus, with a paradigm of inquiry that makes progress in modeling by comparing generative fits across different model versions.

Acknowledgements

This project was supported by the National Science Foundation (BCS-1457992). We thank Frank E. Ritter and Matthew Kelly for their comments on an earlier version.

References

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Quin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036–1060.

Badecker, W. & Kuminiak, F. (2007). Morphology, agreement and working memory retrieval in sentence production: evidence from gender and case in Slovak. *Journal of Memory and Language*, *56*(1), 65–85.

Bock, J. K. & Levelt, W. J. M. (2002). Language production. *Psycholinguistics: Critical concepts in psychology*, *5*, 405–452.

Conway, A. R., Kane, M. J., Bunting, M. F., Hambrick, D. Z., Wilhelm, O., & Engle, R. W. (2005). Working Memory Span Tasks: A Methodological Review and User's Guide. *Psychonomic Bulletin & Review*, *12*(5), 769–786.

Daneman, M. & Carpenter, P. A. (1980). Individual differences in working memory and reading. *Journal of Verbal Learning and Verbal Behavior*, *19*(4), 450–466.

Dell, G. S., Chang, F., & Griffin, Z. M. (1999). Connectionist models of language production: lexical access and grammatical encoding. *Cognitive Science*, *23*(4), 517–542.

Ferreira, F. & Swets, B. (2002). How incremental is language production? Evidence from the production of utterances requiring the computation of arithmetic sums. *Journal of Memory and Language*, *46*(1), 57–84.

Gibson, E. (1998). Linguistic complexity: locality of syntactic dependencies. *Cognition*, *68*(1), 1–76.

Godfrey, J. J., Holliman, E. C., & McDaniel, J. (1992). Switchboard: telephone speech corpus for research and development. In *Ieee international conference on acoustics, speech, and signal processing (ICASSP-92)* (Vol. 1, pp. 517–520). IEEE.

Harrison, A. (2005). jACT-R. http://jact-r.org/.

Hartsuiker, R. J. & Barkhuysen, P. N. (2006). Language production and working memory: the case of subject-verb agreement. *Language and Cognitive Processes*, *21*(1-3), 181–204.

Joshi, A. K. & Schabes, Y. (1997). Tree-adjoining grammars. In *Handbook of formal languages* (pp. 69–123). Springer.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady* (Vol. 10, p. 707).

Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop* (pp. 74–81).

Slevc, L. R. (2011). Saying what's on your mind: working memory effects on sentence production. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *37*(6), 1503.

Steedman, M. (2000). Information structure and the syntax-phonology interface. *Linguistic inquiry*, *31*(4), 649–689.

Steedman, M. & Baldridge, J. (2011). Combinatory categorial grammar. *Non-Transformational Syntax: Formal and Explicit Models of Grammar. Wiley-Blackwell.*

Ferreira, V. S. (1996). Is it better to give than to donate? Syntactic flexibility in language production. *Journal of Memory and Language*, *35*, 724–755.