Corpus-Based Lexical Choice in Natural Language Generation

Srinivas Bangalore and **Owen Rambow**
AT&T Labs – Research
180 Park Avenue
Florham Park, NJ 07932
{srini,rambow}@research.att.com

Abstract

Choosing the best lexeme to realize a meaning in natural language generation is a hard task. We investigate different tree-based stochastic models for lexical choice. Because of the difficulty of obtaining a sense-tagged corpus, we generalize the notion of synonymy. We show that a tree-based model can achieve a word-bag based accuracy of 90%, representing an improvement over the baseline.

1 Introduction: Lexical Choice

Sentence planning in natural language generation (NLG) can be characterized as the collection of tasks related to the choice of (abstract) linguistic resources in order to achieve elementary communicative goals. These tasks include lexical and syntactic choice. Lexical choice is the choice of meaning-bearing lexemes. (Function words express grammatical meaning such as tense and aspect features, or are entirely grammatically determined, and therefore are usually chosen not during sentence planning, but during linguistic realization.) Syntactic choice is the choice of how meaning-bearing lexemes are combined. It is clear that lexical and syntactic choice are not independent. For example, if we choose the verb *give*, we can choose between a nominal or prepositional realization of the goal argument (*John gave the organization a car* or *John gave a car to the organization*), while if we choose *donate*, only the prepositional realization is possible.

In many systems, lexical choice is handled in a dictionary mapping from domain concepts (or, in machine translation (MT), source language words to target language words). These dictionaries are typically hand-crafted.[1] While such an approach is perfectly reasonable for many applications, especially those in which linguistic variation in the target texts is restricted, there are applications (including many MT applications) in which the range of linguistic variation of the texts to be generated is far too vast to allow for hand-coding of dictionaries. Furthermore, even for domains with a limited range of linguistic variation, the task of hand-coding the necessary resources can be formidable when a generation system must be ported to a new domain, a new genre, or especially a new language.

In seminal work, Langkilde and Knight (1998) have used a language model as a way of choosing among different lexical and syntactic options. In (Bangalore and Rambow, 2000), we present a model of syntactic choice which, like Langkilde and Knight (1998) relies on a linear language model, but unlike their approach, also uses a tree-based representation of syntactic structure, a tree model, and an independently hand-crafted grammar. We show that the addition of the tree-based model improves the performance of the syntactic choice module. The system, called FERGUS (Flexible Rationalist-Empiricist Generation Using Syntax), does not, how-

[1] There have been some advances in automatic transfer dictionary building for MT, but the automatically extracted dictionaries do not allow for very fine-grained control in lexical choice.

ever, perform any lexical choice: all lexemes in the string to be generated must be represented in the input tree. In this paper we discuss ways of extending FERGUS to handle lexical choice.

The paper is structured as follows. In Section 2, we give a very brief overview over FERGUS as previously published. In Section 3, we present the lexical choice problem and how we propose to model it, and in Section 4 we discuss three ways of integrating lexical choice with the existing syntactic choice. We present our results in Section 5, and conclude with a discussion and an outlook.

2 System Overview

In order to model syntax, we use an existing wide-coverage grammar of English, the XTAG grammar developed at the University of Pennsylvania (XTAG-Group, 1999). XTAG is a tree-adjoining grammar (TAG) (Joshi, 1987). In a TAG, the elementary structures are phrase-structure trees which are composed using two operations, substitution (which appends one tree at the frontier of another) and adjunction (which inserts one tree into the middle of another). In linguistic uses of TAG, we associate one lexical item (its *anchor*) with each tree, and one or (typically) more trees with each lexical item. Since each lexical item is associated with a whole tree (rather than just a phrase-structure rule, for example), we can specify within the structure associated with the lexeme various of the syntactic properties of the lexeme, such as its predicate-argument structure (by including nodes at which its arguments must substitute), the syntactic realization of its arguments (say, using prepositions, or preposed), its passive valency (i.e., how the tree connects to its governor), and morpho-syntactic constraints such as subject-verb agreement.

We can refer to a tree by a combination of its name, called its *supertag*, and its anchor. A supertag encodes all syntactic properties of an instance of a lexeme. For example, α_2 is the supertag of a tree anchored by a transitive verb that projects to S, while γ_3 is the supertag of an adjunct tree anchored by a preposition which takes a nominal argument and adjoins to N.

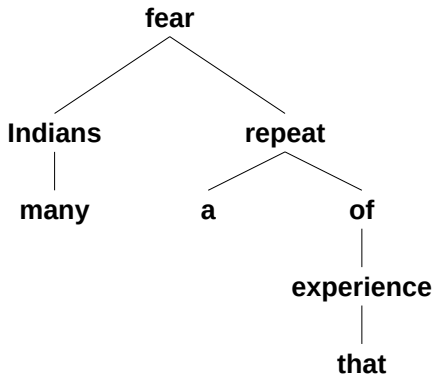

Figure 1: Input to the Syntax Chooser (and the Lexeme Chooser) for sentence *many Indians feared a repeat of that experience*

The original version of FERGUS (without lexical choice) is composed of three modules: the Syntax Chooser, the Unraveler, and the Linear Precedence (LP) Chooser. The diagram in Figure 2 shows this architecture, augmented on the left by the Lexeme Chooser which we will discuss later. The input to the system is a dependency tree as shown in Figure 1. The nodes are labeled only with lexemes, not with supertags.[2] The Tree Chooser then uses a stochastic tree model to choose TAG trees for the nodes in the input structure. This step can be seen as analogous to "supertagging" (Bangalore and Joshi, 1999), except that now supertags (i.e., names of trees) must be found for words in a tree rather than for words in a linear sequence. The tree model is a representation of the XTAG derivation for 1,000,000 words of the Wall Street Journal.

For our example sentence, the input to the system is the tree shown in Figure 1, and the output from the Tree Chooser is the tree as shown in Figure 3. This tree, while more specified than the input tree, still does not determine the surface structure: adjuncts

[2] In the system that we used in the experiments described in this paper, all words (including function words) need to be present in the input representation, fully inflected. This is of course unrealistic for applications.

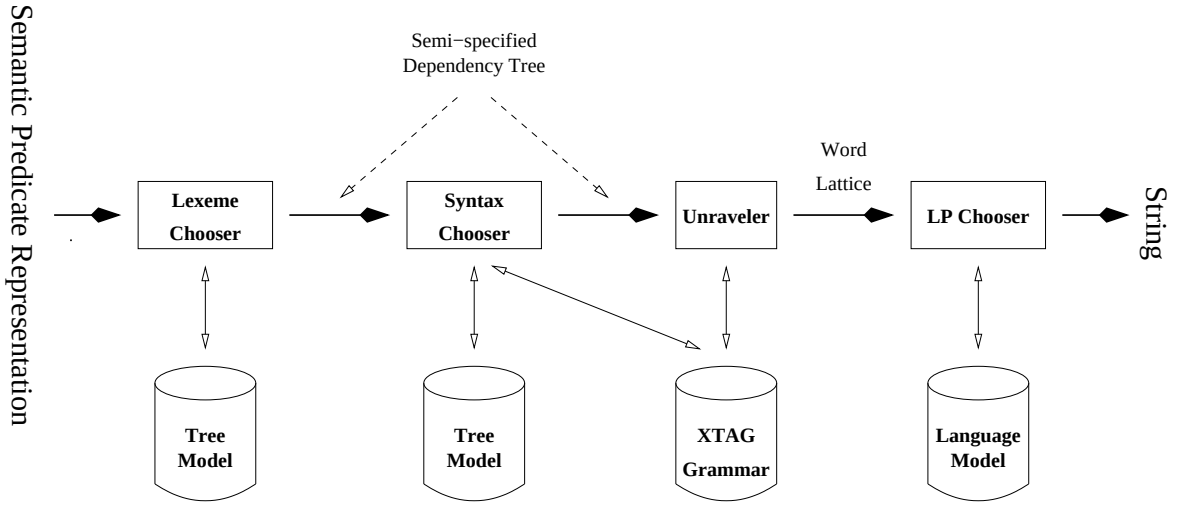

Figure 2: Architecture of FERGUS

are underspecified with respect to the adjunction site and/or the adjunction direction (from left or from right). Furthermore, since the grammatical role information for complements need not be specified in the input, complements are unordered with respect to other complements.

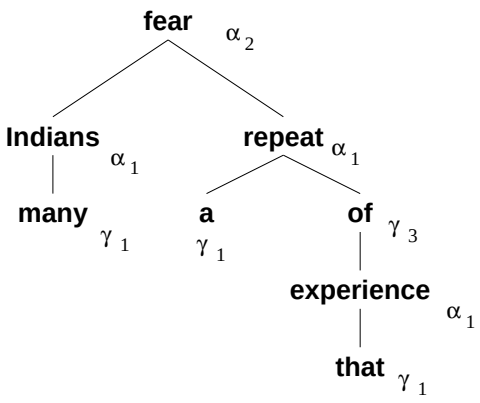

Figure 3: Output of the Syntax Chooser

The Unraveler then uses the XTAG grammar to produce a lattice of all possible linearizations that are compatible with the underspecified supertagged tree and the XTAG. Underspecifications lead to disjunctions in the lattice. We rank these word sequences in the order of their likelihood by composing the lattice with a finite-state machine representing a trigram language model. This model has been constructed from 1,000,0000 words of Wall Street Journal corpus. The LP Chooser then picks the best path through the lattice resulting from the composition using.

3 Representing Meaning

If lexical choice is the choice of meaning-bearing lexemes in order to convey meaning, the question arises what the input to the lexical choice module should be, i.e., how to represent pre-lexical meaning. A popular way of representing meanings is to use the synonym sets (or *synsets* for short) defined in WordNet (Fellbaum, 1997). Each synset is a set of words of the same class which are roughly synonymous in one of their meanings. For example, the verb *fear* has five senses, two of which correspond to these synsets:[3]

Sense	Synset
1	fear, dread
2	reverence, fear, revere, venerate

The synsets partition the set of all senses of all lexemes, each corresponding to a distinct meaning. Thus, they are suitable representations in the input to the sentence plan-

[3]The other three senses correspond to synsets which contain only *fear*.

ner. If we had a corpus in which each word were annotated with its sense (i.e., a WordNet synset), then we could learn a mapping from meanings to lexeme. Unfortunately, no extensive sense-tagged corpus currently exists, though there are efforts underway to create one. Nonetheless, it is clear that sense-tagging is more complex a task than syntactic tagging, and therefore it will in the foreseeable future be a formidable task to sense-tag a new corpus. Recall that one of the motivations for stochastic NLG is the ability to quickly port to new domains or even languages.

Therefore, we have chosen to represent meaning in a more sloppy manner, namely by using the union of all the synsets of a lexeme. We refer to these sets as *supersynsets*. For example, the supersynset for *estimate* contains all the nouns given above in the table:

> Supersynset for verb *fear*: *fear, dread, reverence, revere, venerate*

If we don't assume we know the part-of-speech of a lexeme, we get the union of all synsets of the lexeme in all possible parts-of-speech:

> Supersynset for POS-unspecified *fear*: *fear, dread, reverence, revere, venerate, fearfulness, fright, concern, care*

While a synset represents a meaning, a supersynset represents *meaning potential*: it encodes all possible meanings of a lexeme. Unlike synsets, supersynsets do not form a partition of anything useful. Each lexeme is associated with exactly one supersynset; a supersynset may be associated with more than one lexeme, but need not be, even if it contains more than one lexeme. However, if lexeme l_1 is in the supersynset of lexeme l_2, then l_2 is of course also a member of the supersynset of l_1. Clearly, it is straightforward (and deterministic) to map from a representation based on lexemes to one based on supersynsets.

Since, for practical reasons, we must work with supersynsets and not synsets, we are framing the lexical choice question not as the task of choosing lexical items to convey a meaning specified in the input, but rather as a task of choosing the most appropriate synonym for a lexeme specified in the input (where the lexeme represents its own meaning potential). Thus, the input to the Lexeme Chooser is formally the same as the input to the Syntax Chooser, namely the tree shown in Figure 1; however, these node labels represent meaning potentials rather than actual lexemes, and Lexeme Chooser may or may not change the label when it determines the lexeme. Note that we can use the approach presented in this paper in order to perform the lexical choice task as it is usually framed: if we are given synsets (i.e., meanings) in the input representation, we can choose a member of that synset and then proceed in the manner described in this paper. However, we are not guaranteed to obtain a synonym from the original synset.

We assume that the structural relation between the lexemes is not changed during lexical or syntactic choice. Since the input representation may be underspecified with respect to the type of dependency relation (adjunct or argument, which type of argument), all that remains fixed is the the choice of heads and dependents. But it is well known that there are synonymous realizations in which meanings are distributed in different structural configurations, for example *we walked across the state* and *we traversed the state on foot*.[4] The issue of dealing with such structural paraphrases is a complex one and presumably requires corpora which are annotated in more complex ways than the syntactically annotated corpora available to us today.

4 Lexical Choice: Basic Architecture

In the architecture diagram in Figure 2, lexical choice happens before syntactic choice. However, *a priori* there are other architectural options as well. In this section, we inves-

[4] These kinds of examples are known as *structural divergences* when found in the context of machine translation (Dorr, 1994).

tigate three basic options of integrating lexical with syntactic choice: lexical choice before syntactic choice; integrated simultaneous lexical and syntactic choice; and lexical choice after syntactic choice. We observe that these three possible architectures do not accomplish exactly the same tasks. If we perform lexical choice after syntactic choice, then the range of possible synonyms is reduced: not only must we choose a member of the supersynset of the input lexeme, but it must also be compatible with the syntactic choice already made. For example, if the input lexeme is *give*, we can choose *donate* as a synonym. However, if we first choose a double-object construction (*give a car to the organization*) as the syntactic realization of the lexeme yet to be chosen, then *donate* is excluded as a lexical choice. The task in the syntactic-before-lexical-choice architecture is thus easier than the task of the lexical-before-syntactic-choice (or lexical-with-syntactic-choice) architecture, since the choice is more constrained.

The first task – that of the lexical-before-syntactic-choice (or lexical-with-syntactic-choice) architecture – is thus to take an input tree of the form shown in Figure 1, and to choose lexemes from the supersynsets associated with each lexeme, and then (or simultaneously) to choose supertags for the chosen lexemes. The second task – that of the syntactic-before-lexical-choice architecture – is to take a tree of the form shown in Figure 3, and to choose lexemes from the supersynsets associated with each lexeme which are compatible with the already made syntactic choices.

For the first task, we distinguish two types of architectures, lexical-before-syntactic-choice and lexical-with-syntactic-choice. In both architectures, we use top-down algorithms in which decisions are first made for a mother node before her daughters are dealt with. For the first lexical-before-syntactic-choice architecture, we use four different models.

- **LexSyn**: We choose the lexeme of the daughter node based on her mother's lexeme and supertag. More precisely, given a mother's lexeme l_m and supertag s_m, we find the daughter lexeme l_d that maximizes $p(l_d|l_m, s_m)$.[5]

- **Lex**: We choose the lexeme of the daughter node based on her mother's lexeme only. More precisely, we find the daughter lexeme l_d that maximizes $p(l_d|l_m)$.

- **Syn**: We choose the lexeme of the daughter node based on her mother's supertag only. More precisely, we find the daughter lexeme l_d that maximizes $p(l_d|s_m)$.

- **DtrLex**: We choose the lexeme of the daughter node based on the simple lexical frequency of the lexemes in the daughter's supersynset, without regard to the mother. More precisely, we find the daughter lexeme l_d that maximizes $p(l_d)$.

Since sparseness of data is a major problem, we combine these models, backing off from one to a less specific one as needed. For example, **LexSyn-DtrLex** is a model in which we take each member of the daughter's supersynset and choose the one with the highest value for $p(l_d|l_m, s_m)$. If no data is available for any lexemes of the daughter's supersynset, then we backoff and choose the lexeme from the supersynset which is most frequent in the training corpus. In all models, if the model does not result in a lexical choice (because of unseen data), we randomly choose a lexeme from the supersynset. (Note that **DtrLex** almost always results in a choice.)

For the lexical-with-syntactic-choice architecture, we use a single model:

- **LexSynJoint**: We choose the lexeme and the supertag of the daughter node based on her mother's lexeme and supertag. More precisely, given a mother's

[5]Note that this model implies that lexical and syntactic choice are performed interleaved: for a given node, first the lexeme is chosen, then its supertag. Then attention shifts to the lexeme's daughters, and for each daughter, when the lexeme is chosen, the mother's supertag is already available. *Interleaved* execution of lexical and syntactic choice should not be confused with *combined* execution of lexical and syntactic choice, discussed below.

lexeme l_m and supertag s_m, we find the daughter lexeme l_d and the daughter supertag s_d that maximize $p(l_d, s_d|l_m, s_m)$.

Here, the only backoff we consider is to **DtrLex**, with subsequent independent syntactic choice.

For the second task – that of the syntactic-before-lexical-choice architecture – we define two types of algorithms. In the **tree** algorithm, we choose the lexemes in the tree model after we have chosen the syntax, in a second pass through the tree. Each lexeme is chosen based on its own supertag, its mother node's lexeme, and its mother's supertag. If no data is available, we first back off to considering its own supertag and its mother node's lexeme, then just its own supertag, and finally to the frequency of the daughter lexemes. In the **linear** algorithm, we include all the alternative lexemes in the lattice passed to the LP Chooser, where the language model imposes a choice of lexeme. For this task, it was necessary to change the backoff parameters in the language model in such a way that unseen words were weighted very low, unlike cases in which the language model is used for recognition or parsing. For each of these two algorithms, we can consider all members of the supersynset of the lexeme, or only those with the part-of-speech as specified in the input representation. This gives us four algorithms, **tree-all**, **tree-pos**, **linear-all**, **linear-pos**.

5 Evaluation and Results

The evaluation is based on comparison with a test corpus of 100 sentences randomly chosen from the Penn Tree Bank WSJ corpus (not used in training, of course). The average sentence length is 16.69 words, and the average lexical ambiguity per word is 4.8. (The maximum lexical ambiguity is 69 for *take*.) The premise of the evaluation is that the goal of generation is to match the gold standard as closely as possible, even if other results would also be acceptable or even appropriate; in some applications, variation may be useful, in which case this evaluation may not be appropriate.

For the evaluation, we use two different metrics. The first metric, **string accuracy**, is based on similar metrics used in machine translation. We compare the output of FERGUS to the gold standard sentence and count the number of move and substitution operations that need to be performed to transform the actual output into the gold standard. This number is subtracted from and then divided by the length of the sentence, yielding a number between 0 and 1, with 1 the best score. For a more detailed discussion of the issue of metrics in evaluation, of the metrics we have used, and of an experiment relating human judgments to these metrics, see (Bangalore et al., 2000). String accuracy measures the performance of the entire FERGUS system.

For the second metric, **bag accuracy**, we disregard linear order and calculate recall and precision on the bag (i.e., multiset) of lexemes in the generated string as compared to the bag of lexemes in the gold standard sentence. Since the number of words in the two sentences is usually the same, recall and precision are very close, and we report as set accuracy their average, which can be interpreted as the percentage of correctly generated lexemes. Bag accuracy measures the performance of the Lexeme Chooser only.

The results for the first task (the lexical-before-syntactic-choice and lexical-with-syntactic-choice architectures) are summarized in the table in Figure 4. As we can see, **DtrLex** on its own far outperforms any other algorithm that does not back off to it. All of the algorithms that have one backoff to **DtrLex** perform at the same level, as does **LexSyn-Syn-DtrLex**, while **LexSyn-Lex-DtrLex** performs slightly better.

The results for the second task (choosing lexemes after syntactic choice) are summarized in the table in Figure 5. Because of the different tasks, the results cannot be meaningfully compared to Figure 4, but we can see that, as expected, FERGUS performs better on this task than on the first task. Without part-of-speech information, the Tree model perform better than the linear model, but the advantage is lost when part-of-speech infor-

Model	String Accuracy	Bag Accuracy
random	0.34	0.69
LexSyn	0.34	0.69
LexSyn-Lex	0.48	0.79
LexSyn-Lex-DtrLex	0.63	0.88
LexSyn-Syn-DtrLex	0.62	0.87
LexSyn-DtrLex	0.62	0.87
Lex	0.48	0.79
Lex-DtrLex	0.62	0.87
LexSynJoint	0.45	0.77
LexSynJoint-DtrLex	0.61	0.87
DtrLex	0.59	0.85

Figure 4: Summary of results for the lexical-before-syntactic-choice and lexical-with-syntactic-choice architectures for the first task

mation is used as well.

We now draw tentative conclusions for future work from these results. Clearly, using information from the mother node increases performance, if only slightly. However, there is no reliable evidence that the lexical-before-syntactic-choice architecture outperforms the lexical-with-syntactic-choice architectures or *vice versa*. There is also no evidence that the mother's supertag affects lexical choice in the daughter.[6] As a result, we conclude that for the first task we can restrict our attention to a simplified architecture in which lexical choice for the entire tree occurs before syntactic choice for the entire tree (i.e., the two choices are neither simultaneous nor interleaved). Furthermore, the importance of the backoff **DtrLex** model shows the importance of the sparse data problem. Thus, the model we will concentrate on is **Lex-DtrLex**.

6 Context in Lexical Choice

We now investigate how much context in the syntax tree is needed in order to optimize lexical choice. In the experiments reported in

[6]However, we have not used information about the (semantic or syntactic) role of the daughter, and it stands to reason that this information can help in lexical choice.

Model	String Accuracy	Bag Accuracy
Tree-All	0.69	0.93
Tree-Pos	0.68	0.93
Linear-All	0.67	0.90
Linear-Pos	0.70	0.93

Figure 5: Summary of results for the syntactic-before-lexical-choice algorithms for the second task

the previous section, a weakness quickly becomes apparent: while lexical choice depends on the mother node, the root lexeme cannot be chosen in this manner. In the experiments reported above we simply use lexical frequency at the root. A wrong choice at the root, however, can lead to a cascade of subsequent wrong choices at lower nodes as they all depend on previously made wrong choices. We therefore will base lexical choice on an extended notion of tree context which includes the daughters.

Because the number of daughters in a dependency tree is not *a priori* bound, it is impossible to create a probability model of the type $p(l|l_{d_1}, \ldots, l_{d_n})$, where l is the node's lexeme and l_{d_i} is the lexeme of the ith daughter. Instead, we make an independence assumption for the daughters and calculate the node-daughter probabilities separately. Note that the daughters' lexemes have not actually been chosen yet, so we use the daughters' meaning potential rather than their lexemes, as discussed in Section 3.

The results are shown in Figure 6. Here, we have renamed the **Lex** model to **Node**, to reflect the fact that only information about the node whose lexeme is being chosen is taken into account. Model **Mother-node** is the **Lex-DtrLex** model from Figure 5: we determine a node's lexeme based on the mother's lexeme. In model **Node-daughters** the choice is based on the daughters only. Finally, model **Mother-node-daughters** considers both the mother's lexeme and the daughters' (excluding, of course, the mother in case of the root node, and daughters in case

Model	String Accuracy	Bag Accuracy
Random	0.34	0.69
Node	0.59	0.85
Mother-node	0.62	0.87
Node-daughters	0.63	0.88
Mother-node-daughters	0.67	0.90

Figure 6: Summary of results for different syntactic context usage

of leaf nodes).

As we can see, extending the context in the tree from which we draw information can significantly improve performance beyond the baseline of choosing the most frequent lexeme from a supersynset (model **Node** in Figure 6).

7 Using POS Information

For our sample sentence in Figure 3, the **Mother-node-daughters** model chooses *concern* for the root node. The verb *concern* is not a synonym of the verb *fear*, but the noun *concern* is a synonym of the noun *fear*. For many applications, it is quite reasonable to assume that we know the lexical class of some or all labels in the input structure. We therefore investigated the performance of our best model so far, **Mother-node-daughters**, if we restrict our choice to those lexemes that have the same part-of-speech as the input meaning potential. Since the supersynsets associated with each input lexeme are now smaller, the performance cannot be inferior than in the general case (at least as measured by bag accuracy). Indeed, for our example sentence, we now obtain the desired choice, namely *fear*. However, to our surprise, overall performance on our test set increased only very slightly, to a string accuracy of 0.68 (from 0.67) and to a bag accuracy of 0.91 (from 0.90). We conjecture that most errors are within the correct lexical class, and thus not saved by taking part-of-speech into account.

8 Conclusion

We have seen that the performance of lexical choice can be improved over the baseline of choosing the most frequent member of a supersynset by taking syntactic context in the dependency tree into account. Since sparseness of data is a major problem, we intend to train the models on the new Brown corpus of 30,000,000 words.

References

Srinivas Bangalore and Aravind Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–266.

Srinivas Bangalore and Owen Rambow. 2000. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*, Saarbrücken, Germany.

Srinivas Bangalore, Owen Rambow, and Steve Whittaker. 2000. Evaluation metrics for generation. In *Proceedings of the First International Natural Language Generation Conference (INLG2000)*, Mitzpe Ramon, Israel.

Bonnie J. Dorr. 1994. Machine translation divergences: A formal description and proposed solution. *Computational Linguistics*, 20(4):597–635.

Christiane Fellbaum. 1997. *WordNet: An Electronic Lexical Database*. MITPress, Cambridge, MA.

Aravind K. Joshi. 1987. An introduction to Tree Adjoining Grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*, pages 87–115. John Benjamins, Amsterdam.

Irene Langkilde and Kevin Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *36th Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING-ACL'98)*, pages 704–710, Montréal, Canada.

The XTAG-Group. 1999. A lexicalized Tree Adjoining Grammar for English. Technical report, Institute for Research in Cognitive Science, University of Pennsylvania.