Exploiting a Probabilistic Hierarchical Model for Generation

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

Abstract

Previous stochastic approaches to generation do not include a tree-based representation of syntax. While this may be adequate or even advantageous for some applications, other applications profit from using as much syntactic knowledge as is available, leaving to a stochastic model only those issues that are not determined by the grammar. We present initial results showing that a tree-based model derived from a tree-annotated corpus improves on a tree model derived from an unannotated corpus, and that a tree-based stochastic model with a hand-crafted grammar outperforms both.

1 Introduction

For many applications in natural language generation (NLG), the range of linguistic expressions that must be generated is quite restricted, and a grammar for generation can be fully specified by hand. Moreover, in many cases it is very important not to deviate from certain linguistic standards in generation, in which case hand-crafted grammars give excellent control. However, in other applications for NLG the variety of the output is much bigger, and the demands on the quality of the output somewhat less stringent. A typical example is NLG in the context of (interlingua- or transfer-based) machine translation. Another reason for relaxing the quality of the output may be that not enough time is available to develop a full grammar for a new target language in NLG. In all these cases, stochastic ("empiricist") methods provide an alternative to hand-crafted ("rationalist") approaches to NLG. To our knowledge, the first to use stochastic techniques in NLG were Langkilde and Knight (1998a) and (1998b). In this paper, we present FERGUS (Flexible Empiricist/Rationalist Generation Using Syntax). FERGUS follows Langkilde and Knight's seminal work in using an n-gram language model, but we augment it with a tree-based stochastic model and a traditional tree-based syntactic grammar. More recent work on aspects of stochastic generation include (Langkilde and Knight, 2000), (Malouf, 1999) and (Ratnaparkhi, 2000).

Before we describe in more detail how we use stochastic models in NLG, we recall the basic tasks in NLG (Rambow and Korelsky, 1992; Reiter, 1994). During **text planning**, content and structure of the target text are determined to achieve the overall communicative goal. During **sentence planning**, linguistic means – in particular, lexical and syntactic means – are determined to convey smaller pieces of meaning. During **realization**, the specification chosen in sentence planning is transformed into a surface string, by linearizing and inflecting words in the sentence (and typically, adding function words). As in the work by Langkilde and Knight, our work ignores the text planning stage, but it does address the sentence planning and the realization stages.

The structure of the paper is as follows. In Section 2, we present the underlying grammatical formalism, lexicalized tree-adjoining grammar (LTAG). In Section 3, we describe the architecture of the system, and some of the modules. In Section 4 we discuss three experiments. In Section 5 we compare our work to that of Langkilde and Knight (1998a). We conclude with a summary of on-going work.

2 Modeling Syntax

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, 1987a). In

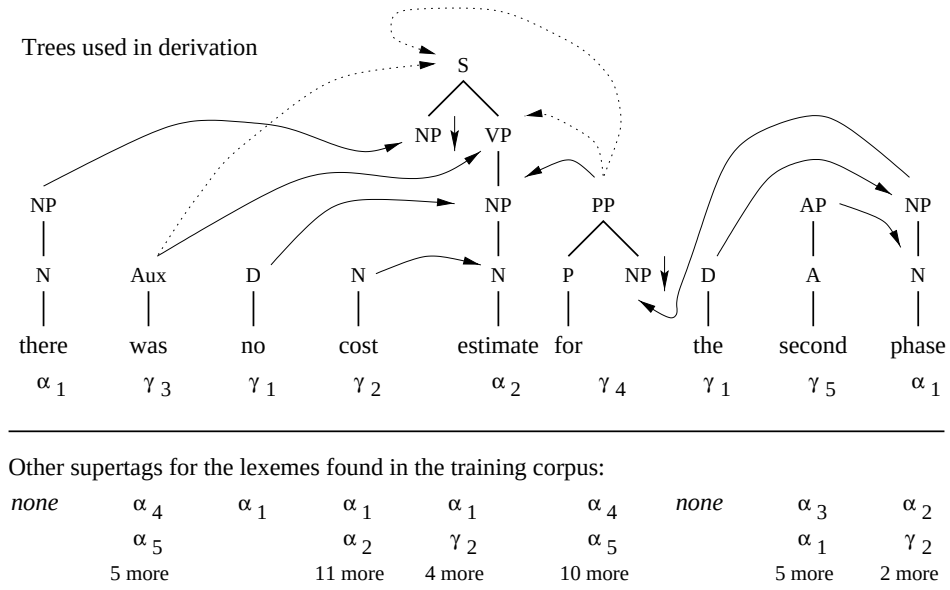

Figure 1: An excerpt from the XTAG grammar to derive *There was no cost estimate for the second phase*; dotted lines show possible adjunctions that were not made

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 graphical representation, nodes at which substitution can take place are marked with down-arrows. 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; as a result we obtain a lexicalized TAG or LTAG. Since each lexical item is associated with a whole tree (rather than just a phrase-structure rule, for example), we can specify both the predicate-argument structure of the lexeme (by including nodes at which its arguments must substitute) and morpho-syntactic constraints such as subject-verb agreement within the structure associated with the lexeme. This property is referred to as TAG's *extended domain of locality*. Note that in an LTAG, there is no distinction between lexicon and grammar. A sample grammar is shown in Figure 1.

We depart from XTAG in our treatment of trees for adjuncts (such as adverbs), and instead follow McDonald and Pustejovsky (1985). While in XTAG the elementary tree for an adjunct contains phrase structure that attaches the adjunct to nodes in another tree with the specified label (say, VP) from the specified direction (say, from the left), in our system the trees for adjuncts simply express their active valency, but not how they connect to the lexical item they modify. This information is kept in the *adjunction table* which is associated with the grammar; an excerpt is shown in Figure 2. Trees that can adjoin to other trees (and have entries in the adjunction table) are called *gamma-trees*, the other trees (which can only be substituted into other trees) are *alpha-trees*.

stag	anchored by	adjoins to	direction
γ_1	Det	NP	right
γ_2	N	N	right
γ_3	Aux	S, VP	right
γ_4	Prep	NP, VP	left
	or	S	right
γ_5	Adj	N	right

Figure 2: Adjunction table for grammar fragment

Note that we can refer to a tree by a combination of its name, called its supertag, and its anchor. For example, α_1 is the supertag of an alpha-tree anchored by a noun that projects up to NP, while γ_2 is the supertag of a gamma tree anchored by a noun that only projects to N (we

assume adjectives are adjoined at N), and, as the adjunction table shows, can right-adjoin to an N. So that **estimate**$_{\alpha_2}$ is a particular tree in our LTAG grammar. Another tree that a supertag can be associated with is α_2, which represents the predicative use of a noun.[1] Not all nouns are associated with all nominal supertags: the expletive *there* is only an α_1.

When we derive a sentence using an LTAG, we combine elementary trees from the grammar using adjunction and substitution. For example, to derive the sentence *There was no cost estimate for the second phase* from the grammar in Figure 1, we substitute the tree for *there* into the tree for *estimate*. We then adjoin in the trees for the auxiliary *was*, the determiner *no*, and the modifying noun *cost*. Note that these adjunctions occur at different nodes: at VP, NP, and N, respectively. We then adjoin in the preposition, into which we substitute *phase*, into which we adjoin *the* and *second*. Note that all adjunctions are by gamma trees, and all substitution by alpha trees.

If we want to represent this derivation graphically, we can do so in a *derivation tree*, which we obtain as follows: whenever we adjoin or substitute a tree t_1 into a tree t_2, we add a new daughter labeled t_1 to the node labeled t_2. As explained above, the name of each tree used is the lexeme along with the supertag. (We omit the address at which substitution or adjunction takes place.) The derivation tree for our derivation is shown in Figure 3. As can be seen, this structure is a dependency tree and resembles a representation of lexical argument structure.

Joshi (1987b) claims that TAG's properties make it particularly suited as a syntactic representation for generation. Specifically, its extended domain of locality is useful in generation for localizing syntactic properties (including word order as well as agreement and other morphological processes), and lexicalization is useful for providing an interface from semantics (the derivation tree represent the sentence's predicate-argument structure). Indeed, LTAG has been used extensively in generation, starting with (McDonald and Pustejovsky, 1985).

[1]Sentences such as *Peter is a doctor* can be analyzed with with *be* as the head, as is more usual, or with *doctor* as the head, as is done in XTAG because the *be* really behaves like an auxiliary, not like a full verb.

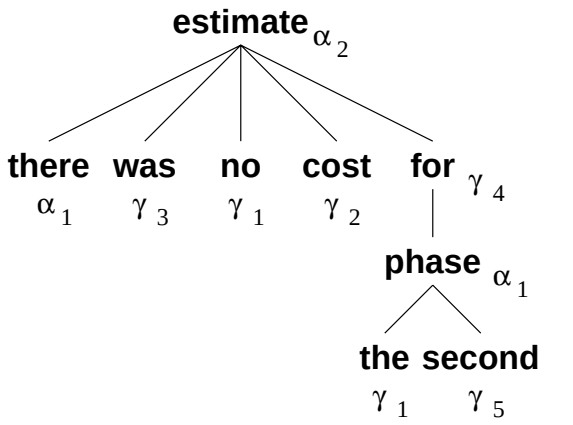

Figure 3: Derivation tree for LTAG derivation of *There was no cost estimate for the second phase*

3 System Overview

FERGUS is composed of three modules: the Tree Chooser, the Unraveler, and the Linear Precedence (LP) Chooser. The input to the system is a dependency tree as shown in Figure 4. Note that 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 Unraveler then uses the XTAG grammar to produce a lattice of all possible linearizations that are compatible with the supertagged tree and the XTAG. The LP Chooser then chooses the most likely traversal of this lattice, given a language model. We discuss the three components in more detail.

The Tree Chooser draws on a tree model, which is a representation of XTAG derivation for 1,000,000 words of the Wall Street Journal.[3] The Tree Chooser makes the simplifying as-

[2]In the system that we used in the experiments described in Section 4, all words (including function words) need to be present in the input representation, fully inflected. This is of course unrealistic for applications. In this paper, we only aim to show that the use of a Tree Model improves performance of a stochastic generator. See Section 6 for further discussion.

[3]This was constructed from the Penn Tree Bank using some heuristics, since the Penn Tree Bank does not contain full head-dependent information; as a result of the use of heuristics, the Tree Model is not fully correct.

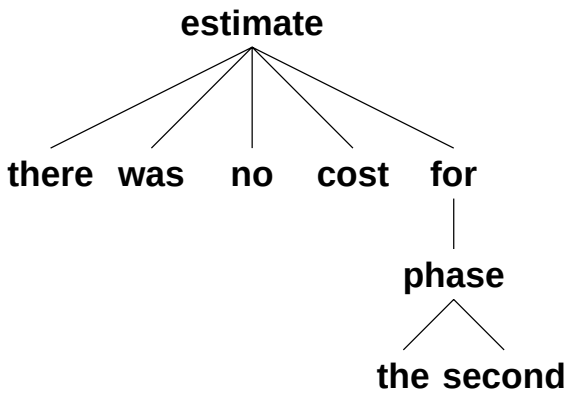

Figure 4: Input to FERGUS

sumptions that the choice of a tree for a node depends only on its daughter nodes, thus allowing for a top-down dynamic programming algorithm. Specifically, a node η in the input structure is assigned a supertag s so that the probability of finding the treelet composed of η with supertag s and all of its daughters (as found in the input structure) is maximized, and such that s is compatible with η's mother and her supertag s_m. Here, "compatible" means that the tree represented by s can be adjoined or substituted into the tree represented by s_m, according to the XTAG grammar. For our example sentence, the input to the system is the tree shown in Figure 4, and the output from the Tree Chooser is the tree as shown in Figure 3. Note that while a derivation tree in TAG fully specifies a derivation and thus a surface sentence, the output from the Tree Chooser does not. There are two reasons. Firstly, as explained at the end of Section 2, for us trees corresponding to adjuncts are underspecified with respect to the adjunction site and/or the adjunction direction (from left or from right) in the tree of the mother node, or they may be unordered with respect to other adjuncts (for example, the famous adjective ordering problem). Secondly, supertags may have been chosen incorrectly or not at all.

The Unraveler takes as input the semi-specified derivation tree (Figure 3) and produces a word lattice. Each node in the derivation tree consists of a lexical item and a supertag. The linear order of the daughters with respect to the head position of a supertag is specified in the XTAG grammar. This information is consulted to order the daughter nodes

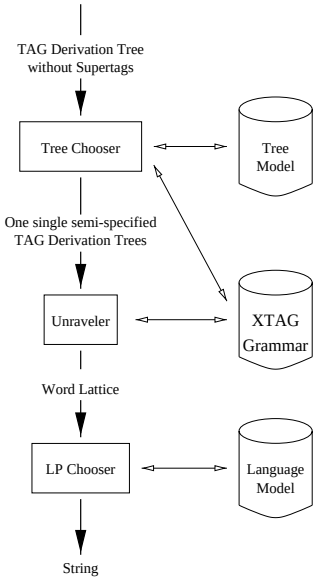

Figure 5: Architecture of FERGUS

with respect to the head at each level of the derivation tree. In cases where a daughter node can be attached at more than one place in the head supertag (as is the case in our example for *was* and *for*), a disjunction of all these positions are assigned to the daughter node. A bottom-up algorithm then constructs a lattice that encodes the strings represented by each level of the derivation tree. The lattice at the root of the derivation tree is the result of the Unraveler. The resulting lattice for the example sentence is shown in Figure 6.

The lattice output from the Unraveler encodes all possible word sequences permitted by the derivation structure. 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. We pick the best path through the lattice resulting from the composition using the Viterbi algorithm, and this top ranking word sequence is the output of the LP Chooser.

4 Experiments and Results

In order to show that the use of a tree model and a grammar does indeed help performance, we performed three experiments:

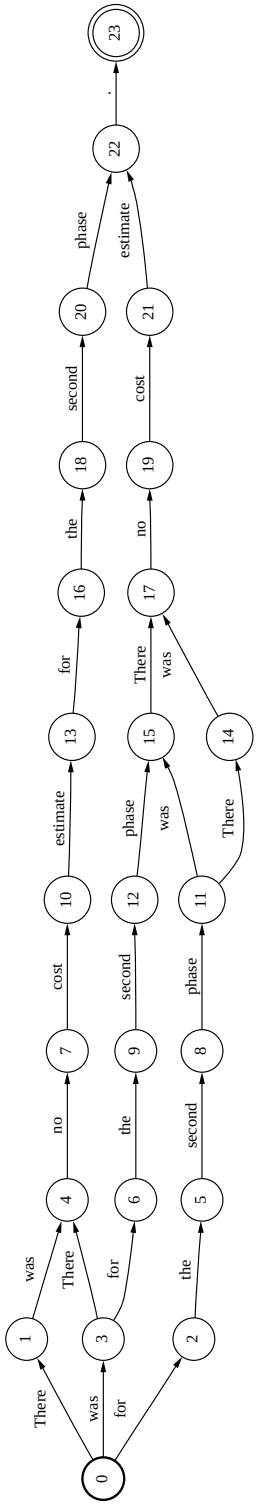

Figure 6: Word lattice for example sentence after Tree Chooser and Unraveler using the supertag-based model

- For the baseline experiment, we impose a random tree structure for each sentence of the corpus and build a Tree Model whose parameters consist of whether a lexeme l_d precedes or follows her mother lexeme l_m. We call this the Baseline Left-Right (LR) Model. This model generates *There was estimate for phase the second no cost .* for our example input.
- In the second experiment, we derive the parameters for the LR model from an annotated corpus, in particular, the XTAG derivation tree corpus. This model generates *There no estimate for the second phase was cost .* for our example input.
- In the third experiment, as described in Section 3, we employ the supertag-based tree model whose parameters consist of whether a lexeme l_d with supertag s_d is a dependent of l_m with supertag s_m. Furthermore we use the supertag information provided by the XTAG grammar to order the dependents. This model generates *There was no cost estimate for the second phase .* for our example input, which is indeed the sentence found in the WSJ.

As in the case of machine translation, evaluation in generation is a complex issue. We use two metrics suggested in the MT literature (Alshawi et al., 1998) based on string edit distance between the output of the generation system and the reference corpus string from the WSJ. These metrics, simple accuracy and generation accuracy, allow us to evaluate without human intervention, automatically and objectively.[4]

Simple accuracy is the number of insertion (I), deletion (D) and substitutions (S) errors between the target language strings in the test corpus and the strings produced by the generation model. The metric is summarized in Equation (1). R is the number of tokens in the target string. This metric is similar to the string distance metric used for measuring speech recognition accuracy.

$$SimpleAccuracy = (1 - \frac{I + D + S}{R}) \qquad (1)$$

[4]We do not address the issue of whether these metrics can be used for comparative evaluation of other generation systems.

Tree Model	Simple Accuracy	Generation Accuracy	Average time per sentence
Baseline LR Model	41.2%	56.2%	186ms
Treebank derived LR Model	52.9%	66.8%	129ms
Supertag-based Model	58.9%	72.4%	517ms

Table 1: Performance results from the three tree models.

Unlike speech recognition, the task of generation involves reordering of tokens. The simple accuracy metric, however, penalizes a misplaced token twice, as a deletion from its expected position and insertion at a different position. We use a second metric, Generation Accuracy, shown in Equation (2), which treats deletion of a token at one location in the string and the insertion of the same token at another location in the string as one single movement error (M). This is in addition to the remaining insertions (I') and deletions (D').

$$GenerationAccuracy = (1 - \frac{M + I' + D' + S}{R}) \quad (2)$$

The simple accuracy, generation accuracy and the average time for generation of each test sentence for the three experiments are tabulated in Table 1. The test set consisted of 100 randomly chosen WSJ sentence with an average length of 16 words. As can be seen, the supertag-based model improves over the LR model derived from annotated data and both models improve over the baseline LR model.

Supertags incorporate richer information such as argument and adjunct distinction, and number and types of arguments. We expect to improve the performance of the supertag-based model by taking these features into account.

In ongoing work, we have developed tree-based metrics in addition to the string-based presented here, in order to evaluate stochastic generation models. We have also attempted to correlate these quantitative metrics with human qualitative judgements. A detailed discussion of these experiments and results is presented in (Bangalore et al., 2000).

5 Comparison with Langkilde & Knight

Langkilde and Knight (1998a) use a hand-crafted grammar that maps semantic representations to sequences of words with linearization constraints. A complex semantic structure is translated to a lattice, and a bigram language model then chooses among the possible surface strings encoded in the lattice.

The system of Langkilde & Knight, Nitrogen, is similar to FERGUS in that generation is divided into two phases, the first of which results in a lattice from which a surface string is chosen during the second phase using a language model (in our case a trigram model, in Nitrogen's case a bigram model). However, the first phases are quite different. In FERGUS, we start with a lexical predicate-argument structure, while in Nitrogen, a more semantic input is used. FERGUS could easily be augmented with a preprocessor that maps a semantic representation to our syntactic input; this is not the focus of our research. However, there are two more important differences. First, the hand-crafted grammar in Nitrogen maps directly from semantics to a linear representation, skipping the arborescent representation usually favored for the representation of syntax. There is no stochastic tree model, since there are no trees. In FERGUS, initial choices are made stochastically based on the tree representation in the Tree Chooser. This allows us to capture stochastically certain long-distance effects which n-grams cannot, such as separation of parts of a collocations (such as *perform an operation*) through interposing adjuncts (*John performed a long, somewhat tedious, and quite frustrating operation on his border collie*). Second, the hand-crafted grammar used in FERGUS was crafted independently from the need for generation and is a purely declarative representation of English syntax. As

such, we can use it to handle morphological effects such as agreement, which cannot in general be done by an n-gram model and which are, at the same time, descriptively straightforward and which are handled by all non-stochastic generation modules.

6 Conclusion and Outlook

We have presented empirical evidence that using a tree model in addition to a language model can improve stochastic NLG.

FERGUS as presented in this paper is not ready to be used as a module in applications. Specifically, we will add a morphological component, a component that handles function words (auxiliaries, determiners), and a component that handles punctuation. In all three cases, we will provide both knowledge-based and stochastic components, with the aim of comparing their behaviors, and using one type as a back-up for the other type. Finally, we will explore FERGUS when applied to a language for which a much more limited XTAG grammar is available (for example, specifying only the basic sentence word order as, say, SVO, and specifying subject-verb agreement). In the long run, we intend FERGUS to become a flexible system which will use hand-crafted knowledge as much as possible and stochastic models as much as necessary.

References

Hiyan Alshawi, Srinivas Bangalore, and Shona Douglas. 1998. Automatic acquisition of hierarchical transduction models for machine tr anslation. In *Proceedings of the 36th Annual Meeting Association for Computational Linguistics*, Montreal, Canada.

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

Srinivas Bangalore, Owen Rambow, and Steve Whittaker. 2000. Evaluation Metrics for Generation. In *Proceedings of International Conference on Natural Language Generation*, Mitzpe Ramon, Isreal.

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

Aravind K. Joshi. 1987b. The relevance of tree adjoining grammar to generation. In Gerard Kempen, editor, *Natural Language Generation: New Results in Artificial Intelligence, Psychology and Linguistics*, pages 233–252. Kluwer Academic Publishers, Dordrecht/Boston/Lancaster.

Irene Langkilde and Kevin Knight. 1998a. 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.

Irene Langkilde and Kevin Knight. 1998b. The practical value of n-grams in generation. In *Proceedings of the Ninth International Natural Language Generation Workshop (INLG'98)*, Niagara-on-the-Lake, Ontario.

Irene Langkilde and Kevin Knight. 2000. Forest-based statistical sentence generation. In *Proceedings of First North American ACL*, Seattle, USA, May.

Robert Malouf. 1999. Two methods for predicting the order of prenominal adjectives in english. In *Proceedings of CLIN99*.

David D. McDonald and James D. Pustejovsky. 1985. Tags as a grammatical formalism for generation. In *23rd Meeting of the Association for Computational Linguistics (ACL'85)*, pages 94–103, Chicago, IL.

Owen Rambow and Tanya Korelsky. 1992. Applied text generation. In *Third Conference on Applied Natural Language Processing*, pages 40–47, Trento, Italy.

Adwait Ratnaparkhi. 2000. Trainable methods for surface natural language generation. In *Proceedings of First North American ACL*, Seattle, USA, May.

Ehud Reiter. 1994. Has a consensus NL generation architecture appeared, and is it psycholinguistically plausible? In *Proceedings of the 7th International Workshop on Natural Language Generation*, pages 163–170, Maine.

The XTAG-Group. 1999. A lexicalized Tree Adjoining Grammar for English. Technical Report `http://www.cis.upenn.edu/~xtag/tech-report/tech-report.html`, The Institute for Research in Cognitive Science, University of Pennsylvania.