Using TAGs, a Tree Model, and a Language Model for Generation

Srinivas Bangalore et Owen Rambow

AT&T Labs-Research, B233
180 Park Ave, PO Box 971
Florham Park, NJ 07932-0971, USA
srini, rambow@research.att.com

Abstract

Previous stochastic approaches to sentence realization 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. In this paper, we present three results in the context of surface realization: a stochastic tree model derived from a parsed corpus outperforms a tree model derived from unannotated corpus; exploiting a hand-crafted grammar in conjunction with a tree model outperforms a tree model without a grammar; and exploiting a tree model in conjunction with a linear language model outperforms just the tree model.

1. Introduction

Most sentence realizers (systems that take a fairly shallow semantic or lexico-syntactic representation and return a surface string in the target language) are entirely grammar-based, including quite a few based on TAG (starting with (McDonald and Pustejovsky1985)). Generators using hand-crafted grammars are useful for constrained applications, when strict control over the output is needed, and when a sufficiently large grammar is available. Recently, (Langkilde and Knight1998a) and (1998b) have used stochastic techniques in NLG, by mapping semantic primitives to a set of possible ordered sequences of tokens, and assembling theses into a lattice. They then use a linear language model to select the best path through the lattice. Stochastic generators are useful when a large grammar is not available, or when the range of generated utterances is large.

To date, generators are either fully hand-crafted or entirely syntax-free, and use a stochastic model only at the level of linear strings. In this paper we present FERGUS (Flexible Empiricist/Rationalist Generation Using Syntax). FERGUS follows Knight and Langkilde's seminal work in using an n-gram language model, but we augment it with a tree-based stochastic model and a TAG grammar. We argue that the combination of all three key modules of our approach – tree model, TAG grammar, linear model – is crucial and improves over models using only a subset of these modules.

The structure of the paper is as follows. In Section 2, we start out by describing a modification to standard TAG that we have followed for the sake of generation. In Section 3, we describe the architecture of the system, and some of the modules. In Section 4 we discuss three experiments. We conclude with a summary of on-going and future work.

2. γ-Trees

We depart from standard TAG practice in our treatment of trees for adjuncts (such as adverbs or adjectives), and instead follow (McDonald and Pustejovsky1985) and (Rambow et al.1995). While in XTAG the elementary tree for an adjunct contains phrase structure that attaches the adjunct to a node 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 own phrase and argument structure (active valency), but not how they connect to the lexical item they modify (passive valency). Information about passive valency is kept in the *adjunction table* which is associated with the grammar. We call trees that can adjoin to other trees (and have entries in the adjunction table) γ-*trees*, the other trees (which can only be substituted into other trees) are α-*trees*, while β trees are now restricted to predicative auxiliary trees. Note that each γ-tree corresponds to a set of predicative auxiliary trees in a traditional TAG grammar (which share common phrase structure but attach differently).

3. System Overview

FERGUS is composed of three modules: the stochastic Tree Chooser, the grammar-based Unraveler, and the stochastic Linear Precedence (LP) Chooser. The input to the system is a dependency tree as shown in Figure 1 on the left. Note that the nodes are labeled only with lexemes, not with supertags. 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 Joshi1999), 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 (XTAG-Group1999) to produce a lattice of all possible linearizations that are compatible with the supertagged tree and the XTAG grammar. The LP Chooser then chooses the most likely traversal of this lattice, given a language model. We discuss the input representation and the three components in turn. For a more detailed overview over the system, see (Bangalore and Rambow2000b).

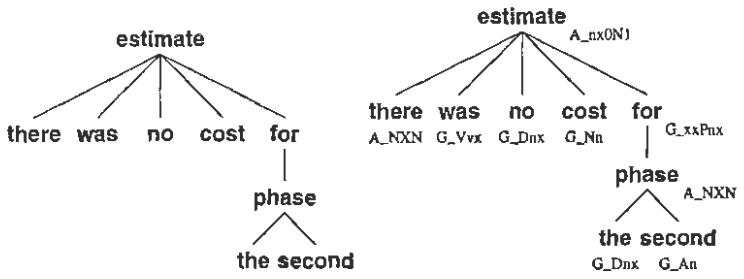

Figure 1: 80

3.1. *The Input to* FERGUS

As mentioned, the input to FERGUS is a dependency tree. We make three remarks.

First: we see the task of FERGUS as one of incremental specification. Clearly, a TAG derivation tree fully specifies a derivation. It consists of three types of information for each node: the supertag, the lexical anchor, and the address at which this tree is attached at the tree of the mother node (except of course for the root). In FERGUS, we assume that the input contains

only the lexeme, and the other information is added during the generation process. As a result, the input tree is actually semantically underspecified – for example, from the tree on the left in Figure 1, we could in theory obtain a sentence such as *cost was no estimate for the second phase there*, by choosing an α-tree for *cost* and an adverbial auxiliary tree for *there*. Thus we leave it to the corpus to determine how the lexemes relate to each other. Clearly, for many applications, we know which role the dependents of a lexeme play in the argument structure of their head, and FERGUS allows us to annotate dependents with a role feature (adj for adjuncts, func for function words, or for arguments an integer referring to the numbering of argument slots in the XTAG grammar). However, the option of leaving the role underspecified is useful in machine translation applications (when the parser cannot fully determine the syntactic roles in the source language), and for all applications because often it is difficult to determine whether a dependent is an argument or an adjunct (for example, *Baton Rouge* in *he disappeared from Baton Rouge*). In realizers that do not allow for underspecification, it is necessary to consult the linguistic data base (lexicon) of the realizer in order to construct valid inputs; FERGUS allows us to leave the role of some dependents open.

Second: 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 our three modules improves performance of a generator.

Third: as is well known, because of lexicalization, the derivation tree of TAG is a dependency tree. However, because of the definition of adjunction, there are cases in which the derivation tree is not the dependency tree as commonly assumed, in particular cases of clausal embedding using predicative auxiliary trees (Rambow and Joshi1996). Because our training corpus is annotated with standard dependency trees and not derivation trees, we assume standard dependency trees as input, and treat the footnode of predicative auxiliary trees as a substitution node. As a consequence, we do not currently exploit the full formal power of TAG and we are not able to generate long-distance dependencies. We intend to address this issue in future work.

3.2. The Tree Chooser

In general, for a given dependency tree, each node can be given more than one supertag in order to turn the tree into a valid derivation tree. If the syntactic roles of the daughter nodes are not fixed, then the subcategorization frame of the mother node needs to be chosen, but even if they are fixed, choices remain such as voice, and how to realize the arguments (for example, dative shift or topicalization). Ideally, we would have a correct set of rules for each choice and enough data in the generation process so that we can make the decision. (Stone and Doran1997) have shown how to integrate such rules into a TAG framework. However, the required research to find the correct rules is nowhere near completed and the data required in order to make such decisions is not always available in generation. An alternative is to assume a default ordering of choices, as does (Becker1998). This cuts back on the required off-line theoretical work and on-line data, but represents a rather inflexible solution. We have chosen to use a stochastic tree model sensitive to the lexemes of the mother and daughter nodes to make this choice.

The Tree Chooser draws on a tree model, which is a representation of an XTAG derivation for 1,000,000 words of the Wall Street Journal. It makes the simplifying assumptions 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 1 on the left, and the output from the Tree Chooser is the tree as shown in Figure 1 on the right. 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, because for us adjunct auxiliary trees are γ-trees and thus underspecified with respect to the adjunction site and/or the adjunction direction (from left or from right) in the tree of the mother node, and they may be unordered with respect to other adjuncts (for example, the famous adjective ordering problem). (See Section 2 above.) Furthermore, supertags may have been chosen incorrectly or not at all.

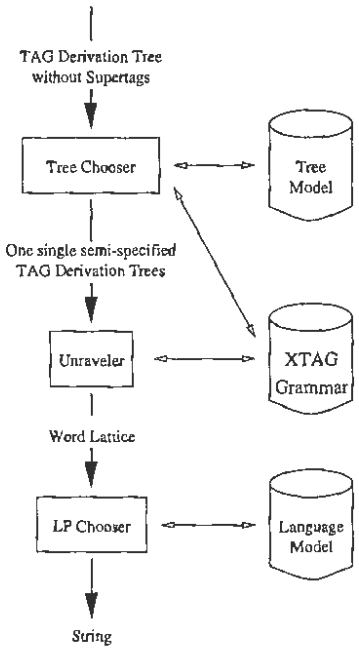

Figure 2: Architecture of FERGUS

3.3. The Unraveler

The Unraveler takes as input the semi-specified derivation tree (Figure 1 on the right) 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 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.

3.4. The Linear Precedence Chooser

The lattice output from the Unraveler encodes all possible word sequences permitted by the derivation structure. Again, it might be possible to develop rules to choose among possible adjunction sites for adverbs, or for choosing possible orderings of adjuncts at the same adjunction site (such as for the notorious adnominal adjective ordering problem). However, such research is not completed, and we instead propose to use a stochastic model in order to make this choice. We rank the word sequences encoded by the lattice 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, a grammar, and a linear model does indeed help performance, we performed four experiments:

- 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 (**TM-LM**), we derive the parameters for the LR model from an annotated corpus, in particular, the XTAG derivation tree corpus. Thus, we use a tree model and a linear language model, but not the TAG grammar. This model generates *There no estimate for the second phase was cost* . for our example input.

- In the third experiment (**TM-XTAG**), we use a tree model which has been trained on a corpus annotated with traditional TAG derivation trees (using β-trees rather than γ-trees). Except in very rare cases, this entirely determines linear order. So in this experiment we use a tree model and the XTAG grammar, but no linear language model.[1]

- In the fourth experiment (**TM-XTAG-LM**), we use the system as described in Section 3. Specifically, 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, but using γ-trees rather than β-trees. 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.

The test corpus is a randomly chosen subset of 100 sentences from the Section 20 of WSJ. The dependency structures for the test sentences were obtained automatically from converting the Penn TreeBank phrase structure trees, in the same way as was done to create the training corpus. The average length of the test sentences is 16.7 words with a longest sentence being 24 words in length.

As in the case of machine translation, evaluation in generation is a complex issue. We use a metric suggested in the MT literature (Alshawi et al.1998) based on string edit distance between

[1] In fact, we use the linear language model in those rare cases when a β trees can be adjoined in more than one position.

the output of the generation system and the reference corpus string from the WSJ. This metric, *generation accuracy*, allows us to evaluate without human intervention, automatically and objectively. Clearly, the metric does not provide a complete assessment of the quality of a generator since often there is more than one "good" result, but we assume that the requirement is to model the corpus as closely as possible (as is the case in some, but not all, applications). We have also independently verified the metric by asking human subjects for subjective judgments; the judgments show significant correlation with the metrics (Bangalore and Rambow2000a).
Generation accuracy, shown in Equation (1), 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 except that it 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').

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

The average generation accuracy for the four experiments are tabulated in Table 1. As can be seen, the use of a tree model improves results over the baseline, but the use of a linear model also improves results if the XTAG grammar is used: the best results are obtained when the tree model, the XTAG grammar, and the linear model are used.

Tree Model	Generation Accuracy
Baseline	56.2%
TM-LM	66.8%
TM-XTAG	68.4%
TM-XTAG-LM	72.4%

Table 1: Performance results from the three tree models.

5. Featurization of Supertags

5.1. Features

As pointed out by (Candito1996) and (Xia et al.1998), a supertag is a composite representation of a few orthogonal linguistic dimensions such as the subcategorization (argument list) of the head (Subcat) and the way in which specific arguments are realized syntactically (Transformation). These dimensions can be represented as features that can potentially be assigned independently of one another. A featurized representation of supertags helps in a more fine-grained error analysis and may allow for better stochastic supertag assignment models. In this section, we will describe our attempt to represent supertags as features and some preliminary results of error analysis using featurized supertags.
Table 2 shows the list of features and their values used in representing the supertags. (The Modifiee features only are used if ADJ is T.) Although the set of features are directly based on those proposed in (Candito1996) and (Xia et al.1998), we have made a few additions, most notably, FRR2, SGP1 and SGP2. While FRR (for "Function Reassignment Rule") is used to represent changes in the valency of a supertag, FRR2 is used to represent the linear order variations of arguments in the supertag such as dative shift and particle shift. Note that FRR, FRR2, and Transformation are all orthogonal to each other. SGP features are used to represent strongly governed prepositions for supertags that use a preposition in the realization of an argument

Features		Possible Values
POS		10 different part-of-speech tags
Subcat		Different argument frames (eg. NP, NP_NP, NP_S ...)
Transformation	Type	Declarative, WH, Relative, Resumptive_Relative, Gerund, Imperative, Inversion
	Argument	NIL,0,1,2
FRR	Type	NIL,Ergative,Equative,Passive,Passive_by,Predicative
	Argument	NIL,1,2
Modifiee	Type	NIL,NP,S,VP,N,Ad,PP,A,D,AP,P,DetP,V
	Direction	NIL,left,right
FRR2		DativeShift, ParticleShift1,ParticleShift2
SGP1		Strongly Governed prepositions for objects
SGP2		Strongly Governed prepositions for indirect objects
ADJ		Flag to indicate adjunct status

Table 2: The set of features and their values used to represent the supertags.

(SGP1 for the direct object, SGP2 for the indirect object). The value of this feature is derived from PP complements present in the corpus.

5.2. Error Analysis

We replaced the supertags in the **TM-XTAG-LM** model with the featurized representation treated as a single string. Since the featurized representation is just a notational variant for supertags, we got the same performance figures. However, the feature representation allows us to analyze the errors with respect to each of the features. We see that the most error occur in the features ADJ, SUBCAT, and POS (with about equal frequency). Errors also occur in TRANS and FRR, but much less frequently, and even less frequently in the other features. A sample of the individual errors with frequency is shown in Table 3.

Correct	FERGUS Assigned	Number
ADJ=NIL	ADJ=T	134
ADJ=T	ADJ=NIL	49
SUBCAT=NP	SUBCAT=NIL	46
SUBCAT=NIL	SUBCAT=NP	30
POS=N	POS=D	16
TRANSARG=NIL	TRANSARG=0	16
POS=V	POS=N	14
POS=G	POS=NIL	14
FRR=NIL	FRR=Predicative	14
TRANS=decl	TRANS=REL	13

Table 3: List of most frequent individual errors by features

We are working on developing models that better predict each of the individual features using modeling techniques from the Machine Learning community such as Bayesian Nets. The use of "featurized" supertags also has the advantage that they allow us to use FERGUS even when the TAG grammar is much less complete than the English XTAG grammar.

6. Future Work

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.

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

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

Tilman Becker. 1998. Fully lexicalized head-driven syntactic generation. In *Proceedings of the 8th International Workshop on Natural Language Generation*, Niagara-on-the-Lake, Ontario.

Marie-Helene Candito. 1996. A Principle-Based Hierarchical Representation of LTAGs. In *Proceedings of COLING-96*, Copenhagen, Denmark.

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.

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 Aravind Joshi. 1996. A formal look at dependency grammars and phrase-structure grammars, with special consideration of word-order phenomena. In Leo Wanner, editor, *Current Issues in Meaning-Text Theory*. Pinter, London.

Owen Rambow, K. Vijay-Shanker, and David Weir. 1995. D-Tree Grammars. In *33rd Meeting of the Association for Computational Linguistics (ACL'95)*, pages 151–158. ACL.

Matthew Stone and Christine Doran. 1997. Sentence planning as description using tree adjoining grammar. In *35th Meeting of the Association for Computational Linguistics (ACL'97)*, pages 198–205, Madrid, Spain.

Fei Xia, Martha Palmer, K.Vijay-Shanker, and Joseph Rosenzweig. 1998. Consistent grammar development using partial-tree descriptions for lexicalized tree-adjoining grammars. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, number 98–12 in IRCS Report. Institute for Research in Cognitive Science, University of Pennsylvania.

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.