A Domain-Specific Statistical Surface Realizer

Jeffrey T. Russell
Center for the Study of Language and Information
Stanford University
jefe@stanford.edu

Abstract

We present a search-based approach to automatic surface realization given a corpus of domain sentences. Using heuristic search based on a statistical language model and a structure we introduce called an *inheritance table* we overgenerate a set of complete syntactic-semantic trees that are consistent with the given semantic structure and have high likelihood relative to the language model. These trees are then lexicalized, linearized, scored, and ranked. This model is being developed to generate real-time navigation instructions.

1 Introduction

The target application for this work is real-time, interactive navigation instructions. Good direction-givers respond actively to a driver's actions and questions, and express instructions relative to a large variety of landmarks, times, and distances. These traits require robust, real-time natural language generation. This can be broken into three steps: (1) generating a route plan, (2) reasoning about the route and the user to produce an abstract representation of individual instructions, and (3) realizing these instructions as sentences in natural language (in our case, English). We focus on the last of these steps: given a structure that represents the semantic content of a sentence, we want to produce an English sentence that expresses this content. According to the traditional division of content determination, sentence planning, and surface realization, our work is primarily concerned with surface realization, but also includes aspects of sentence planning. Our application requires robust flexibility within a restricted domain that is not well represented in the traditional corpora or tools. These requirements suggest using trainable stochastic generation.

A number of statistical surface realizers have been described, notably the FERGUS (Bangalore and Rambow, 2000) and HALogen systems (Langkilde-Geary, 2002), as well as experiments in (Ratnaparkhi, 2000). FERGUS (Flexible Empiricist/Rationalist Generation Using Syntax) takes as input a dependency tree whose nodes are marked with lexemes only. The generator automatically "supertags" each input node with a TAG tree, then produces a lattice of all possible linearizations consistent with the supertagged dependency tree. Finally it selects the most likely traversal of this lattice, conditioned on a domain-trained language model. The HALogen system is a broad-coverage generator that uses a combination of statistical and symbolic techniques. The input, a structure of feature-value pairs (see Section 3.1), is symbolically transformed into a forest of possible expressions, which are then ranked using a corpus-trained statistical language model. Ratnaparkhi also uses an overgeneration approach, using search to generate candidate sentences which are then scored and ranked. His paper outlines experiments with an n-gram model, a trained dependency grammar, and finally a hand-built grammar including content-driven conditions for applying rules. The last of these systems outperformed the n-gram and trained grammar in testing based on human judgments.

Proceedings of the ACL Student Research Workshop, pages 151–156,
Ann Arbor, Michigan, June 2005.

The basic idea of our system fits in the overgenerate-and-rank paradigm. Our approach is partly motivated by the idea of 'softening' Ratnaparkhi's third system, replacing the hand-built grammar rules with a combination of a trained statistical language model and a structure called an *inheritance table*, which captures long-run dependency information. This allows us to overgenerate based on rules that are sensitive to structured content without incurring the cost of designing such rules by hand.

2 Algorithm

We use dependency tree representations for both the semantics and syntax of a sentence; we introduce the *syntactic-semantic (SS) tree* to combine information from both of these structures. An SS tree is constructed by "attaching" some of the nodes of a sentence's semantic tree to the nodes of its syntactic tree, obeying two rules:

- **Each node in the semantic tree is attached to at most one node of the syntactic tree.**
- **Semantic and syntactic hierarchical orderings are consistent.** That is to say, if two semantic nodes x_1 and x_2 are attached to two syntactic nodes y_1 and y_2, respectively, then x_1 is a descendant of x_2 in the semantic tree if and only if y_1 is a descendant of y_2 in the syntactic tree.

The nodes of an SS tree are either unattached semantic or syntactic nodes, or else pairs of attached nodes. The SS tree's hierarchy is consistent with the hierarchies in the syntactic and semantic trees. We say that an SS tree T *satisfies* a semantic structure S if S is embedded in T. This serves as formalization of the idea of a sentence expressing a certain content.

2.1 Outline

The core of our method is a heuristic search of the space of possible SS trees. Our search goal is to find the N best complete SS trees that express the given semantic structure. We take 'best' here to be the trees which have the highest conditional likelihood given that they express the right semantic structure. If S is our semantic structure and LM is our statistical language model, we want to find syntactic trees T that maximize $P_{LM}(T|S)$.

In order to search the space of trees, we build up trees by expanding one node at a time. During the search, then, we deal with *incomplete trees*; that is, trees with some nodes not fully expanded. This means that we need a way to determine how promising an incomplete tree T is: i.e., how good the best complete trees are that can be built up by expanding T. As it turns out (Section 2.2), we can efficiently approximate the function [1] $P_{LM}(T|S)$ for an incomplete tree, and this function is a good heuristic for the maximum likelihood of a complete tree extended from T.

Here is an outline of the algorithm:

- Start with a root tree.
  - Take the top N trees and expand one node in each.
  - Score each expanded tree for $P_{LM}(T|S)$, and put in the search order accordingly.
  - Repeat until we find enough trees that satisfy S.
- Complete the trees.
- Linearize and lexicalize the trees.
- Rank the complete trees according to some scoring function.

2.2 Heuristic

Our search goal is to maximize $P_{LM}(T|S)$. (Henceforth we abbreviate P_{LM} as just P.) Ideally, then, we would at each step expand the incomplete tree that can be extended to the highest-likelihood complete tree, i.e. that has the highest value of $\max_{T'} P(T'|S)$ over all complete trees T' that extend T. We use the notation $T' > T$ when T' is a complete tree that extends an incomplete tree T, and the notation $T' \triangleright S$ when T' satisfies S. Then the "goodness" of a tree T is given by

$$\max_{T'>T} P(T'|S) = \max_{T'>T; T' \triangleright S} P(T')/P(S) \qquad (1)$$

[1] This probability is defined to be the sum of the probabilities $P_{LM}(T|T')P_{LM}(T'|S)$ for all complete trees T'

Since finding this maximum explicitly is not feasible, we use the heuristic $P(T|S)$. By Bayes' rule, $P(T|S) = P(S|T)P(T)/P(S)$, where $P(S)$ is a normalizing factor, $P(T)$ can be easily calculated using the language model (as the product of the probabilities of the node expansions that appear in T), and

$$P(S|T) = \sum_{T'} P(S|T')P(T'|T) = \sum_{T' \triangleright S} P(T'|T)$$

Since $P(T'|T) = P(T|T')P(T')/P(T)$, and since $P(T|T')$ is 1 if $T' > T$ and 0 otherwise, we have

$$P(T|S) = \frac{1}{P(S)} \sum_{T' \triangleright S} P(T|T')P(T')$$

$$= \frac{1}{P(S)} \sum_{T' > T; T' \triangleright S} P(T')$$

Together with Equation 1 this shows that $P(T|S) \geq \max_{T' > T} P(T'|S)$, since the maximum is one of the terms in the sum. This fact is analogous to showing that $P(T|S)$ is an admissible heuristic (in the sense of A* search).

We can see how to calculate $P(T|S)$ in practice by decomposing the structure of a tree T' such that $T' > T$ and $T' \triangleright S$. Since T' extends T, the top of T' is identical to T. The semantic tree S will have some of its nodes in T, and some in the part of T' that extends beyond T. Let $\alpha(S, T)$ be the set containing the highest nodes in S that are not in T. Each node $s \in \alpha(S, T)$ is the root node of a subtree in T'. Each of these subtrees can be considered separately.

First we consider how the these subtrees are joined to the nodes in T. The condition of consistent ordering requires that each node in $\alpha(S, T)$ be a descendant in T' of its parent in S, and moreover it should not be a descendant of any of its siblings in S. Let sib be a set of siblings in $\alpha(S, T)$, and let p be their semantic parent. Then p is the root node of a subtree of T, called T_p. We will designate the *T-set* of sib as the set of leaves of T_p that are not descended from any nodes in S below p–in particular, that are not descended from any other siblings of the nodes in sib. Then in T' all of the nodes in sib must descend from the T-set of sib. In other words, there is a set of subtrees of T' which are rooted at the nodes in the T-set of sib, and all of the nodes in sib appear in these subtrees such that none of them are descended from each other.

This analysis sets us up to rewrite $P(T|S)$ in terms of sums over these various subtrees. We use the notation $P(\{x_1, ..., x_k\} \rightarrow \{y_1, ..., y_l\})$ to denote the probability that the nodes $y_1, ..., y_l$ eventually descend from $x_1, ..., x_k$ without dominating each other; this probability is the sum of $P(T_1, ..., T_k)$ over all sets of trees $T_1 > x_1, ..., T_k > x_k$ such that each node $y_1, ..., y_l$ appears in some T_i and no y_i descends from any y_j. Then we can rewrite $P(T|S)$ as

$$\frac{P(T)}{P(S)} \prod_{sib} P(\text{T-set}(sib) \rightarrow sib) \prod_{x \in \alpha(S,T)} P(x \rightarrow S_x) \quad (2)$$

S_x denotes the subtree of S whose root node is x. $P(x \rightarrow S_x)$ is 1 if S_x contains only the node x, and otherwise is

$$P(x \rightarrow \text{children}_S(x)) \left(\prod_{y \in \text{children}_S(x)} P(y \rightarrow S_y) \right)$$

Rather than calculating the value of formula 2 exactly, we now introduce an approximation to our heuristic function. For sets X, Y, we approximate $P(X \rightarrow Y)$ with $\prod_{y \in Y} P(X \rightarrow y)$. This amounts to two simplifications: first, we drop the restriction that no node be descended from its semantic sibling; second, we assume that the probabilities of each node descending from X are independent from one another.

$P(X \rightarrow y)$ is the probability that at least one $x \in X$ has y as a descendant, i.e. $P(X \rightarrow y) = \text{AL1}_{x \in X} P(x \rightarrow y)$, where AL1 is the 'At-least-one' function.[2] This means that we can approximate $P(T|S)$ as

$$\frac{P(T)}{P(S)} \prod_{y \in \alpha(S,T)} \text{AL1}_{x \in \text{T-set}(y)} P(x \rightarrow y) P(y \rightarrow S_y) \quad (3)$$

[2]That is, given the probabilities of a set of events, the At-least-one function gives the probability of at least one of the events occuring. For independent events, $\text{AL1}\{\} = 0$ and $\text{AL1}\{p_1, ..., p_n\} = p_n + (1 - p_n)\text{AL1}\{p_1, ..., p_{n-1}\}$.

The calculation of $P(T|S)$ has been reduced to finding $P(x \rightarrow y)$ for individual nodes. These values are retrieved from the inheritance table, described below.

Note that when we expand a single node of an incomplete tree, only a few factors in Equation 3 change. Rather than recalculating each tree's score from scratch, then, by caching intermediate results we can recompute only the terms that change. This allows for efficient calculation of the heuristic function.

2.3 Inheritance Table

The inheritance table (IT) allows us to predict the potential descendants of an incomplete tree. For each pair of SS nodes x and y, the IT stores $P(x \rightarrow y)$, the probability that y will eventually appear as a descendant of x. The IT is precomputed once from the language model; the same IT is used for all queries.

We can compute the IT using an iterative process. Consider the transformation $\mathbf{T}$ that takes a distribution $Q(x \rightarrow y)$ to a new distribution $\mathbf{T}(Q)$ such that $\mathbf{T}(Q)(x \rightarrow y)$ is equal to 1 when $x = y$, and otherwise is equal to

$$\sum_{\zeta \in Exp(x)} P_{LM}(\zeta|x) \mathrm{AL1}_{z \in \zeta} Q(z \rightarrow y) \qquad (4)$$

Here $Exp(x)$ is the set of possible expansions of x, and $P_{LM}(\zeta|x)$ is the probability of the expansion ζ according to the language model.

The defining property of the IT's distribution P is that $\mathbf{T}(P) = P$. We can use this property to compute the table iteratively. Begin by setting $P_0(x \rightarrow y)$ to 1 when $x = y$ and 0 otherwise. Then at each step let $P_{k+1} = \mathbf{T}(P_k)$. When this process converges, the limiting function is the correct inheritance distribution.

2.4 Completing Trees

A final important issue is termination. Ordinarily, it would be sensible to remove a tree from the search order only when it is a goal state—that is, if it is a complete tree that satisfies S. However, this turns out to be not the best approach in this case due to a quirk of our heuristic. $P(T|S)$ has two non-constant factors, $P(S|T)$ and $P(T)$. Once all of the nodes in S appear in an incomplete tree T, $P(S|T) = 1$, and so it won't increase as the tree is expanded further. Moreover, with each node expanded, $P(T)$ decreases. This means that we are unlikely to make progress beyond the point where all of the semantic content appears in a tree.

An effective way to deal with this is to remove trees from the search order as soon as $P(S|T)$ reaches 1. When the search terminates by finding enough of these 'almost complete' trees, these trees are completed: we find the optimal complete trees by repeatedly expanding the N most likely almost-complete trees (ranked by $P(T)$) until sufficiently many complete trees are found.

3 Implementation

3.1 Representation

Our semantic representation is based on the HALogen input structure (Langkilde-Geary, 2002). The meaning of a sentence is represented by a tree whose nodes are each marked with a concept and a semantic role. For example, the meaning of the sentence "Turn left at the second traffic light" is represented by the following structure:

```
(maketurn
   :direction (left)
   :spatial-locating
     (trafficlight
         :modifier (second)))
```

The syntax model we use is statistical dependency grammar. As we outlined in Section 2, the semantic and syntactic structures are attached to one another in an SS tree. In order to accomodate the requirement that each semantic node is attached to no more than one syntactic node, collocations like "traffic light" or "John Hancock Tower", are treated as single syntactic nodes. It can also be convenient to extend this idea, treating phrases like "turn around" or "thank you very much" as atomic. In the case where a concept attaches to multi-word expression, but where it is inconvenient to treat the expression as a syntactic atom, we adopt the convention of attaching the concept to the hierarchically dominant word in the expression. For instance, the concept of turning can be attached to the expression "make a

turn"; in this case we attach the concept to the word "make", and not to "turn".

The nodes of an SS tree are (word, part of speech, concept, semantic role) 4-tuples, where the concept and role are left empty for function words, and the word and part of speech are left empty for concepts with no direct syntactic correlate. Generally we omit the word itself from the tree in order to mitigate sparsity issues; these are added to the final full tree by a lexical choice module.

We use a domain-trained language model based on the same dependency structure as our syntactic-semantic representations. The currently implemented model calculates the probability of expansions given a parent node based on an explicit tabular representation of the distribution $P(\zeta|x)$ for each x. This language model is also used to score and rank generated sentences.

3.2 Corpus and Annotation

Training this language model requires an annotated corpus of in-domain text. Our main corpus comes from transcripts of direction-giving in a simulation context, collected using the "Wizard of Oz" set-up described in (Cheng et al., 2004). For development and testing, we extracted approximately 600 instructions, divided into training and test sets. The training set was used to train the language model used for search, the lexical choice module, and the scoring function. Both sets both underwent four partially-automated stages of annotation.

First we tag words with their part of speech, using the Brill tagger with manually modified lexicon and transformation rules for our domain (Brill, 1995). Second, the words are disambiguated and assigned a concept tag. For this we construct a domain ontology, which is used to automatically tag the unambiguous words and prompt for human disambiguation in the remaining cases. The third step is to assign semantic roles. This is accomplished by using a list of contextual rules, similar to the rules used by the Brill tagger. For example, the rule

> CON intersection PREV1OR2OR3WD at
> : spatial-locating

assigns the role "spatial-locating" to a word whose concept is "intersection" if the word "at" appears one, two, or three words before it. A segment of the corpus was automatically annotated using such rules, then a human annotater made corrections and added new rules, repeating these steps until the corpus was fully annotated with semantic roles.

After the first three stages, the sentence, "Turn left at the next intersection" is annotated as follows:

```
turn/VB/maketurn left/RB/
$leftright/direction at/IN the/
DT next/JJ/first/modifier
intersection/NN/intersection/
spatial-locating
```

The final annotation step is parsing. For this we use an approach similar to Pereira and Schabes' grammar induction from partially bracketed text (Pereira and Schabes, 1992). First we annotate a segment of the corpus. Then we use the *inside-outside* algorithm to simultaneously train a dependency grammar and complete the annotation. We then manually correct a further segment of the annotation, and repeat until acceptable parses are obtained.

3.3 Rendering

Linearizing an SS tree amounts to deciding the order of the branches and whether each appears on the left or the right side of the head. We built this information into our language model, so a grammar rule for expanding a node includes full ordering information. This makes the linearization step trivial at the cost of adding sparsity to the language model.

Lexicalization could be relegated to the language model in the same way, by including lexemes in the representation of each node, but again this would incur sparsity costs. The other option is to delegate lexical choice to a separate module, which takes a SS tree and assigns a word to each node. We use a hybrid approach: content words are assigned using a lexical choice module, while most function words are included explicitly in the language model. The current lexical choice module simply assigns each unlabeled node the most likely word conditioned on its (POS, concept, role) triple, as observed in the training corpus.

4 Example

We take the semantic structure presented in Section 3.1 as an example generation query. The search

stage terminates when 100 trees that embed this semantic structure have been found. The best-scoring sentence has the following lexicalized tree:

```
turn/VB/maketurn
  +left/RB/$leftright/direction
  +at/IN
     +traffic_light/NN/
      trafficlight/
      spatial-locating
        -the/DT
        +next/JJ/first/modifier
```

This is finally rendered thus:

turn left at the second traffic_light.

5 Preliminary Results

For initial testing, we separated the annotated corpus into a 565-sentence training set and a 57-sentence test set. We automatically extracted semantic structures from the test set, then used these structures as generation queries, returning only the highest-ranked sentence for each query. The generated results were then evaluated by three independent human annotaters along two dimensions: (1) Is the generated sentence grammatical? (2) Does the generated sentence have the same meaning as the original sentence?

For 11 of the 57 sentences (19%), the query extraction failed due to inadequate grammar coverage.[3] Of the 46 instances where a query was successfully extracted, 3 queries (7%) timed out without producing output. Averaging the annotaters' judgments, 1 generated sentence (2%) was ungrammatical, and 3 generated sentences (7%) had different meanings from their originals. 39 queries (85%) produced output that was both grammatical and faithful to the original sentence's meaning.

6 Future Work

Statistically-driven search offers a means of efficiently overgenerating sentences to express a given semantic structure. This is well-suited not only to our navigation domain, but also to other domains with a relatively small vocabulary but variable and complex content structure. Our implementation of the idea of this paper is under development in a number of directions.

A better option for **robust language modeling** is to use maximum entropy techniques to train a feature-based model. For instance, we can determine the probability of each child using such features as the POS, concept, and role of the parent and previous siblings. It may also be more effective to isolate **linear precedence** from the language model, introducing a non-trivial linearization step. Similarly, the **lexicalization module** can be improved on by using a more context-sensitive model.

Using only a tree-based **scoring function** is likely to produce inferior results to one that incorporates a linear score. A weighted average of the dependency score with an n-gram model would already offer improvement. To further improve fluency, these could also be combined with a scoring function that takes longer-range dependencies into account, as well as penalizing extraneous content.

[3]The corpus was partially annotated for parse data, the full parses being automatically generated from the domain-trained language model. It was at this step that query extraction sometimes failed.

References

Srinivas Bangalore and O. Rambow. 2000. Using TAG, a Tree Model, and a Language Model for Generation. *5th Int'l Workshop on Tree-Adjoining Grammars (TAG+)*, TALANA, Paris.

Eric Brill. 1995. Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics*, 21 (4).

Hua Cheng, H. Bratt, R. Mishra, E. Shriberg, S. Upson, J. Chen, F. Weng, S. Peters, L. Cavedon and J. Niekrasz. 2004. A Wizard Of OZ Framework for Collecting Spoken Human-Computer Dialogs. *Proc. 8th ICSLP*, Jeju Island, Korea.

Irene Langkilde-Geary. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. *Proc. 2nd INLG*, Harriman, NY.

Fernando Pereira and Y. Schabes. 1992. Inside-outside reestimation from partially bracketed corpora. *Proc. 30th ACL*, p.128-135, Newark.

Adwait Ratnaparkhi. 2000. Trainable methods for surface natural language generation. *Proc. 1st NAACL*, Seattle.