Trainable Methods for Surface Natural Language Generation

Adwait Ratnaparkhi
IBM TJ Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598
`aratnapa@us.ibm.com`

Abstract

We present three systems for surface natural language generation that are trainable from annotated corpora. The first two systems, called NLG1 and NLG2, require a corpus marked only with domain-specific semantic attributes, while the last system, called NLG3, requires a corpus marked with both semantic attributes and syntactic dependency information. All systems attempt to produce a grammatical natural language phrase from a domain-specific semantic representation. NLG1 serves a baseline system and uses phrase frequencies to generate a whole phrase in one step, while NLG2 and NLG3 use maximum entropy probability models to individually generate each word in the phrase. The systems NLG2 and NLG3 learn to determine both the word choice and the word order of the phrase. We present experiments in which we generate phrases to describe flights in the air travel domain.

1 Introduction

This paper presents three trainable systems for surface natural language generation (NLG). Surface NLG, for our purposes, consists of generating a grammatical natural language phrase that expresses the meaning of an input semantic representation. The systems take a "corpus-based" or "machine-learning" approach to surface NLG, and learn to generate phrases from semantic input by statistically analyzing examples of phrases and their corresponding semantic representations. The determination of the content in the semantic representation, or "deep" generation, is not discussed here. Instead, the systems assume that the input semantic representation is fixed and only deal with how to express it in natural language.

This paper discusses previous approaches to surface NLG, and introduces three trainable systems for surface NLG, called NLG1, NLG2, and NLG3. Quantitative evaluation of experiments in the air travel domain will also be discussed.

2 Previous Approaches

Templates are the easiest way to implement surface NLG. A template for describing a flight noun phrase in the air travel domain might be `flight departing from $city-fr at $time-dep and arriving in $city-to at $time-arr` where the words starting with "$" are actually variables — representing the departure city, and departure time, the arrival city, and the arrival time, respectively— whose values will be extracted from the environment in which the template is used. The approach of writing individual templates is convenient, but may not scale to complex domains in which hundreds or thousands of templates would be necessary, and may have shortcomings in maintainability and text quality (e.g., see (Reiter, 1995) for a discussion).

There are more sophisticated surface generation packages, such as FUF/SURGE (Elhadad and Robin, 1996), KPML (Bateman, 1996), MUMBLE (Meteer et al., 1987), and RealPro (Lavoie and Rambow, 1997), which produce natural language text from an abstract semantic representation. These packages require linguistic sophistication in order to write the abstract semantic representation, but they are flexible because minor changes to the input can accomplish major changes to the generated text.

The only trainable approaches (known to the author) to surface generation are the purely statistical machine translation (MT) systems such as (Berger et al., 1996) and the corpus-based generation system described in (Langkilde and Knight, 1998). The MT systems of (Berger et al., 1996) learn to generate text in the target language straight from the source language, without the aid of an explicit semantic representation. In contrast, (Langkilde and Knight, 1998) uses corpus-derived statistical knowledge to rank plausible hypotheses from a grammar-based surface generation component.

3 Trainable Surface NLG

In trainable surface NLG, the goal is to learn the mapping from semantics to words that would otherwise need to be specified in a grammar or knowledge base. All systems in this paper use *attribute-value* pairs as a semantic representation, which suffice as a representation for a limited domain like air travel. For example, the set of attribute-value pairs { \$city-fr = New York City, \$city-to = Seattle , \$time-dep = 6 a.m., \$date-dep = Wednesday } represent the meaning of the noun phrase "a flight to Seattle that departs from New York City at 6 a.m. on Wednesday". The goal, more specifically, is then to learn the optimal *attribute ordering* and *lexical choice* for the text to be generated from the attribute-value pairs. For example, the NLG system should automatically decide if the attribute ordering in "flights to New York in the evening" is better or worse than the ordering in "flights in the evening to New York". Furthermore, it should automatically decide if the lexical choice in "flights departing to New York" is better or worse than the choice in "flights leaving to New York". The motivation for a trainable surface generator is to solve the above two problems in a way that reflects the observed usage of language in a corpus, but without the manual effort needed to construct a grammar or knowledge base.

All the trainable NLG systems in this paper assume the existence of a large corpus of phrases in which the values of interest have been replaced with their corresponding attributes, or in other words, a corpus of *generation templates*. Figure 1 shows a sample of training data, where only words marked with a "\$" are attributes. All of the NLG systems in this paper work in two steps as shown in Table 2. The systems NLG1, NLG2 and NLG3 all implement step 1; they produce a sequence of words intermixed with attributes, i.e., a template, from the the attributes alone. The values are ignored until step 2, when they replace their corresponding attributes in the phrase produced by step 1.

3.1 NLG1: the baseline

The surface generation model NLG1 simply chooses the most frequent template in the training data that corresponds to a given set of attributes. Its performance is intended to serve as a baseline result to the more sophisticated models discussed later. Specifically, $nlg_1(A)$ returns the phrase that corresponds to the attribute set A:

$$nlg_1(A) = \begin{cases} \text{argmax}_{phrase \in T_A} C(phrase, A) & T_A \neq \emptyset \\ \text{[empty string]} & T_A = \emptyset \end{cases}$$

where T_A are the phrases that have occurred with A in the training data, and where $C(phrase, A)$ is the training data frequency of the natural language phrase *phrase* and the set of attributes A. NLG1 will fail to generate anything if A is a novel combination of attributes.

3.2 NLG2: n-gram model

The surface generation system NLG2 assumes that the best choice to express any given attribute-value set is the word sequence with the highest probability that mentions all of the input attributes exactly once. When generating a word, it uses local information, captured by word n-grams, together with certain non-local information, namely, the subset of the original attributes that remain to be generated. The local and non-local information is integrated with use of features in a maximum entropy probability model, and a highly pruned search procedure attempts to find the best scoring word sequence according to the model.

3.2.1 Probability Model

The probability model in NLG2 is a conditional distribution over $V \cup *\mathtt{stop}*$, where V is the generation vocabulary and where $*\mathtt{stop}*$ is a special "stop" symbol. The generation vocabulary V consists of all the words seen in the training data. The form of the maximum entropy probability model is identical to the one used in (Berger et al., 1996; Ratnaparkhi, 1998):

$$p(w_i|w_{i-1}, w_{i-2}, attr_i) = \frac{\prod_{j=1}^{k} \alpha_j^{f_j(w_i, w_{i-1}, w_{i-2}, attr_i)}}{Z(w_{i-1}, w_{i-2}, attr_i)}$$

$$Z(w_{i-1}, w_{i-2}, attr_i) = \sum_{w'} \prod_{j=1}^{k} \alpha_j^{f_j(w', w_{i-1}, w_{i-2}, attr_i)}$$

where w_i ranges over $V \cup *\mathtt{stop}*$ and $\{w_{i-1}, w_{i-2}, attr_i\}$ is the history, where w_i denotes the ith word in the phrase, and $attr_i$ denotes the attributes that remain to be generated at position i in the phrase. The f_j, where $f_j(a,b) \in \{0,1\}$, are called *features* and capture any information in the history that might be useful for estimating $p(w_i|w_{i-1}, w_{i-2}, attr_i)$. The features used in NLG2 are described in the next section, and the feature weights α_j, obtained from the Improved Iterative Scaling algorithm (Berger et al., 1996), are set to maximize the likelihood of the training data. The probability of the sequence $W = w_1 \ldots w_n$, given the attribute set A, (and also given that its length is n) is:

$$Pr(W = w_1 \ldots w_n | len(W) = n, A) = \prod_{i=1}^{n} p(w_i|w_{i-1}, w_{i-2}, attr_i)$$

3.2.2 Feature Selection

The feature patterns, used in NLG2 are shown in Table 3. The actual features are created by matching the patterns over the training data, e.g., an actual feature derived from the word bi-gram template might be:

$$f(w_i, w_{i-1}, w_{i-2}, attr_i) = \begin{cases} 1 & \text{if } w_i = \mathtt{from} \\ & \text{and } w_{i-1} = \mathtt{flight} \\ & \text{and } \mathtt{\$city-fr} \in attr_i \\ 0 & \text{otherwise} \end{cases}$$

```
flights on $air from $city-fr to $city-to the $time-depint of $date-dep
$trip flights on $air from $city-fr to $city-to leaving after $time-depaft on $date-dep
flights leaving from $city-fr going to $city-to after $time-depaft on $date-dep
flights leaving from $city-fr to $city-to the $time-depint of $date-dep
$air flight $fltnum from $city-fr to $city-to on $date-dep
$city-fr to $city-to $air flight $fltnum on the $date-dep
$trip flights from $city-fr to $city-to
```

Table 1: Sample training data

Input to Step 1:	{ \$city-fr, \$city-to, \$time-dep, \$date-dep }
Output of Step 1:	"a flight to \$city-to that departs from \$city-fr at \$time-dep on \$date-dep"
Input to Step 2:	"a flight to \$city-to that departs from \$city-fr at \$time-dep on \$date-dep", { \$city-fr = New York City, \$city-to = Seattle , \$time-dep = 6 a.m., \$date-dep = Wednesday }
Output of Step 2:	"a flight to Seattle that departs from New York City at 6 a.m. on Wednesday"

Table 2: Two steps of NLG process

Low frequency features involving word n−grams tend to be unreliable; the NLG2 system therefore only uses features which occur K times or more in the training data.

3.2.3 Search Procedure

The search procedure attempts to find a word sequence $w_1 \dots w_n$ of any length $n \leq M$ for the input attribute set A such that

1. w_n is the stop symbol *stop*
2. All of the attributes in A are mentioned at least once
3. All of the attributes in A are mentioned at most once

and where M is an heuristically set maximum phrase length.

The search is similar to a left-to-right breadth-first-search, except that only a fraction of the word sequences are considered. More specifically, the search procedure implements the recurrence:

$$\begin{aligned} W_{N,1} &= top(N, \{w | w \in V\}) \\ W_{N,i+1} &= top(N, next(W_{N,i})) \end{aligned}$$

The set $W_{N,i}$ is the top N scoring sequences of length i, and the expression $next(W_{N,i})$ returns all sequences $w_1 \dots w_{i+1}$ such that $w_1 \dots w_i \in W_{N,i}$, and $w_{i+1} \in V \cup$ *stop*. The expression $top(N, next(W_{N,i}))$ finds the top N sequences in $next(W_{N,i})$. During the search, any sequence that ends with *stop* is removed and placed in the set of completed sequences. If N completed hypotheses are discovered, or if $W_{N,M}$ is computed, the search terminates. Any incomplete sequence which does not satisfy condition (3) is discarded and any complete sequence that does not satisfy condition (2) is also discarded.

When the search terminates, there will be at most N completed sequences, of possibly differing lengths. Currently, there is no normalization for different lengths, i.e., all sequences of length $n \leq M$ are equiprobable:

$$\begin{aligned} Pr(len(W) = n) &= \tfrac{1}{M} \quad n \leq M \\ &= 0 \quad n > M \end{aligned}$$

NLG2 chooses the best answer to express the attribute set A as follows:

$$nlg_2(A) = \operatorname{argmax}_{W \in W_{nlg2}} Pr(len(W) = n) \cdot Pr(W | len(W) = n, A)$$

where W_{nlg2} are the completed word sequences that satisfy the conditions of the NLG2 search described above.

3.3 NLG3: dependency information

NLG3 addresses a shortcoming of NLG2, namely that the previous two words are not necessarily the best informants when predicting the next word. Instead, NLG3 assumes that conditioning on *syntactically related* words in the history will result on more accurate surface generation. The search procedure in NLG3 generates a syntactic dependency tree from

Description	Feature $f(w_i, w_{i-1}, w_{i-2}, attr_i) = \dots$
No Attributes remaining	1 if $w_i =?$ and $attr_i = \{\}$, 0 otherwise
Word bi-gram with attribute	1 if $w_i =?$ and $w_{i-1} =?$ and $? \in attr_i$, 0 otherwise
Word tri-gram with attribute	1 if $w_i =?$ and $w_{i-1}w_{i-2} =??$ and $? \in attr_i$, 0 otherwise

Table 3: Features patterns for NLG2. Any occurrence of "?" will be instantiated with an actual value from training data.

top-to-bottom instead of a word sequence from left-to-right, where each word is predicted in the context of its syntactically related parent, grandparent, and siblings. NLG3 requires a corpus that has been annotated with tree structure like the sample dependency tree shown in Figure 1.

3.3.1 Probability Model

The probability model for NLG3, shown in Figure 2, conditions on the parent, the two closest siblings, the direction of the child relative to the parent, and the attributes that remain to be generated.

Just as in NLG2, p is a distribution over $V \cup$ `*stop*`, and the Improved Iterative Scaling algorithm is used to find the feature weights α_j. The expression $ch_i(w)$ denotes the ith closest child to the headword w, $par(w)$ denotes the parent of the headword w, $dir \in \{\texttt{left}, \texttt{right}\}$ denotes the direction of the child relative to the parent, and $attr_{w,i}$ denotes the attributes that remain to be generated in the tree when headword w is predicting its ith child. For example, in Figure 1, if $w =$"flights", then $ch_1(w) =$"evening" when generating the left children, and $ch_1(w) =$"from" when generating the right children. As shown in Figure 3, the probability of a dependency tree that expresses an attribute set A can be found by computing, for each word in the tree, the probability of generating its left children and then its right children.[1] In this formulation, the left children are generated independently from the right children. As in NLG2, NLG3 assumes the uniform distribution for the length probabilities $Pr(\#\text{ of left children} = n)$ and $Pr(\#\text{ of right children} = n)$ up to a certain maximum length $M' = 10$.

3.3.2 Feature Selection

The feature patterns for NLG3 are shown in Table 4. As before, the actual features are created by matching the patterns over the training data. The features in NLG3 have access to syntactic information whereas the features in NLG2 do not. Low frequency features involving word $n-$grams tend to be unreliable; the NLG3 system therefore only uses features which occur K times or more in the training data. Furthermore, if a feature derived from Table 4 looks at a particular word $ch_i(w)$ and attribute a, we only allow it if a has occurred as a descendent of $ch_i(w)$ in some dependency tree in the training set. As an example, this condition allows features that look at $ch_i(w) =$"to" and \$city-to$\in attr_{w,i}$ but disallows features that look at $ch_i(w) =$"to" and \$city-fr$\in attr_{w,i}$.

[1]We use a dummy ROOT node to generate the top most head word of the phrase

3.4 Search Procedure

The idea behind the search procedure for NLG3 is similar to the search procedure for NLG2, namely, to explore only a fraction of the possible trees by continually sorting and advancing only the top N trees at any given point. However, the dependency trees are not built left-to-right like the word sequences in NLG2; instead they are built from the current head (which is initially the root node) in the following order:

1. Predict the next left child (call it x_l)
2. If it is `*stop*`, jump to (4)
3. Recursively predict children of x_l. Resume from (1)
4. Predict the next right child (call it x_r)
5. If it is `*stop*`, we are done predicting children for the current head
6. Recursively predict children of x_r. Resume from (4)

As before, any incomplete trees that have generated a particular attribute twice, as well as completed trees that have not generated a necessary attribute are discarded by the search. The search terminates when either N complete trees or N trees of the maximum length M are discovered. NLG3 chooses the best answer to express the attribute set A as follows:

$$nlg_3(A) = \operatorname*{argmax}_{T \in T_{nlg3}} Pr(T|A)$$

where T_{nlg3} are the completed dependency trees that satisfy the conditions of the NLG3 search described above.

4 Experiments

The training and test sets used to evaluate NLG1, NLG2 and NLG3 were derived semi-automatically from a pre-existing annotated corpus of user queries in the air travel domain. The annotation scheme used a total of 26 attributes to represent flights.

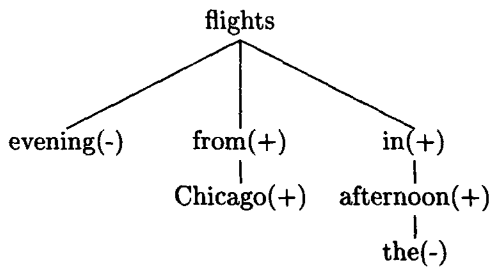

Figure 1: Sample dependency tree for the phrase *evening flights from Chicago in the afternoon.* - and + signs indicate left or right child, respectively.

$$\begin{aligned} p(ch_i(w)|w, ch_{i-1}(w), ch_{i-2}(w), par(w), dir, attr_{w,i}) &= \frac{\prod_{j=1}^{k} \alpha_j^{f_j(ch_i(w), w, ch_{i-1}(w), ch_{i-2}(w), par(w), dir, attr_{w,i})}}{Z(w, ch_{i-1}(w), ch_{i-2}(w), par(w), dir, attr_{w,i})} \\ Z(w, ch_{i-1}(w), ch_{i-2}(w), par(w), dir, attr_{w,i}) &= \sum_{w'} \prod_{j=1}^{k} \alpha_j^{f_j(w', w, ch_{i-1}(w), ch_{i-2}(w), par(w), dir, attr_{w,i})} \end{aligned}$$

Figure 2: NLG3: Equations for the probability of the ith child of head word w, or $ch_i(w)$

$$\begin{aligned} Pr(T|A) &= \prod_{w \in T} Pr_{left}(w|A) Pr_{right}(w|A) \\ Pr_{left}(w|A) &= Pr(\#\text{ of left children} = n) \prod_{i=1}^{n} p(ch_i(w)|w, ch_{i-1}(w), ch_{i-2}(w), par(w), dir = \texttt{left}, attr_{w,i}) \\ Pr_{right}(w|A) &= Pr(\#\text{ of right children} = n) \prod_{i=1}^{n} p(ch_i(w)|w, ch_{i-1}(w), ch_{i-2}(w), par(w), dir = \texttt{right}, attr_{w,i}) \end{aligned}$$

Figure 3: NLG3: Equations for the probability of a dependency tree T

Description	Feature $f(ch_i(w), w, ch_{i-1}(w), ch_{i-2}(w), par(w), dir, attr_{w,i}) = \ldots$
Siblings	1 if $ch_i(w)$ =? and $ch_{i-1}(w)$ =? and $ch_{i-2}(w)$ =? and dir =? and ? $\in attr_{w,i}$, 0 otherwise
Parent + sibling	1 if $ch_i(w)$ =? and $ch_{i-1}(w)$ =? and w =? and dir =? and ? $\in attr_{w,i}$, 0 otherwise
Parent + grandparent	1 if $ch_i(w)$ =? and w =? and $par(w)$ =? and dir =? and ? $\in attr_{w,i}$, 0 otherwise

Table 4: Features patterns for NLG3. Any occurrence of "?" will be instantiated with an actual value from training data.

System	Parameters	% Correct	% OK	% Bad	% No output	% error reduction from NLG1
NLG1	-	84.9	4.9	7.2	3.0	-
NLG2	N=10,M=30,K=3	88.2	4.7	6.4	0.7	22
NLG3	N=5,M=30,K=10	89.9	4.4	5.5	0.2	33

Table 5: Weighted evaluation of trainable surface generation systems by judge A

System	Parameters	% Correct	% OK	% Bad	% No output	% error reduction from NLG1
NLG1	-	81.6	8.4	7.0	3.0	-
NLG2	N=10,M=30,K=3	86.3	5.8	7.2	0.7	26
NLG3	N=5,M=30,K=10	88.4	4.0	7.4	0.2	37

Table 6: Weighted evaluation of trainable surface generation systems by judge B

System	Parameters	% Correct	% OK	% Bad	% No output	% error reduction from NLG1
NLG1	-	48.4	6.8	24.2	20.5	-
NLG2	N=10,M=30,K=3	64.7	12.1	22.6	0.5	32
NLG3	N=5,M=30,K=10	63.1	11.6	23.7	1.6	29

Table 7: Unweighted evaluation of trainable surface generation systems by judge A

System	Parameters	% Correct	% OK	% Bad	% No output	% error reduction from NLG1
NLG1	-	41.1	8.9	29.5	20.5	-
NLG2	N=10,M=30,K=3	62.1	13.7	23.7	0.5	36
NLG3	N=5,M=30,K=10	65.3	11.1	22.1	1.6	41

Table 8: Unweighted evaluation of trainable surface generation systems by judge B

The training set consisted of 6000 templates describing flights while the test set consisted of 1946 templates describing flights. All systems used the same training set, and were tested on the attribute sets extracted from the phrases in the test set. For example, if the test set contains the template "flights to $city-to leaving at $time-dep", the surface generation systems will be told to generate a phrase for the attribute set { $city-to, $time-dep }. The output of NLG3 on the attribute set { $city-to, $city-fr, $time-dep } is shown in Table 9.

There does not appear to be an objective *automatic* evaluation method[2] for generated text that correlates with how an actual person might judge the output. Therefore, two judges — the author and a colleague — manually evaluated the output of all three systems. Each judge assigned each phrase from each of the three systems one of the following rankings:

Correct: Perfectly acceptable

OK: Tense or agreement is wrong, but word choice is correct. (These errors could be corrected by post-processing with a morphological analyzer.)

Bad: Words are missing or extraneous words are present

No Output: The system failed to produce any output

While there were a total 1946 attribute sets from the test examples, the judges only needed to evaluate the 190 *unique* attribute sets, e.g., the attribute set { $city-fr $city-to } occurs 741 times in the test data. Subjective evaluation of generation output is not ideal, but is arguably superior than an automatic evaluation that fails to correlate with human linguistic judgement.

The results of the manual evaluation, as well as the values of the search and feature selection parameters for all systems, are shown in Tables 5, 6, 7, and 8. (The values for N, M, and K were determined by manually evaluating the output of the 4 or 5 most common attribute sets in the training data). The *weighted* results in Tables 5 and 6 account for multiple occurrences of attribute sets, whereas the *unweighted* results in Tables 7 and 8 count each unique attribute set once, i.e., { $city-fr $city-to } is counted 741 times in the weighted results but once in the unweighted results. Using the weighted results, which represent testing conditions more realistically than the unweighted results, both judges found an improvement from NLG1 to NLG2, and from NLG2 to NLG3. NLG3 cuts the error rate from NLG1 by at least 33% (counting anything without a rank of **Correct** as wrong). NLG2 cuts the error rate by at least 22% and underperforms NLG3, but requires far less annotation in its training data. NLG1 has no chance of generating anything for 3% of the data — it fails completely on novel attribute sets. Using the unweighted results, both judges found an improvement from NLG1 to NLG2, but, surprisingly, judge A found a slight decrease while judge B found an increase in accuracy from NLG2 to NLG3. The unweighted results show that the baseline NLG1 does well on the common attribute sets, since it correctly generates only less than 50% of the unweighted cases but over 80% of the weighted cases.

5 Discussion

The NLG2 and NLG3 systems automatically attempt to generalize from the knowledge inherent in the training corpus of templates, so that they can generate templates for novel attribute sets. There

[2]Measuring word overlap or edit distance between the system's output and a "reference" set would be an automatic scoring method. We believe that such a method does not accurately measure the correctness or grammaticality of the text.

Probability	Generated Text
0.107582	$time-dep flights from $city-fr to $city-to
0.00822441	$time-dep flights between $city-fr and $city-to
0.00564712	$time-dep flights $city-fr to $city-to
0.00343372	flights from $city-fr to $city-to at $time-dep
0.0012465	$time-dep flights from $city-fr to to $city-to

Table 9: Sample output from NLG3. (Dependency tree structures are not shown.) Typical values for attributes: \$time-dep = "10 a.m.", \$city-fr = "New York", \$city-to = "Miami"

is some additional cost associated with producing the syntactic dependency annotation necessary for NLG3, but virtually no additional cost is associated with NLG2, beyond collecting the data itself and identifying the attributes.

The trainable surface NLG systems in this paper differ from grammar-based systems in how they determine the attribute ordering and lexical choice. NLG2 and NLG3 automatically determine attribute ordering by simultaneously searching multiple orderings. In grammar-based approaches, such preferences need to be manually encoded. NLG2 and NLG3 solve the lexical choice problem by learning the words (via features in the maximum entropy probability model) that correlate with a given attribute and local context, whereas (Elhadad et al., 1997) uses a rule-based approach to decide the word choice.

While trainable approaches avoid the expense of crafting a grammar to determine attribute ordering and lexical choice, they are less accurate than grammar-based approaches. For short phrases, accuracy is typically 100% with grammar-based approaches since the grammar writer can either correct or add a rule to generate the phrase of interest once an error is detected. Whereas with NLG2 and NLG3, one can tune the feature patterns, search parameters, and training data itself, but there is no guarantee that the tuning will result in 100% generation accuracy.

Our approach differs from the corpus-based surface generation approaches of (Langkilde and Knight, 1998) and (Berger et al., 1996). (Langkilde and Knight, 1998) maps from semantics to words with a concept ontology, grammar, and lexicon, and ranks the resulting word lattice with corpus-based statistics, whereas NLG2 and NLG3 automatically learn the mapping from semantics to words from a corpus. (Berger et al., 1996) describes a statistical machine translation approach that generates text in the target language directly from the source text. NLG2 and NLG3 are also statistical learning approaches but generate from an actual semantic representation. This comparison suggests that statistical MT systems could also generate text from an "interlingua", in a way similar to that of knowledge-based translation systems.

We suspect that our statistical generation approach should perform accurately in domains of similar complexity to air travel. In the air travel domain, the length of a phrase fragment to describe an attribute is usually only a few words. Domains which require complex and lengthy phrase fragments to describe a single attribute will be more challenging to model with features that only look at word n-grams for $n \in \{2, 3\}$. Domains in which there is greater ambiguity in word choice will require a more thorough search, i.e., a larger value of N, at the expense of CPU time and memory. Most importantly, the semantic annotation scheme for air travel has the property that it is both rich enough to accurately represent meaning in the domain, but simple enough to yield useful corpus statistics. Our approach may not scale to domains, such as freely occurring newspaper text, in which the semantic annotation schemes do not have this property.

Our current approach has the limitation that it ignores the values of attributes, even though they might strongly influence the word order and word choice. This limitation can be overcome by using features on values, so that NLG2 and NLG3 might discover — to use a hypothetical example — that "flights leaving \$city-fr" is preferred over "flights from \$city-fr" when \$city-fr is a particular value, such as "Miami".

6 Conclusions

This paper presents the first systems (known to the author) that use a statistical learning approach to produce natural language text directly from a semantic representation. Information to solve the attribute ordering and lexical choice problems— which would normally be specified in a large handwritten grammar— is automatically collected from data with a few feature patterns, and is combined via the maximum entropy framework. NLG2 shows that using just local n-gram information can outperform the baseline, and NLG3 shows that using syntactic information can further improve generation accuracy. We conjecture that NLG2 and NLG3 should work in other domains which have a complexity similar to air travel, as well as available annotated data.

7 Acknowledgements

The author thanks Scott McCarley for serving as the second judge, and Scott Axelrod, Kishore Papineni, and Todd Ward for their helpful comments on this work. This work was supported in part by DARPA Contract # MDA972-97-C-0012.

References

John Bateman. 1996. Kpml development environment – multilingual linguistic resource development and sentence generation. Technical report, German Centre for Information Technology (GMD), Institute for Integrated Information and Publication Systems (IPSI), Darmstadt, Germany.

Adam Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. 1996. A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 22(1):39–71.

Michael Elhadad and Jacques Robin. 1996. An overview of surge: a reusable comprehensive syntactic realization component. Technical Report 96-03, Ben Gurion University, Beer Sheva, Israel.

Michael Elhadad, Kathleen McKeown, and Jacques Robin. 1997. Floating constraints in lexical choice. *Computational Linguistics*, pages 195–239.

Irene Langkilde and Kevin Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, University of Montreal, Montreal, Quebec, Canada.

Benoit Lavoie and Owen Rambow. 1997. A fast and portable realizer for text generation systems. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 265–268, Washington D.C., March 31–April 3.

M. W. Meteer, D. D. McDonald, S.D. Anderson, D. Forster, L.S. Gay, A.K. Huettner, and P. Sibun. 1987. Mumble-86: Design and implementation. Technical Report Technical Report COINS 87-87, University of Massachusetts at Amherst.

Adwait Ratnaparkhi. 1998. *Maximum Entropy Models for Natural Language Ambiguity Resolution.* Ph.D. thesis, University of Pennsylvania.

Ehud Reiter. 1995. Nlg vs. Templates. In *Proceedings of the 5th European Workshop on Natural Language Generation*, Leiden, The Netherlands.