Statistical Generation: Three Methods Compared and Evaluated

Anja Belz
ITRI, University of Brighton
Lewes Road, Brighton BN2 4GJ, UK
Anja.Belz@itri.brighton.ac.uk

Abstract

Statistical NLG has largely meant n-gram modelling which has the considerable advantages of lending robustness to NLG systems, and of making automatic adaptation to new domains from raw corpora possible. On the downside, n-gram models are expensive to use as selection mechanisms and have a built-in bias towards shorter realisations. This paper looks at treebank-training of generators, an alternative method for building statistical models for NLG from raw corpora, and two different ways of using treebank-trained models during generation. Results show that the treebank-trained generators achieve improvements similar to a 2-gram generator over a baseline of random selection. However, the treebank-trained generators achieve this at a much lower cost than the 2-gram generator, and without its strong preference for shorter realisations.

1 Introduction

Traditionally, NLG systems have been built as deterministic decision-makers, that is to say, they generate one string of words for any given input, in a sequence of decisions that increasingly specify the word string. In practice, this has meant carefully handcrafting generators to make decisions locally, at each step in the generation process. Such generators tend to be specialised and domain-specific, are hard to adapt to new domains, or even subdomains, and have no way of dealing with incomplete or incorrect input. Wide-coverage tools only exist for surface realisation, and tend to require highly specific and often idiosyncratic inputs. The rest of NLP has reached a stage where state-of-the-art tools are expected to be *generic*: wide-coverage, reusable and robust. NLG is lagging behind the field on all three counts.

The last decade has seen a new generation of NLG methodologies that are characterised by a separation between the definition of the generation space (all possible generation processes from inputs to outputs) on the one hand, and control over the decisions that lead to a (set of) output realisation(s), on the other.

In making this separation, *generate-and-select* NLG takes several crucial steps towards genericity: reusing systems becomes easier if the selection mechanism can be adjusted or replaced separately, without changing the definition of the generation space; coverage can be increased more easily if every expansion of the generation space does not have to be accompanied by handcrafted rules controlling the resulting increase in nondeterminism; and certain types of selection methods can provide robustness, for example through probabilistic choice.

Statistical generation has aroused by far the most interest among these methods, and it has mostly meant n-gram selection: a packed representation of all alternative (partial) realisations is produced, and an n-gram language model is applied to select the most likely realisation. N-gram methods have several desirable properties: they offer a fully automatic method for building and adapting control mechanisms for generate-and-select NLG from raw corpora (reusability); they base selections on statistical models (robustness); and they can potentially be used for deep as well as surface generation.

However, n-gram models are expensive to apply: in order to select the most likely realisation according to an n-gram model, all alternative realisations have to be generated and the probability of each realisation according to the model has to be calculated. This can get very expensive (even if packed representations of the set of alternatives are used), especially when the system accepts incompletely specified input, because the number of alternatives can be vast. In Halogen, Langkilde [2000] deals with trillions of alternatives, and the generator used in the experiments reported in this paper has up to 10^{40} alternative realisations (see Section 4.3 for empirical evidence of the relative inefficiency of n-gram generation).

Furthermore, n-gram models have a built-in bias towards shorter strings. This is because they calculate the likelihood of a string of words as the joint probability of the words, or, more precisely, as the product of the probabilities of each word given the $n-1$ preceding words. The likelihood of any string will therefore generally be lower than that of any of its substrings (see Section 4.3 for empirical evidence of this bias). This is wholly inappropriate for NLG where equally good realisations can vary greatly in length (see Section 5 for discussion of normalisation for length in statistical modelling).

The research reported in this paper is part of an ongoing research project[1] the purpose of which is to investigate issues in generic NLG. The experiments (Section 4.3) were carried out to evaluate and compare different methods for exploiting the frequencies of word sequences and word sequence cooccurrences in raw text corpora to build models for NL generation, and different ways of using such models during generation. One of the methods uses a standard 2-gram model for selection among all realisations (with a new selection algorithm, see Section 4.3). The other two use a *treebank-trained* model of the generation space (Section 3). The basic idea behind treebank-training of generators is simple: determine for the strings and substrings in the corpus the different ways in which the generator could have generated them, i.e. the different sequences of decisions that lead to them, then collect frequency counts for individual decisions, and determine a probability distribution over decisions on this basis. In the experiments, treebank-trained generation models are combined with two different ways of using them during generation, one locally and one globally optimal (Section 3.1). All three methods are evaluated on a corpus of weather forecasts (Section 4.1).

2 Generate-and-select NLG

Generate-and-select NLG separates the definition of the space of all possible generation processes (the generation space) from the mechanism that controls which (set of) realisation(s) is selected as the output. Generate-and-select methods vary primarily along three dimensions:

(i) Number of (partial) solutions generated at each step: some methods generate all possibilities, then select; some select a subset of partial solutions at each step; some use an automatically adaptable decision module to select the (single) next partial solution.

(ii) Type of decision-making module and method of construction/adaptation: statistical models, various machine learning techniques or manual construction/adaptation.

(iii) Size of subtask of the generation process that the method is applied to: from the entire generation process e.g. in text summarisation, to all of surface realisation for domain-independent generation.

Methods that can in principle be used to stochastically generate text have existed for a long time, but statistical generation from specified inputs started with Japan-Gloss [Knight *et al.*, 1994; 1995] (which replaced PENMAN's defaults with statistical decisions), while comprehensive statistical generation started with Nitrogen [Knight and Langkilde, 1998] (which represented the set of alternative realisations as a word lattice and selected the best with a 2-gram model) and its successor Halogen [Langkilde, 2000] (where the word lattice was replaced by a more efficient AND/OR-tree representation).

Since then, a steady stream of publications has reported work on statistical NLG. In FERGUS, Bangalore *et al.* used an XTAG grammar to generate a word lattice representation of a small number of alternative realisations, and a 3-gram model to select the best [Bangalore and Rambow, 2000b]. Humphreys *et al.* [2001] reused a PCFG trained for NL parsing to build syntactic generation trees from candidate syntactic nodes.

Recently, Habash [2004] reported work using structural 2-grams for lexical/syntactic selection tasks (using joint probability of word and parent word in dependency structures, instead of probability of word given preceding word), as well as conventional n-grams for selection among surface strings. Velldal *et al.* [2004] compared the performance of a 4-gram model trained on the BNC[2] with a Maximum Entropy model reused from a parsing application and trained on the small, domain-specific LOGON corpus, finding that the domain-specific ME model performs better on the LOGON corpus, but a combined model performs best.

Some statistical NLG research has looked at subproblems of language generation, such as ordering of NP premodifiers [Shaw and Hatzivassiloglou, 1999; Malouf, 2000], attribute selection in content planning [Oh and Rudnicky, 2000], NP type determination [Poesio *et al.*, 1999], pronominalisation [Strube and Wolters, 2000], and lexical choice [Bangalore and Rambow, 2000a].

In hybrid symbolic-statistical approaches, White [2004] prunes edges in chart realisation using n-gram models, and Varges uses quantitative methods for determining the weights on instance features in instance-based generation [Varges and Mellish, 2001].

The likelihood of realisations given concepts or semantic representations has been modeled directly, but is probably limited to small-scale and specialised applications: summarisation construed as term selection and ordering [Witbrock and Mittal, 1999], grammar-free stochastic surface realisation [Oh and Rudnicky, 2000], and surface realisation construed as attribute selection and lexical choice [Ratnaparkhi, 2000].

Some of the above papers compare the purely statistical methods to other machine learning methods such as memory-based learning and reinforcement learning. Some other research has focussed on machine learning methods, e.g. Walker *et al.* [2001] look at using a boosting algorithm to train a sentence plan ranker on a corpus of labelled examples, and Marciniak & Strube [2004] construe the entire generation process as a sequence of classification problems, solved by corpus-trained feature-vector classifiers.

Generate-and-select NLG has been applied either to all of surface realisation, or to a small subproblem in deep or surface realisation (not to the entire generation process); it is either very expensive or not guaranteed to find the optimal solution; and the models it has used are either shallow and unstructured, or require manual corpus annotation.

3 Treebank-Training of Generators

Treebank-training of generators is a method for modelling likelihoods of realisations in generation. It is introduced in this section first in terms of the general idea behind it (which could be implemented with various formalisms, training methods and generation algorithms), and then in Sec-

[1]Controlled Generation of Text (CoGenT) `http://www.itri.brighton.ac.uk/projects/cogent`.

[2]British National Corpus.

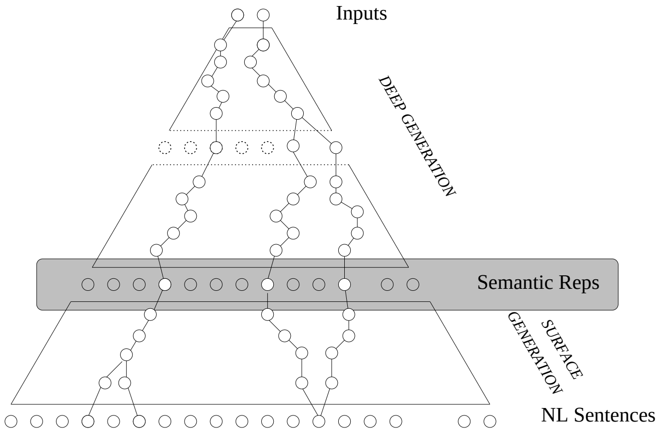

Figure 1: Generation space as decision tree.

tion 3.1 by a description of the actual technique that was used in the experiments reported below.

The generation space of a generator can be seen as a set of decision trees, where the root nodes correspond to the inputs, and the paths down the trees are all possible generation processes in the generator that lead to a realisation (leaf) node (a view discussed in more detail in [Belz, 2004]).

Consider the diagrammatic generation space representation in Figure 1. It shows examples of three realisations and the sequences of generator decisions that generate them, represented as paths connecting decision nodes and leading to realisation nodes. In this view of the generation space, using an n-gram model is equivalent to following all paths (from the given input node) down to the realisations, and then applying the model to select the most likely realisation.

An alternative is to estimate the likelihood of a realisation in terms of the likelihoods of the generator decisions that give rise to it, looking at the possible sequences of decisions that generate the realisation (its 'derivations'). One way of doing this is to say the likelihood of a string is the sum of the likelihoods of its derivations. To train such a model on a corpus of raw text, the set of derivations for each sentence is determined, frequencies for individual decisions are added up, and a probability distribution over sets of alternative decisions is estimated.

If a sentence has more than one derivation, as in the example on the right of Figure 1, there are two possibilities: either the frequency counts are evenly divided between them, or disambiguation is carried out to determine the correct derivation. The former uses the surface frequencies of word strings regardless of their meaning and structure, as do n-gram models. The latter is complicated by the fact that there is not always a single 'correct' derivation in generation (e.g. an expression may end up being passivised in more than one way).

There are at least three strategies for using a treebank-trained model during generation: (i) select the most likely decision at each choice point; (ii) select the most likely generation process (joint probability of all decisions); or (iii) select the most likely string of words (summed probabilities of all generation processes that generate the string). The first would always make the same decision given the same alternatives, whereas for (ii) and (iii) it would depend also on the other decisions in the derivation(s). On the other hand, the complexity of (iii) is much greater than that of (ii) which in turn is greater than that of (i).

3.1 Context-free Generator Treebank-Training without Disambiguation

There are many different ways of representing generator decisions and annotating sentences with derivations, and treebank-training is clearly not equally suitable for all types of generators. In the current version of the method, generation rules must be context-free with atomic arguments. Derivations for sentences in the corpus are then standard context-free derivations, and corpora are annotated in the standard context-free way with brackets and labels.

No disambiguation is performed, the assumption being that all derivations are equally good for a sentence. If a sentence has more than one derivation, frequency counts are divided equally between them.

The three basic steps in context-free generator treebank-training are:

1. For each sentence in the corpus, find all *generation processes* that generate it, that is, all the ways in which the generator could have generated it. For each generation process, note the sequences of generator decisions involved in it (the *derivations* for the sentence). If there is no complete derivation, maximal partial derivations are used instead.
2. Annotate the (sub)strings in the sentence with the derivation, resulting in a *generation tree* for the sentence. If there is more than one derivation for the sentence, create a set of annotated trees. The resulting annotated corpus is a *generation treebank*.
3. Obtain frequency counts for each individual decision from the annotations, adding $1/n$ to the count for every decision, where n is the number of alternative derivations; convert counts into probability distributions over alternative decisions, smoothing for unseen decisions.

The probability distribution is currently smoothed with the simple add-1 method[3]. This is equivalent to Bayesian estimation with a uniform prior probability on all decisions, and is entirely sufficient for present purposes given the very small vocabulary and the good coverage of the data. A standard maximum likelihood estimation is performed: the total number of occurrences of a decision (e.g. passive) is divided by the total number of occurrences of all alternatives (e.g. passive + active). In the context-free setting, a decision type corresponds to a nonterminal N, and decisions correspond to expansion rules $N \rightarrow \alpha$. Given a function $c(x)$ which returns the frequency count for a decision x, normalising each occurrence by the number of derivations for the sentence, the probability of a decision is obtained in the standard way (R is the set of all decisions):

$$p(N \rightarrow \alpha) = \frac{c(N \rightarrow \alpha)}{\sum_{i:N \rightarrow \alpha_i \in R} c(N \rightarrow \alpha_i)} \qquad (1)$$

[3]Also known as applying Laplace's law.

Greedy generation

One way of using a treebank-trained generator is to make the single most likely decision at each choice point in a generation process. This is not guaranteed to result in the most likely generation *process*, but the computational cost in application is exceedingly low.

Viterbi generation

The alternative is to do a Viterbi search of the generation forest for a given input, which maximises the joint likelihood of all decisions taken in the generation process. This is guaranteed to select the most likely generation process, but is considerably more expensive. The efficiency of greedy probabilistic generation, Viterbi generation and 2-gram post-selection is compared in Section 4.3 below.

A possible alternative to greedy search is to use a non-uniform random distribution proportional to the likelihoods of alternatives. E.g. if there are two alternative decisions $D1$ and $D2$, with the model giving $p(D1) = .8$ and $p(D2) = .2$, then the generator would decide $D1$ with probability $.8$, and $D2$ with a probability of $.2$ for an arbitrary input (instead of always deciding $D1$ as does the greedy generator). However, such a strategy, while increasing variation, would come at the price of lowering the overall likelihood of making the right decision. With the strategy of always going for the most frequent alternative, the overall likelihood of making the right decision when faced with the choice $D1$ or $D2$ is simply $.8$ in the current example (1 for $D1$, 0 for $D2$). With the likelihood-proportional random strategy, however, the overall likelihood of making the right decision is only $.68$ ($.64$ for $D1$, $.04$ for $D2$). Variation can alternatively be increased by making the model more fine-grained.

4 Evaluation on Weather Forecast Generation

4.1 Domain and Data: Weather Forecasting

The corpus used in the experiments reported below is the SUMTIME-METEO corpus created by the SUMTIME project team in collaboration with WNI Oceanroutes [Sripada *et al.*, 2002]. The corpus was collected by WNI Oceanroutes from the commercial output of five different (human) forecasters, and each instance in the corpus consists of three numerical data files (output by three different weather simulators) and the weather forecast file written by the forecaster on the evidence of the data files (and sometimes additional resources). Following the SUMTIME work, the experiments reported below focussed on the part of the forecasts that predicts wind characteristics for the next 15 hours. Such ‘wind statements’ look as follows (for 10-08-01):

```
---------------------------------------------
2.FORECAST 1500 GMT FRI 10-Aug,TO 0600GMT SAT
11-Aug 2001
=====WARNINGS:        NIL             =======
WIND(KTS)  CONFIDENCE: HIGH
  10M:     WNW-NW 12-15 BACKING W'LY 05-10 BY
           MIDNIGHT, THEN SW-SSW BY MORNING
  50M:     WNW-NW 15-18 BACKING W'LY 06-12 BY
           MIDNIGHT, THEN SW-SSW BY MORNING
---------------------------------------------
```

```
---------------------------------------------
O1, O2 AND O3 OIL FIELDS (EAST OF SHETLAND)
10-08-01

10/18  WNW  11  13  17  1.7  2.7  NW  1.5  7
10/21  W     8  10  12  1.5  2.4  NW  1.4  7
11/00  W     7   8  10  1.4  2.2  NW  1.4  7
11/03  SW    7   8  10  1.4  2.2  NW  1.3  7
11/06  SW    7   8  10  1.3  2.1  NW  1.3  7
11/09  SSW  10  12  15  1.3  2.1  NW  1.2  7
11/12  S    14  17  21  1.5  2.4  NW  1.2  7
11/15  S    20  25  31  1.8  2.9  WNW 1.3  7
11/18  S    22  27  34  1.9  3.0  SW  1.5  8
11/21  S    24  30  37  2.3  3.7  S   1.7  8
12/00  S    28  35  43  3.0  4.8  S   1.9  8
12/03  S    28  35  43  3.0  4.8  S   1.9  8
12/06  SW   27  33  41  3.0  4.8  S   2.0  8
12/09  WSW  26  32  40  2.9  4.6  SSW 2.0  8
12/12  WSW  25  31  39  2.9  4.6  SW  2.0  8
12/15  WSW  25  31  39  2.9  4.6  WSW 2.0  8
12/18  WSW  24  30  37  3.1  5.0  WSW 2.1  8
12/21  SW   23  28  35  2.9  4.6  WSW 2.2  9
13/00  SW   21  26  32  2.8  4.5  WSW 2.3  9
13/03  SW   19  23  29  2.4  3.8  WSW 2.1  8
13/06  SSW  19  23  29  2.2  3.5  SW  2.0  8
13/09  SSW  20  25  31  2.2  3.5  SSW 1.9  8
13/12  SSW  21  26  32  2.4  3.8  SSW 1.8  8
---------------------------------------------
```

Figure 2: Metereological data file for 10-08-01.

To keep things simple, only the data file type that contains (virtually all) the information about wind parameters (the .tab file type) was used. Figure 2 is the .tab file corresponding to the above forecast. The first column is the day/hour time stamp, the second the wind direction predicted for the corresponding time period; the third the wind speed at 10m above the ground; the fourth the gust speed at 10m; and the fifth the gust speed at 50m. The remaining columns contain wave data.

The mapping from time series data to forecast is not straightforward (even when all three data files are taken into account). An example here is that while the wind direction in the first part of the wind statement is given as WNW-NW, NW does not appear as a wind speed anywhere in the data file. Nor is it obvious why the wind speeds 11, 12 and 7 are mapped to the two ranges 12-15 and 5-10.

The SUMTIME Project construed the mapping from time series data to weather forecasts as two tasks [Sripada *et al.*, 2003]: selecting a subset of the time series data to be included in the forecast, and expressing this subset of numbers as an NL forecast. The focus of the research reported here is not the numerical summarisation of time series data, but NLG techniques. Therefore, the SUMTIME-METEO corpus was converted into a parallel corpus of wind statements and the wind data included in each statement. The wind data is a vector of time stamps and wind parameters, and was ‘reverse-engineered’, by automatically aligning wind speeds and wind directions in the forecasts with time-stamps in the data file. In order to do this, wind speed and directions in the data file have to be matched with those in the forecast. This was not straightforward either, because more often than not, there is no exact match in the data file for the wind speeds and directions in the forecast. The strategy adopted in the work reported here was to select the time stamp beside the first exact match, and to leave time undefined if there was no exact match. The instances in the final training corpus look as follows (same example, 10-08-01, resulting in two instances):

```
---------------------------------------------
WNW-NW 12 15 - - - W 05 10 - - 21 SW-SSW - -
- - 03
WNW-NW 12-15 BACKING W'LY 05-10 BY MIDNIGHT
THEN SW-SSW BY MORNING

SSW 05 10 - - - S - - - - - - 26 30 38 38 -1
SSW 05-10 BACKING S'LY AND INCREASING 26-30
GUSTS 38 BY END OF PERIOD
---------------------------------------------
```

4.2 Automatic Generation of Weather Forecasts

The three generation methods compared below are all generate-and-select methods. The idea was to build a basic generator that for any given input generates a set of alternatives that reflects *all* the variation found in the corpus (rather than deciding which alternative to select in which context), and then to create statistical decision makers trained on the corpus to select (a subset of) alternatives. The rest of this section describes the basic generator, and the following section describes the experiments that were carried out.

The basic generator was written semi-automatically as a set of generation rules with atomic arguments that convert an input vector of numbers in steps to a set of NL forecasts. The automatic part was analysing the entire corpus with a set of simple chunking rules that split wind statements into wind direction, wind speed, gust speed, gust statements, time expressions, transition phrases (such as *and increasing*), premodifiers (such as *less than* for numbers, and *mainly* for wind direction), and post-modifiers (e.g. *in or near the low centre*). The manual part was to write the chunking rules themselves, and higher-level rules that combine different sequences of chunks into larger components.

The higher-level generation rules were based on an interpretation of wind statements as sequences of fairly independent units of information, each containing as a minimum a wind direction or wind speed range, and as a maximum all the chunks listed above. The only context encoded in the rules was whether a unit of information was the first in a wind statement, and whether a wind statement contained wind direction (only), wind speed (only), or both. The final generator takes as inputs number vectors of length 6 to 36, and has a large amount of non-determinism. For the simplest input (one number), it generates 8 alternative realisations. For the most complex input (36 numbers), it generates 4.96×10^{40} alternatives (as a tightly packed AND/OR-tree).

4.3 Experiments and Results

The converted corpus (as described in Section 4.1 above) consisted of 2,123 instances, corresponding to a total of 22,985 words. This may not sound like much, but considering that the entire corpus only has a vocabulary of about 90 words (not counting wind directions), and uses only a handful of different syntactic structures, the corpus provides extremely good coverage (an initial impression confirmed by the small differences between training and testing data results below).

The corpus was divided at random into training and testing data at a ratio of 9:1. The training set was used to treebank-train the weather forecast generation grammar (as described in Section 3.1) and a back-off 2-gram model (using the SRILM toolkit, [Stolcke, 2002]). The treebank-trained generation grammar was used in conjunction with a greedy and a Viterbi generation method. The 2-gram model was used in more or less exactly the way reported in the Halogen papers [Langkilde, 2000]. That is to say, the packed AND/OR-tree representation of all alternatives is generated in full, then the 2-gram model is applied to select the single best one.

One small difference to Halogen is that a Viterbi algorithm was used to identify the single most likely string. This is achieved as follows. The AND/OR-tree is interpreted directly as a finite-state automaton where the states correspond to words as well as to the nodes in the AND/OR-tree. The transitions are assigned probabilities according to the 2-gram model, and then a straightforward single-best Viterbi algorithm is applied to find the best path through the automaton.

Random selection among all alternatives was used as a baseline. All results were evaluated against the gold standard (the human written forecasts) of the test set. Results were validated with 5-fold cross-validation.

In the following overview of the results[4], the similarity between automatically generated forecasts and gold standard was measured by conventional string-edit (SE) distance with substitution at cost 2, and deletion and insertion at cost 1. Baseline results are given as absolute SE scores, results for the non-random generators in terms of improvement over the baseline (reduction in string-edit distance, with SE score in brackets)[5].

	Training set	Test set
Random	14.92	14.37
TT/Greedy	-50.64% (7.36)	-50.30% (7.41)
TT/Viterbi	-52.32% (7.11)	-52.11% (7.14)
2-gram	-58.44% (6.20)	-57.91% (6.27)

The SE scores show that, as expected, the improvements for the training set are slightly larger in all cases. The greedy generator achieves a significant improvement over random selection, but is outperformed by the Viterbi generator, with 2-gram selection the clear overall winner.

The generated strings were also evaluated using the BLEU metric [Papineni *et al.*, 2001] which is ultimately based on n-gram agreement between generated and gold standard strings. In simple terms, the more 1-grams, 2-grams, ... and n-grams are the same between two strings, the higher their BLEU score. This implies that BLEU with $n = 1$ is the most closely related to SE scoring (which can also be seen from the similarity between the relative scores assigned by BLEU_1 and SE

[4]All numbers in this paper are either given in their entirety or truncated (not rounded).

[5]Percentage reductions were calculated on the non-truncated string-edit distances.

	BLEU$_1$		BLEU$_2$		BLEU$_3$		BLEU$_4$	
	Train	Test	Train	Test	Train	Test	Train	Test
Random	0.479	0.485	0.390	0.382	0.310	0.301	0.225	0.215
TT/Greedy	0.678	0.671	0.598	0.591	0.521	0.514	0.437	0.429
TT/Viterbi	0.674	0.668	0.595	0.589	0.518	0.512	0.434	0.427
2-gram	0.708	0.704	0.620	0.614	0.542	0.536	0.460	0.453

Table 1: BLEU$_n$ scores for training and test sets, $1 \leq n \leq 4$.

to the generators). Table 1 gives BLEU$_n$ scores for the 4 generators, for $1 \leq n \leq 4$. These scores give a different impression of the results. BLEU consistently scores the Viterbi generator *lower* than the greedy generator, although the difference between them is only 0.003 on average, less than the average mean deviation for the two generators across the five runs (0.005). The difference between the random generator and the other methods increases significantly with increasing n. For $n = 2$ and $n = 3$, the differences between the 2-gram generator and the two treebank-trained generators drops noticeably.

Results were stable over the five runs of the cross-validation. For the test set, the mean deviation in SE scores was between 0.13 and 0.35, and in BLEU scores between 0.0034 and 0.0065. The margins by which the 2-gram generator outperformed the other two were nearly identical across the five runs. The SE score of the greedy generator was lower (by nearly identical margins) than that of the Viterbi generator in all runs. All four BLEU scores of the greedy generator, however, were slightly higher than those of the Viterbi generator in four out of five runs, and slightly lower in one run. Small mean deviation figures and consistency of results confirm the significance of the differences between the scores in the SE and BLEU evaluations to some extent.

There is a debate over the degree to which an evaluation metric for NLG should be sensitive to word order and word adjacency. The appropriate degree of word-order sensitivity may vary from one evaluation task to the next, but for the task of evaluating weather forecasts it certainly is important. A clear example is time expressions. A forecast can contain up to five different time expressions, which have to observe the correct chronological order. It is not appropriate to reward the mere presence (regardless of place in the string) of, say, *by midnight* (which is what some evaluation metrics are specifically designed to do, e.g. [Bangalore *et al.*, 2000]). SE scoring has a tendency to reward proximity to the intended place (although not in a very straightforward way), and BLEU is increasingly strict about place with increasing n.

The SE score gives an intuitive, initial impression of how much the three methods have learned in comparison to the baseline: it is easy to conceptualise how different one string is from another if you know there are two insert and delete operations between them (not so easy with the BLEU scores). However, by far the more complete picture (and the most appropriate to this evaluation task) is given by BLEU. It shows that the 2-gram generator does better than the other two, but not by a very large margin. The margin is important considering the far greater expense of n-gram generation. The following table shows the total amount of time it took to test the training and test sets with the different methods in one (controlled) run.

Total time (minutes)	Training set	Test set
TT/greedy	23m52s	2m06s
TT/Viterbi	3h22m33s	27m05s
2-gram	24h04m05s	3h35m06s

Testing the training set took all methods about 7 times as long as the test set. The Viterbi generator took about 13 times longer than the greedy generator for the test set, and about 9 times longer for the training set. The 2-gram generator took 6.5 times longer than the Viterbi generator on the test set, and 7.5 times longer on the training set.

To make the comparison fair, the Viterbi and the 2-gram generator were implemented identically as far as possible. Both start by generating the packed AND/OR-tree of all alternatives. During this process, the former makes a note of the rule probabilities (at the AND and OR nodes) along with the rules, and then directly identifies the most likely realisation by a Viterbi method. The 2-gram generator first has to annotate the AND/OR-tree with 2-gram probabilities looked up in the 2-gram model before using the same Viterbi method. The overhead of looking up the 2-gram model to score the packed representation is the only difference between the two methods and multiplies out into a large overhead in computing time for the 2-gram generator[6], which would make it unsuitable for use in practical applications.

The random generator has no preference for shorter strings at all, and has an average string length almost twice that of the other generators. The 2-gram generator has an almost absolute preference of shorter over longer strings, and so produces the shortest strings. The Viterbi generator does not prefer shorter strings, but does prefer shorter derivations, and there is a correlation between string length and derivation length. The greedy generator does not have a built-in preference for shorter strings or derivations, but it reflects the fact that short (sub)strings were more frequent in the training corpus:

gold:	10.8	greedy:	9.3	2-gram:	8.7
random:	17.2	Viterbi:	9.0		

In some application domains, the n-gram model's preference for shorter strings is irrelevant: e.g. in speech recognition (where n-gram models are used widely) the alternatives among which the model must choose are always of the same length. In language generation, where equally good alterna-

[6]Even if the lookup can be implemented more efficiently there will always be some overhead multiplied by the total number of 2-grams in the packed representation.

tives can vary greatly in length, this preference is positively harmful (see following section for discussion of methods to counteract this bias).

5 Discussion and Further Research

The n-gram model's bias towards shorter strings is an example of a general case: whenever the likelihood of a larger unit that can vary in length (e.g. sentence) is modelled in terms of the joint probability of length-invariant smaller units, larger units that are composed of fewer smaller units are more likely. The possibility of counteracting this bias has been investigated. In parsing, Magerman & Marcus [1991] and Briscoe & Carroll [1993] used the geometric mean of probabilities instead of the product of probabilities. In later work, Briscoe & Carroll [1992] use a normalisation approach, the equivalent of which for n-gram selection would be to 'pad' shorter alternatives with extra 'dummy' words up to the length of the longest alternative, and to use the geometric mean of the probabilities assigned by the n-gram model to the non-dummy words as the probability of any dummy word. It has been observed that such methods turn a principled probabilistic model into an *ad hoc* scoring function [Manning and Schuetze, 1999, p. 443]. It certainly means getting rid of the n-gram model's particular independence assumptions, without replacing them with a principled alternative.

The NLG community is traditionally wary of using evaluation metrics like the ones used here. NLU, and in particular the numbers-driven parsing community, has not tended to worry about the fact that the gold standard parse is usually not the only correct one. It is accepted that it is an imperfect way of evaluating results, albeit the only realistic possibility, especially for large amounts of data. Furthermore, where a parser or generator has been trained directly on a corpus, it is a fair way of estimating how well a method has succeeded in its aim, namely to learn from the corpus.

Some typical example outputs from all generators are shown in an appendix to this paper. The greedy and Viterbi generators tend to have very similar output. Both rely on a small number of short, high-frequency phrases, e.g. `SOON`, `VEERING`, `INCREASING`, and `LATER`. The 2-gram generator has a similar tendency, but with a larger number of phrases and more variation. If the generation rules allow it, the greedy generator always prefixes `SOON` (all three examples), whereas the Viterbi generator can avoid it (third example).

The quality of the 2-gram generator's output is independent of the quality of the weather forecast generation rules, as long as they reflect all the variation in the corpus. However, the other two generators are entirely dependent on the quality of the generation rules, in particular on whether enough information is incorporated in the rules to base decisions on. The only room for improving the 2-gram generator is in using alternative statistical estimation techniques, but there is a lot of potential for improving the probabilistic-rule-driven generators, by (i) improving the quality of the generation rules (e.g. by incorporating more contextual information in the generation rules, in particular always including a time stamp); (ii) using alternative methods for building treebank models (e.g. extending the local context of rules has proved useful in parsing); and (iii) using alternative methods for exploiting treebank models during generation, e.g. it would be good to have a generation strategy that has (some of) the vastly superior efficiency of the greedy generator without its repetitiveness, while not sacrificing (too much of) the overall likelihood of making the right decision (unlike the non-uniform random strategy discussed in Section 3.1). Future research will look at all three areas, using for evaluation a larger and more varied corpus from a different domain as well as the SUMTIME corpus.

The three statistical generation methods evaluated in this paper all work with raw corpora, but vary hugely in efficiency. Generation speed and the ability to adapt generators to new domains with no annotation bottleneck are crucial for the development of practical, generic NLG tools. The work presented in this paper is intended to be a step in this direction.

Acknowledgements

The research reported in this paper is part of the CoGenT project, an ongoing research project supported under UK EPSRC Grant GR/S24480/01. Many thanks to John Carroll, Roger Evans and Richard Power, as well as to the anonymous reviewers, for very helpful comments.

References

[Bangalore and Rambow, 2000a] S. Bangalore and O. Rambow. Corpus-based lexical choice in natural language generation. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL '00)*, pages 464–471, 2000.

[Bangalore and Rambow, 2000b] S. Bangalore and O. Rambow. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING '00)*, pages 42–48, 2000.

[Bangalore *et al.*, 2000] S. Bangalore, O. Rambow, and S. Whittaker. Evaluation metrics for generation. In *Proceedings of the 1st International Conference on Natural Language Generation (INLG '00)*, pages 1–8, 2000.

[Belz, 2004] A. Belz. Underspecification for NLG. In *INLG04 Posters: Extended Abstracts of Posters Presented at the Third International Conference on Natural Language Generation*. Technical Report ITRI-04-01, ITRI, University of Brighton, 2004.

[Briscoe and Carroll, 1992] T. Briscoe and J. Carroll. Probabilistic normalisation and unpacking of packed parse forests for unification-based grammars. In *Proceedings of the AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, pages 33–38, 1992.

[Briscoe and Carroll, 1993] T. Briscoe and J. Carroll. Generalised probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59, 1993.

[Habash, 2004] N. Habash. The use of a structural n-gram language model in generation-heavy hybrid machine translation. In A. Belz, R. Evans, and P. Piwek, editors, *Proceedings of the Third International Conference on Natural*

Language Generation (INLG '04), volume 3123 of *LNAI*, pages 61–69. Springer, 2004.

[Humphreys *et al.*, 2001] K. Humphreys, M. Calcagno, and D. Weise. Reusing a statistical language model for generation. In *Proceedings of the 8th European Workshop on Natural Language Generation (EWNLG '01)*, pages 86–91, 2001.

[Knight and Langkilde, 1998] K. Knight and I. Langkilde. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics (COLING-ACL '98)*, pages 704–710, 1998.

[Knight *et al.*, 1994] K. Knight, I. Chander, M. Haines, V. Hatzivassiloglou, E. Hovy, M. Iida, S. Luk, A. Okumura, R. Whitney, and K. Yamada. Integrating knowledge bases and statistics in MT. In *Proceedings of the 1st Conference of the Association for Machine Translation in the Americas (AMTA '94)*, pages 134–141, 1994.

[Knight *et al.*, 1995] K. Knight, I. Chander, M. Haines, V. Hatzivassiloglou, E. Hovy, M. Iida, S. Luk, R. Whitney, and K. Yamada. Filling knowledge gaps in a broad-coverage MT system. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI '95)*, pages 1390–1397, 1995.

[Langkilde, 2000] I. Langkilde. Forest-based statistical sentence generation. In *Proceedings of the 6th Applied Natural Language Processing Conference and the 1st Meeting of the North American Chapter of the Association of Computational Linguistics (ANLP-NAACL '00)*, pages 170–177, 2000.

[Magerman and Marcus, 1991] D. Magerman and M. Marcus. Pearl: A probabilistic chart parser. In *Proceedings of the 2nd International Workshop on Parsing Technologies*, pages 193–199, 1991.

[Malouf, 2000] R. Malouf. The order of prenominal adjectives in natural language generation. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL '00)*, pages 85–92, 2000.

[Manning and Schuetze, 1999] C. Manning and H. Schuetze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[Marciniak and Strube, 2004] T. Marciniak and M. Strube. Classification-based generation using TAG. In A. Belz, R. Evans, and P. Piwek, editors, *Proceedings of the Third International Conference on Natural Language Generation (INLG '04)*, volume 3123 of *LNAI*, pages 100–109. Springer, 2004.

[Oh and Rudnicky, 2000] A. Oh and A. Rudnicky. Stochastic language generation for spoken dialogue systems. In *Proceedings of the ANLP-NAACL 2000 Workshop on Conversational Systems*, pages 27–32, 2000.

[Papineni *et al.*, 2001] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. BLEU: A method for automatic evaluation of machine translation. IBM research report, IBM Research Division, 2001.

[Poesio *et al.*, 1999] M. Poesio, R. Henschel, J. Hitzeman, and R. Kibble. Statistical NP generation: A first report. In R. Kibble and K. van Deemter, editors, *Proceedings of ESSLLI Workshop on NP Generation*, pages 30–42, 1999.

[Ratnaparkhi, 2000] A. Ratnaparkhi. Trainable methods for surface natural language generation. In *Proceedings of the 6th Applied Natural Language Processing Conference and the 1st Meeting of the North American Chapter of the Association of Computational Linguistics (ANLP-NAACL '00)*, pages 194–201, 2000.

[Shaw and Hatzivassiloglou, 1999] J. Shaw and V. Hatzivassiloglou. Ordering among premodifiers. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL '99)*, pages 135–143, 1999.

[Sripada *et al.*, 2002] S. Sripada, E. Reiter, J. Hunter, and J. Yu. SUMTIME-METEO: Parallel corpus of naturally occurring forecast texts and weather data. Technical Report AUCS/TR0201, Computing Science Department, University of Aberdeen, 2002.

[Sripada *et al.*, 2003] S. Sripada, E. Reiter, J. Hunter, and J. Yu. Exploiting a parallel text-data corpus. In *Proceedings of Corpus Linguistics 2003*, pages 734–743, 2003.

[Stolcke, 2002] A. Stolcke. SRILM: An extensible language modeling toolkit. In *Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP '02)*, pages 901–904,, 2002.

[Strube and Wolters, 2000] M. Strube and M. Wolters. A probabilistic genre-independent model of pronominalization. In *Proceedings of the 6th Applied Natural Language Processing Conference and the 1st Meeting of the North American Chapter of the Association of Computational Linguistics (ANLP-NAACL '00)*, pages 18–25, 2000.

[Varges and Mellish, 2001] S. Varges and C. Mellish. Instance-based natural language generation. In *Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL '01)*, pages 1–8, 2001.

[Velldal *et al.*, 2004] E. Velldal, S. Oepen, and D. Flickinger. Paraphrasing treebanks for stochastic realization ranking. In *Proceedings of the 3rd Workshop on Treebanks and Linguistic Theories (TLT '04)*, Tuebingen, Germany, 2004.

[Walker, 2001] M. Walker. An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. *Journal of Artificial Intelligence Research*, 12:387–416, 2001.

[White, 2004] M. White. Reining in CCG chart realization. In A. Belz, R. Evans, and P. Piwek, editors, *Proceedings of the Third International Conference on Natural Language Generation (INLG '04)*, volume 3123 of *LNAI*, pages 182–191. Springer, 2004.

[Witbrock and Mittal, 1999] M. Witbrock and V. Mittal. Ultra-summarization: A statistical approach to generating highly condensed non-extractive summaries. In *Proceedings of the 22nd International Conference on Research and Development in Information Retrieval (SIGGIR '99)*, pages 315–316, 1999.

A Example output

10Jul2001_03/3:

Gold	ENE SOON 20-24 BACKING NE 08-12 BY LATE AFTERNOON
Random	ENE VERY SOON 20-24 AHEAD OF THE FRONT AT FIRST EASING AND BACKING NE’LY 8-12 BY THE AFTERNOON
Greedy	SOON ENE INCREASING 20-24 VEERING NE 8-12 LATER
Viterbi	SOON ENE INCREASING 20-24 VEERING NE 8-12 LATER
2-gram	ENE AND 20-24 BACKING NE 8-12 LATER

10Nov2000_16/1:

Gold	N 10 OR LESS VEERING SE AND RISING 20-24 LATE IN THE PERIOD
Random	N 10 OR LESS THEN LATE IN PERIOD BACKING SE STEADILY INCREASING TO 20-24 IN FRONTAL ZONE FOR A TIME EARLY EVENING
Greedy	SOON N 10 OR LESS VEERING SE INCREASING 20-24 LATER
Viterbi	SOON N 10 OR LESS VEERING SE INCREASING 20-24 LATER
2-gram	N 10 OR LESS BACKING SE AND 20-24 BY EVENING

10Feb2002_04/1:

Gold	WNW 22-28 GUSTS 38 VEERING NW BY MIDDAY THEN BACKING AND DECREASING W’LY 18-22 BY MID EVENING
Random	WNW 22-28 GUSTS 38 IN ANY SHOWERS THEN SLOWLY BACKING TO NW BY MID MORNING WEST 18-22 LATER
Greedy	SOON WNW 22-28 GUSTS 38-38 VEERING NW LATER VEERING W 18-22 LATER
Viterbi	WNW 22-28 GUSTS 38-38 VEERING NW LATER W 18-22 LATER
2-gram	WNW 22-28 GUSTS 38 LATER NW LATER W 18-22 LATER