An Empirical Verification of Coverage and Correctness for a General-Purpose Sentence Generator

Irene Langkilde-Geary
Information Sciences Institute
University of Southern California
ilangkil@isi.edu

Abstract

This paper describes a general-purpose sentence generation system that can achieve both broad scale coverage and high quality while aiming to be suitable for a variety of generation tasks. We measure the coverage and correctness empirically using a section of the Penn Treebank corpus as a test set. We also describe novel features that help make the generator flexible and easier to use for a variety of tasks. To our knowledge, this is the first empirical measurement of coverage reported in the literature, and the highest reported measurements of correctness.

1 Introduction

Natural language generation (NLG) is a subtask of a wide variety of applications. Such applications include machine translation, human-computer dialogue, summarization, report creation, automatic technical documentation, proof/decision explanation, customized instructions, item and event descriptions, question answering, tutorials, stories, and more. While many applications use a custom-built generator, a general-purpose system can facilitate reuse of resources and reduce the costs of building applications.

Research into general-purpose generation has tended to focus on sentence realization, which is one of the most common recurring subtasks of generation. Sentence realization is the process of transforming a syntactic sentence plan into a linearly-ordered, grammatical string of morphologically inflected words. To be generally useful, a realizer needs to be able to handle a wide array of syntactic phenomena. It also needs to produce grammatically correct output.

Prominent general-purpose realization systems developed to date include FUF/Surge (Elhadad, 1993) and (Elhadad and Robin, 1998), RealPro (Lavoie and Rambow, 1997), Penman/KPML (Bateman, 1996), and Nitrogen (Langkilde and Knight, 1998a), (Knight and Hatzivassiloglou, 1995). These systems have demonstrated their general usefulness by being deployed in a variety of different applications. However, it is still difficult ascertain the degree to which they have achieved broad coverage of natural language or high quality output because no empirical evaluation has been performed.

At best, suites of example inputs have been used for regression testing. However, such regression suites are biased towards the capabilities of their respective systems and consist of relatively few inputs compared to the variety of input classes that are possible. For example, there are currently 500 test inputs distributed with Surge, and about 210 for English with KPML. At any rate, no matter how large the regression suite may be, the inherent irregularity of natural language makes regression testing inadequate as a means of assessing coverage or quality.

In practice, there is a seemingly irreconcilable conflict between broad coverage and high quality output. It is usually the case that the rules and class features are simultaneously too general to rule out undesirable combinations, and yet too restrictive to

"battalion assigned to go on defense/offense"
"defending battalion"
"offending battalion"
(Meteer, 1990)

"won/lost three straight/consecutive X"
"won/lost three straight"
"won/lost three consecutive"
(J.Robin, 1994)

"finger" vs "digit" vs "phalange"
"in the end zone" vs * "on the end zone"
"tell her hi" vs * "say her hi"
"the vase broke" vs * "the food ate"
(Knight et al., 1995)

Figure 1: Expressibility problems

allow some combinations that are valid. Figure 1 shows some examples of this expressibility problem. High quality output is thus much easier to achieve with smaller-scale applications or in limited domains.

In this paper we introduce HALogen, a general-purpose sentence generator that achieves both broad coverage of English and high quality output as measured against an unseen section of the Penn Treebank (Marcus et al., 1993). We compare it to the only other generation system that has performed such an evaluation, a limited-purpose system named Fergus (Bangalore et al., 2000). HALogen's development has been guided in part by the question, "What is the simplest input notation that suffices to represent and correctly generate all valid sentences, and yet at the same time is easy for applications to use?" We describe novel aspects that help make the generator flexible and easier to use for a variety of applications. Section 2 gives a brief overview of the HALogen sentence generation system. Section 3 describes the setup of the empirical evaluation, and Section 4 discusses the results. Finally, Section 5 concludes.

2 HALogen

HALogen is a successor to Nitrogen (Langkilde and Knight, 1998a), (Langkilde and Knight, 1998b) and (Langkilde, 2000). It is a hybrid symbolic and statistical system, with a two-stage architecture. In the first stage, symbolic knowledge is used to transform an input into a forest of possible expressions. In the second stage a statistical ranker computes the most likely expression using a corpus-based probabilistic model. This section describes HALogen's input and processing stages. It also compares HALogen to its predecessor.

2.1 Input

The input to HALogen is a labeled feature-value structure. There are two main types of features: relations and properties. Relation features describe the relationship between the instance (or head value) and other values. A value can be a word, a concept, or a compound value composed of nested feature-value structures. Relations can be syntactic, semantic, or even non-linguistic. Two example inputs are shown in Figure 2.

As seen in the examples, multiple relations can appear at each nesting level in the the input. Most relations can only occur once each at any given nesting level. The main exceptions are modifier and adverbial relations, which can occur any number of times. Relations are order-independent, which means that the order in which relations occur in the input does not affect the order in which their values occur in the output. An exception is when the same relation occurs more than once at the same nesting level. In this situation, the values with the same relation will occur adjacent to each other in the output in same order that they appeared in the input, unless a PermuteNodes flag is set. If this flag is set, all possible adjacent permutations will be tried.

HALogen's input also makes use of atomic-valued property features. These features describe linguistic properties of an instance or a clause. HALogen does not require them to be specified in the input, but they may be specified in order to over-

```
(e1 / eat
    :subject (d1 / dog)
    :object (b1 / bone
                :premod (m1 / meaty))
    :adjunct (t1 / today))

(e2 / eat
    :agent (d2 / dog)
    :patient (b2 / bone
                 :premod (m2 / meaty))
    :temporal-locating (t2 / today))
```

Figure 2: Example Inputs

```
VERB
-----
:MOOD  infinitive, infinitive-to,
       imperative, present-participle,
       past-participle, indicative
:TENSE present, past
:PERSON s (3s), p (1s 1p 2s 2p 3p), all
:MODAL should, would, could, may, might,
       must, can, will
:TAXIS perfect, none
:ASPECT continuous, simple
:VOICE active, passive
:SUBJECT-POSITION default, post-aux,
                  post-vp
:PASSIVE-SUBJECT-ROLE  logical-object
         logical-dative logical-postmod
:DATIVE-POSITION  shifted, unshifted

NOUN
----
:CAT1 common, proper, pronoun, cardinal
:NUMBER singular, plural

ADJECTIVE OR ADVERB
-------------------
:CAT1 comparative, superlative,
      negative ("not")

GENERAL
--------
:LEX (root form of a word as a string)
:CAT open class: vv, nn, jj, rb
     closed class: cc, dt, pdt, in, to,
              rp, sym, wdt, wp, wrb, uh
     punctuation: same as Penn Treebank
```

Figure 3: Property features used in HALogen

ride the defaults the system provides. The main properties that HALogen recognizes are listed in Figure 3, together with their possible values.

2.2 Symbolic Generator

An input is first processed by the symbolic generator. The symbolic generator consists of a set of about 255 mapping rules that transform an input into a packed set of possible expressions, referred to as a forest. A forest is a non-recursive context-free grammar. The left-hand-side of a mapping rule specifies the conditions for matching, such as the presence of a particular feature at the top-level of the input. The right-hand-side lists one or more outcomes. There are four kinds of mapping rules, recasting, ordering, filling, and morphing.

Recasting rules map one relation to another. They are used to map semantic relations into syntactic ones, such as :agent into :subject or :object, for example. They make it possible to localize constraints. As a result, the rule set as a whole is more modular and concise. Recasting rules facilitate a continuum of abstraction levels from which an application can choose to express an input. Recasting rules are also a tool that an application can use to customize HALogen, if desired. Recasting can be used to map non-linguistic or domain-specific relations into those already recognized by HALogen. Langkilde and Knight (1998a) describe the recasting technique in greater detail.

Ordering rules assign a linear order to the values whose features matched with the rule. Ordering rules typically match with syntactic features at the lowest level of abstraction. An ordering rule splits an input apart into several pieces. The values of the features that matched with the rule are extracted from the input and independently recirculated through the mapping rules. The remaining portion of the original input continues to circulate through the rules where it left off. When each of the pieces finishes circulating through the rules, a new forest node is created that composes the results in the designated linear order.

A filling rule adds missing information to underspecified inputs. This type of rule tests whether a particular feature is absent. If so, it will generate one or more copies of the input, one for each possible value of the feature, and add the feature-value pair to the copy. Each copy is then independently circulated through the mapping rules.

A morph rule produces a morphological inflection of a base lexeme, based on the property features associated with it.

The symbolic generator proceeds by comparing the top level of an input with each of the mapping rules in turn. The mapping rules decompose the input and recursively process the nested levels. Base input fragments are converted into elementary forests and then recombined according to the mapping rules. The final resulting forest is then processed by the statistical ranker.

HALogen's mapping rules are hand-written, but developed in part by using the *tgrep* tree grep program distributed with the Treebank. The tgrep program indexes all sections of Treebank, including section 23, the section used for testing in the experiments described later. However, the tgrep program does not indicate the section number of any trees that are retrieved, and appears to display matching trees in sectional order (ie., those from section 1 first). So the use of the test section in development of the mapping rules has been at most indirect.

2.3 Statistical Ranker

The forest ranker applies a bottom-up dynamic programming algorithm to extract the N most likely phrases from a forest. It uses an ngram language model built using Version 2 of the CMU Statistical Language Modeling Toolkit (Clarkson and Rosenfeld, 1997). Unigram, bigram and trigram models are all available. They are trained on 250 million words of Wall Street Journal newspaper text, excluding text from 1989 (from which the Penn Treebank is derived). The ranker finds an optimal solution with respect to the language model. It is described in greater detail in (Langkilde, 2000).

2.4 Comparison to Nitrogen

While Nitrogen recognizes a few syntactic relations, its focus is on semantic and other more abstract relations. HALogen, on the other hand, adds a full set of deep and shallow syntactic features intended to achieve extensive coverage of English syntax. Semantic rules from Nitrogen were modified in HALogen to map to syntactic ones rather than directly specifying an order on constituents. The syntactic features not only facilitate methodical, thorough coverage of English syntax, they also enable an application to precisely control the desired output, if desired.

HALogen handles adjunct relations, both semantic and syntactic, much better than Nitrogen. In HALogen, semantic adjuncts such as :spatial-locating and :temporal-locating can occur multiple times at a given level of nesting in an input, rather than being artificially restricted to just one each per node. HALogen also offers both the capability to try all possible order permutations of adjuncts, as well as the ability to impose partial order constraints on them. Applications concerned with rhetorical structure or coherence across sentences in multi-sentence generation need to have this kind of control. In contrast, Nitrogen arbitrarily assigns a single fixed order to adjuncts according to the order of the respective semantic relations in the set of mapping rules.

Nitrogen uses categorial grammar notations in the mapping rules to constrain generation output. However, HALogen abandons this because it has proved overly restrictive in scaling up to broad coverage. HALogen relies more heavily on the statistical ranker to implement grammatical preferences. When constraints are imposed, it is done by adding features to an input using the recasting mechanism. HALogen uses a feature named :type to impose three gross category constraints on phrases: verbal, nominal, or other. The symbolic processing engine checks the consistency and soundness of each input after recasting.

Other improvements over Nitrogen, though there is not space here to fully describe them, include:

- In input:
  - meta *OR* nodes possible at every level of nesting, not just top level, to specify disjunctions of values
  - compound values permitted for instance relation, to represent scope or influence constituent ordering
  - template-like capability using :template and :filler roles with labels
- Efficiency: improved cache and rule matching procedure
- Weights possible in grammar rules, input, and for concept-to-word mappings
- Polished output

3 Experimental Setup

The goal of an empirical evaluation of coverage and quality is to measure the extent to which any and every valid English sentence can be represented and generated. Generation is usually notoriously difficult to evaluate because grammaticality is difficult to measure automatically, and more than one output can be acceptable. However, although variations in output are usually acceptable in the context of specific applications, different applications can have different constraints on the kinds of variation in output that they accept. By demonstrating the capability to produce any desired sentence exactly, a system can assure that all possible application constraints on the output can be met. Thus, these experiments focus on whether a desired sentence can be produced exactly, though this kind of measurement is harsher than necessary.

Section 23 of the Penn Treebank is used to evaluate coverage and quality. Inputs to HALogen were automatically constructed from the Treebank annotation and then regenerated by the system. The output was then compared to the original sentence.

The Penn Treebank offers several advantages as a test set. It contains real-world sentences, it is large, and can be assumed to exhibit a very broad array of syntactic phenomena. It is not biased towards system-specific capabilities, since it was collected independently. It also acts as a standard for linguistic representation, offering the potential of interoperability with other natural language programs based on it, such as parsers. At the same time, there are limits to its usefulness. It only represents the domain of newspaper text, and thus does not test the stylistic, structural, and content variations that can occur in other domains such as question answering or dialogue. It also does not evaluate how the system handles nonsensical inputs or inputs that might not be expressible in grammatical English.

The input construction process involved finding the root forms of words, factoring Treebank categories of open class words into more basic features, heuristically designating constituent heads, inferring syntactic and logical roles for each node, making coordination bracketing more explicit, reorganizing compound prepositions into a single constituent, associating punctuation with a content-bearing constituent, flattening VP's, flattening nodes with only one child, removing null elements, and dropping some function words (ex: simple dative 'to', 'of'; benefactive 'for'; logical-subject 'by', auxiliary verbs, and some punctuation). The resulting structure is a hierarchical functional dependency tree.

The generator's primary tasks in this evaluation are to determine the linear order of constituents, perform morphological inflections, and insert needed function words. Six experiments were run to evaluate HALogen's performance with inputs that were underspecified in different ways. The ability to handle underspecification eases the information burden on client applications. It also makes the generator more flexible in meeting the varying needs and constraints of different types of applications.

```
:LOGICAL-SUBJECT :INSTANCE    :ANCHOR
:LOGICAL-OBJECT  :POLARITY    :TOPIC
:LOGICAL-DATIVE  :ADJUNCT     :PREMOD
:CLOSELY-RELATED :WITHINMOD   :POSTMOD
:BENEFACTIVE     :INTROCONJ   :CONJ
:PREDICATE       :LEFTPUNC    :PUNC
:DETERMINER      :RIGHTPUNC   :PREDET
```

Figure 4: Relations used in experiments

The experiments use only a subset of the relations that HALogen actually recognizes. Specifically, they use the mix of deep and shallow syntactic relations shown in Figure 4. No semantic relations are used, since they either can not be straight-forwardly derived from the Penn treebank annotation, or are too ambiguous to be adequately handled on the scale of the experiments in this paper.

In the first experiment, labeled "Almost fully spec" in Figure 7, the inputs contain nearly enough detail to fully determine a unique output. The inputs contain as much detail as it was possible to straight-forwardly obtain from the Treebank annotation. An example is shown in Figure 5. In this experiment, adjuncts are represented either as premodifiers, postmodifiers or within-modifiers. (Within-modifiers are verbal modifiers that come between the subject and the object). A flag in the generator is set so that constituents with the same role occurring at the same level of nesting (such as modifiers) will be ordered in the output in the same relative order in which they appear in the input (and adjacent to each other). This order flag allows applications that plan discourse structure before doing sentence realization to control the coherence across sentences. For example, dialogue systems often want old or background information to appear before new information. (Partial order constraints can also be specified by using extra levels of nesting in the input. However, this capability is not exercised in these experiments.)

In the second experiment, "Permute same-roles," the permutation flag set in the first experiment is reversed. Constituents with the same role are permuted in place, and the statistical ranker is expected to choose the most likely order. An exception occurs if there happen to be more than five constituents with the same role. For computational reasons, the constituents are not permuted in this case. Instead, they are placed in reverse order in the output (to avoid unfairly inflating the accuracy results). Everything else remains the same as in the first experiment.

```
(H34911
  :MOOD INDICATIVE
  :PREMOD (H34876 :CAT RB :CAT1 COMPARATIVE :LEX "earlier")
  :LOGICAL-SUBJECT (H34879 :DET (H34877 :CAT DT :LEX "the")
                     / (H34878 :DET NONE :CAT NN :CAT1 COMMON :NUMBER SING
                         :LEX "company"))
  / (H34880 :CAT VV :TENSE PAST :LEX "announce")
  :POSTMOD  . . .
  :PUNC PERIOD)
```

BIGRAM and TRIGRAM: Earlier the company announced it would sell its aging fleet of Boeing Co. 707s because of increasing maintenance costs.
ORIGINAL: same as above

Figure 5: Fragment of an almost fully specified input, and its output

```
(H37 :ADJUNCT "earlier"
     :LOGICAL-SUBJECT (H5 / "company")
     / "announce"
     :ADJUNCT . . .
     :PUNC PERIOD)
```

BIGRAM: It would sell its fleet age of Boeing Co. 707s because of maintenance costs increase the company announced earlier.
TRIGRAM: The company earlier announced it would sell its fleet age of Boeing Co. 707s because of the increase maintenance costs.
ORIGINAL: see Figure 5

Figure 6: Fragment of a minimally specified input, and its output

The third experiment, "Permute, no dir" is like the second, but in addition, all modifiers are mapped to the :adjunct relation, thus increasing the number of constituents that get permuted. The statistical ranker must not only determine the order of the modifiers with respect to each other, but must determine the direction of each one with respect to the head.

The fourth experiment, "Underspec det", is like the first except that common determiners are left unspecified. Specifically, "the", "a", "an", "any", and "some" are dropped from the input. The null-determiner feature is also dropped from all nominal phrases that had no determiner in the original Treebank annotation. This experiment tests the ability of the generator to supply the appropriate determiner, or figure out that none is needed.

The fifth experiment, "No leaf, clause feats", is also like the first experiment except that all the leaf and clause properties listed in Figure 3 are dropped from the input. Only the value of the :lex feature is retained for the input.

The sixth experiment, "Min spec," represents the opposite extreme from the first experiment. All the information dropped in experiments 2-5 is also dropped in this experiment. An example of such an input and its output is shown in Figure 6.

For computational reasons, a bigram model, not trigram, was applied by the ranker for these experiments. However, for the sake of comparison, Figures 5 and 6 show the output from both the bigram and trigram models.

4 Results

Results of the experiments are shown in Figure 7. Section 23 of the Penn Treebank contains 2416 sentences. The average Penn sentence consisted of 23.5 tokens–the shortest had two, and the longest 66. The input construction tool produced inputs from 98% of the Penn sentences, or 2377 inputs. HALogen produced output for approximately 80% of the inputs. Assuming the test set is representative of English, and coverage is defined as the percent of syntactic constructions (ie., inputs) for which the generator produces correlative output, then we can estimate HALogen's coverage of English at about 80%.

We applied three different metrics to evaluate quality, the IBM Bleu score (Papineni et al., 2001), the average NIST simple string accuracy, and exact match. The IBM Bleu score is a geometric average of n-gram accuracy with respect to the original Penn sentence, adjusted by a length penalty factor LP. Namely, BLEU $= \exp(\sum_{n=1}^{N} w_n \log p_n) \times$ LP.

Input characteristics:	Almost fully spec	Permute same-roles	Permute, no dir	Under-spec det	No leaf, clause feats	Min spec
Median num of sen gen by an input:	72	576	2e+5	7e+6	9e+9	4e+16
Smallest num of sen gen by an input:	1	1	1	2	2	4
Max num of sen gen by an input:	2e+10	1e+14	8e+19	9e+25	2e+32	2e+53
Average ranking time per input (secs):	0.013	0.023	0.226	0.036	0.207	18.3
Average total time per input (secs):	28.9	27.1	29.8	28.4	30.2	55.5
Average length of gen sentences:	22.4	22.4	22.4	21.9	21.7	21.0
Average length of exact matches:	20.9	18.8	17.0	16.4	16.0	11.5
Num of inputs that produced output:	1968	1968	1966	1981	1812	1884
Num of exact matches:	1132	807	555	391	299	98
Percent inputs that produced output:	82.8%	82.8%	82.7%	83.3%	76.2%	79.3%
Percent exact matches in output:	57.5%	41.0%	28.2%	19.7%	16.5%	5.2%
Ave. NIST simple string accuracy:	94.5%	81.6%	69.6%	85.1%	81.1%	55.3%
IBM Bleu score:	0.924	0.826	0.757	0.776	0.717	0.514

Figure 7: Experimental Results

LP $= \exp(1 - r/c)$ if $c \leq r$, and LP $= 1$ if $c > r$, where $w_n = 1/N$, $N = 4$, c is the system output length, and r is the reference length.

The average NIST simple string accuracy score reflects the average number of insertion (*I*), deletion (*D*), and substitution (*S*) errors between the output sentence and the original Penn sentence. Formally, SSA $= 1 - (I + D + S)/R$, where R is the number of tokens in the original sentence.

The Bleu and NIST metrics agree fairly closely with each other in all the experiments. Using these metrics, HALogen's output ranged from about 93% correct when inputs were almost fully specified, to 53% correct with minimally specified inputs. Almost 58% of the outputs were exact matches in the first experiment, dropping to 5% in the sixth. Although the quality in the sixth experiment is substantially worse than that of the first, it requires much less information in the input, and thus is much easier for a client application to produce. For applications like machine translation that can tolerate imperfect output but have difficulty supplying detailed inputs, HALogen can be an appealing tool.

The second and third experiments show that permuting same-role nodes does not have as big an impact on quality as one might expect. This probably reflects the ngram model's strength in capturing order information, especially for single-word modifiers. It may also reflect the relative infrequence of nodes having multiple modifiers. In the fourth experiment, HALogen doesn't do as well at selecting determiners as one might expect. The divergence in this experiment between the exact match metric and the other metrics probably results from the need to choose determiners in nearly every sentence, while determiners constitute only a fraction of the words in a sentence. The fifth experiment shows the largest drop in accuracy compared to the second through fourth, suggesting that this problem is harder than the others. However, HALogen still achieves about 75% correct on this very frequent problem.

One topic of interest is the causes of generation failure, and the causes of inexact matches in the first experiment. One such cause is syntactic phenomena that are known to not be handled appropriately yet. This includes right-node-raising, dislocated constituents, discontinuous constituents, and headless constituents. Another important cause is errors and inconsistencies in the original Treebank annotation. The automatic input construction process also introduces some errors, particularly in heuristically selecting constituent heads. At other times, HALogen deliberately fails to generate an input because the phrase itself is malformed.

The only other system to have done a similar empirical evaluation is FERGUS. FERGUS applied a statistical tree model, TAG grammar, and ngram language model to 100 inputs consisting of a complete dependency tree labeled with fully inflected words but no roles. The realization problem was limited to determining constituent order. FERGUS achieved 58% correct using the NIST metric, a result comparable to that of the "Min spec" experiment just described. In contrast to HALogen, FERGUS offers no means for controlling the generation of a specific output exactly. On the other hand, it seems likely that a tree model such as the one used by FERGUS could improve HALogen's accuracy. See (Daume et al., 2002) for evidence of this. Such work is currently under way for HALogen.

5 Summary

In conclusion, we empirically verified HALogen's coverage and accuracy using section 23 of the Penn Treebank as a test set. On a set of 2400 automatically-derived inputs, 80% produced output that was about 94% correct when the input was almost fully specified. Accuracy was measured using IBM Bleu scores and NIST simple string accuracy scores which compared outputs to the original sentences. About 57% of the outputs were exact matches with the original. Using minimally specified inputs, accuracy was still more than 51%, 5% of which were exact matches. The flexibility that HALogen offers in the degree of specification of the input adds to its general-purposeness. Tasks like dialogue that require high quality output and need to exert significant control over the output can do so if they wish, while tasks like translation, that generally suffer from an inability to provide much information but that can accept lower quality output, still obtain output without providing many specification details.

In future work, we plan to apply a statistical model of syntax, continue to broaden syntactic coverage, extend the system to handle additional subtasks of realization such as pronomialization and ellipsis, and evaluate the system in the context of different applications.

References

S. Bangalore, O. Rambow, and S. Whittaker. 2000. Evaluation metrics for generation. In *Proc. of 1st INLG*.

J. Bateman. 1996. KPML development environment — multilingual linguistic resource development and sentence generation. Technical report, German Centre for Information Technology (GMD).

P.R. Clarkson and R. Rosenfeld. 1997. Statistical language modeling using the cmu-cambridge toolkit. In *Proc. ESCA Eurospeech*.

H. Daume, K. Knight, I. Langkilde-Geary, D. Marcu, and K. Yamada. 2002. Experiments using a statistical model of syntax. In *submitted*.

M. Elhadad and J. Robin. 1998. Surge: a comprehensive plug-in syntactic realization component for text generation. In *http://www.cs.bgu.ac.il/ elhadad/pub.html, submitted*.

M. Elhadad. 1993. FUF: The universal unifier—user manual, version 5.2. Technical Report CUCS-038-91, Columbia University.

J.Robin. 1994. *Revision-based generation of natural language summaries providing historical background: corpus-based analysis, design, implementation and evaluation.* Ph.D. thesis, Columbia University.

K. Knight and V. Hatzivassiloglou. 1995. Two-level, many-paths generation. In *Proc. ACL*.

K. Knight, I. Chander, M. Haines, V. Hatzivassiloglou, E. Hovy, M. Iida, S. K. Luk, R. Whitney, and K. Yamada. 1995. Filling knowledge gaps in a broad-coverage MT system. In *Proc. IJCAI*.

I. Langkilde and K. Knight. 1998a. Generation that exploits corpus-based statistical knowledge. In *Proc. COLING-ACL*.

I. Langkilde and K. Knight. 1998b. The practical value of n-grams in generation. In *Proc. International Natural Language Generation Workshop*.

I. Langkilde. 2000. Forest-based statistical sentence generation. In *Proc. NAACL*.

B. Lavoie and O. Rambow. 1997. RealPro – a fast, portable sentence realizer. In *ANLP'97*.

M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of english: the Penn treebank. *Computational Linguistics*, 19(2).

M. Meteer. 1990. *The Generation Gap - the problem of expressibility in text planning*. Ph.D. thesis, U. of Massachusetts.

K. Papineni, S. Roukos, T. Ward, and W-J. Zhu. 2001. Bleu: a method for automatic evaluation of machine translation. Technical Report RC22176, IBM Research Division.