Fluency and Completeness in Instance-based Natural Language Generation

Sebastian Varges

LTG and ICCS

varges@cogsci.ed.ac.uk

Division of Informatics

University of Edinburgh

Abstract

A fundamental assumption underlying candidate ranking in corpus-based approaches to Natural Language Generation is the idea that in order to be fluent the output should be as similar to a (human-authored) corpus as possible. However, the goal of maximizing fluency can conflict with other goals, like conveying the maximal amount of input and being faithful. We employ an instance-based sentence generation system to investigate how the right balance between the different goals can be struck and show empirical results supporting our proposals.

1 Introduction

In recent years, ranking approaches to Natural Language Generation (NLG) have become increasingly popular. They abandon the idea of generation as a deterministic decision-making process in favour of approaches that combine overgeneration with ranking at some stage in processing. A major motivation is the potential reduction of manual development costs and increased flexibility and robustness.

Several approaches use ranking models trained on corpora of human-authored texts to judge the *fluency*[1] of the candidates produced by the generation system. The seminal work of (Langkilde and Knight, 1998) presents a sentence realizer that uses ngram models trained on a large corpus to rank candidates. Ratnaparkhi (2000) presents a sentence realizer that has been trained on a domain-specific corpus augmented with semantic attribute-value pairs. The goal of the system is to learn attribute ordering and lexical choice. Bangalore and Rambow (2000) describe a realizer that uses an ngram model combined with a tree-based stochastic model. Ranking techniques have also been explored in more restricted areas like determining the ordering of NP premodifiers (Shaw and Hatzivassiloglou, 1999).

In this paper, we investigate how to deal with situations in which the goal of maximizing fluency

---

[1] A generated sentence is assumed to be 'fluent' if it is likely to belong to a chosen training corpus (corresponding to the 'prior probability' in Statistical Machine Translation).

conflicts with other goals like conveying the maximal amount of input – *completeness* according to (Wedekind, 1988) – and being faithful. A trade-off between these goals may be necessary if candidates need to be compared that express different parts of the input. This in turn may be due to the fact that the grammar is not able to produce candidates that convey the entire input, and even if there are such candidates, there may be other ones that express slightly less input but are more fluent.

This paper is organized as follows: first, we present Instance-based Natural Language Generation, giving examples from an implemented system. In section 3, we describe the tension between fluency and completeness, propose two solutions and discuss under which circumstances they are applicable. Section 4 discusses how problems of lack of faithfulness can arise from the omission of parts of the generation input and shows two techniques that remedy the problem. In section 5, we show empirical results supporting our proposed solutions. Section 6 provides further discussion, and section 7 concludes the paper.

2 Instance-based NLG

The Instance-based Natural Language Generation approach of (Varges and Mellish, 2001) employs two basic components: a rule system that generates a number of possible realization candidates from a meaning representation and an instance-based ranker that uses examples taken from a training corpus to score the candidates. In general, it is assumed that the highest ranked candidate is the output of the generator. It is also assumed that the generator's goal is to produce a single sentence.

The input to the generator is a set of attribute-value pairs. An example:

(1)

0	IN_FULLNAME	Bruno DeGol
1	IN_OTHERPOST_NODET	chairman
2	IN_OTHERCOMP	DeGol Brothers Lumber
3	IN_OTHERCOMP_LOC	Gallitzin Pa
4	POST_INDEF	director
5	COMP_DESCR	bank-holding company
6	BOARD_INCR	11

```
NP-IN_FULLNAME     -->     IN_FULLNAME                      (Bruno DeGol)
NP-POST_INDEF      -->     a POST_INDEF                     (director)
PP-COMP_DESCR      -->     of this COMP_DESCR               (bank-holding company)
S-BOARD_INCR       -->     expanding the board to BOARD_INCR (11)
...
Sup-IN_FULLNAME    -->     NP-IN_FULLNAME  VP-POST_INDEF .
NP-IN_FULLNAME     -->     NP-IN_FULLNAME  ,  NP-IN_OTHERPOST_NODET  ,
VP-POST_INDEF      -->     was named S-POST_INDEF  ,  S-BOARD_INCR
S-POST_INDEF       -->     NP-POST_INDEF  PP-COMP_DESCR
...
```

Figure 1: Input rules (top) and phrasal rules (bottom) for treebank sentence (slot fillers in brackets)

This attribute-value representation is part of the semantic markup that has been manually applied to a corpus of texts on management successions as they can be found in the *Who's News* section of the *Wall Street Journal*. The original corpus sentence is part of the Penn treebank II:

(2) Bruno DeGol, chairman of DeGol Brothers Lumber, Gallitzin, Pa., was named a director of this bank-holding company, expanding the board to 11 members.

The semantic markup is based on a flat annotation scheme that assigns semantic roles to data that is expected to be the input to the generator. This implies that content selection has already taken place. The level of semantic representation is similar to that of many Information Extraction applications. Tags (used as attributes in the generation input) can only mark contiguous strings which are used as 'values' or 'slot fillers' in the generation input. Determiners are not part of the marked string and therefore are not expected to be part of the slot fillers. However, they are indicated by the annotation tags (eg.: IN_OTHERPOST_NODET), allowing the system to require certain determiners even if they are not adjacent to the slot filler (e.g.: a [IN_AGE 59]-year-old [IN_OTHERPOST_INDEF director]).

The treebank structure for sentence (2), in combination with the semantic tags is used to automatically derive a context-free grammar (see (Varges and Mellish, 2001) for a description of the grammar construction algorithm). Some of the rules extracted are shown in figure 1. There are two different kinds of rules: 'input rules' that consume input tags directly, and 'phrasal rules' that combine the result of input rules and other phrasal rules. Categories of rule left-hand sides are formed by concatenating the syntactic category of the treebank parse with the first tag found on the rule right-hand sides.

We divided the annotated treebank of 144 sentences into training and test set. For a training set of 104 annotated treebank sentences, the system constructs 155 different input rules and 240 different phrasal rules. These are expressed as productions in a production system which organizes them into a Rete network (Forgy, 1982). Compared to standard chart algorithms, Rete networks have the advantage of avoiding repeated matches when rules have some right-hand side elements in common. This is particularly useful when large numbers of rules are derived automatically. Our Rete-based generator basically works like a bottom-up chart generator except that active edges (partially activated productions) are not part of the chart but rather part of the Rete network.

When input (1) is given to a rule system containing the rules in figure 1, the rule system is able to produce the sentence in (2) as a template into which the slot fillers can be inserted:

(3) IN_FULLNAME , IN_OTHERPOST_NODET of IN_OTHERCOMP , IN_OTHERCOMP_LOC , was named a POST_INDEF of this COMP_DESCR , expanding the board to BOARD_INCR members .

The grammar interpreter is able to generate 9713 candidates overall for input (1). Many of these have errors and are expected to be ranked low.

The training set sentences are not only used for grammar rule construction but also as nearest-neighbours in the instance-based ranker. To this end, candidates as well as instances, i.e. annotated sentences of the training set, are represented as ngrams derived from template-like representations as in (3), yielding trigrams like {"named a POST_INDEF', 'a POST_INDEF of'}, for example. As a result of this choice of representation, only tags and words outside tagged areas are visible to the ranker. Slot fillers do not influence the result of ranking. We use trigrams rather than bigram terms since the context provided by bigrams seems too limited. This is because punctuation is treated just like words, resulting in a contextual 'boundary' for bigram terms (a bigram-based ranker has no means to relate words to the left and the right of a comma, for example).

Pseudo-document vectors are constructed for the trigram terms derived from both training set instances (at compile-time) and edges (at run-time), and the score of an edge with respect to some instance is determined by the cosine distance between the two vectors. This is a common distance metric in Information Retrieval. The ngram terms are weighted according to the $tf \cdot idf$ term weighting scheme. A template like (3), if generated, is a per-

fect match for the corresponding instance, yielding a maximal cosine score of 1.0. Depending on the number and weight of matching terms in relation to non-matching terms, many similarity scores will be lower.

Candidate production and ranking are interleaved so that the search for candidates can be restricted as soon as a first candidate has been produced. (We treat all edges of syntactic category S as candidates for the moment.) This is done by employing an A^*-style search algorithm which computes the best possible score (the *expectation*) for all edges. The expectation is the score that an edge may obtain if it is combined with other edges that contain exactly those terms that are still missing in the edge w.r.t. to a specific instance. Therefore, there are generally as many expectations for some edge as there are instances in the instance base. If an edge has no expectation that is greater than the actual cosine score of an already existing candidate, it can safely be dropped since it will never become a better sentence than has been obtained already. Thus, the score of already produced candidates serves as a threshold for the expectation of edges. Search is most efficient if the expectation is also used to determine the agenda ordering.

3 Fluency and completeness

A fundamental assumption in instance-based NLG is the idea that in order to be fluent the output should be as similar to human-authored corpus sentences as possible. Consider the following example sentence taken from the test set and the corresponding input:

(4) James L. Pate, 54-year-old executive vice president, was named a director of this oil concern, expanding the board to 14 members.

0	IN_FULLNAME	James L Pate
1	IN_AGE	54
2	IN_OTHERPOST_NODET	executive vice president
3	POST_INDEF	director
4	COMP_DESCR	oil concern
5	BOARD_INCR	14

From input (4) the grammar rules produce three candidates that have a cosine score of 1.0 with some instance of the training set, i.e. those instances have exactly the same template representation as one of the candidates:

(5) James L Pate was named a director of this oil concern . (coverage 3/6)

(6) James L Pate was elected a director of this oil concern , increasing the number of seats to 14 . (coverage 4/6)

(7) James L Pate , 54 years old , was elected a director of this oil concern , expanding the board to 14 members . (coverage 5/6)

Overall, 7074 candidates can be produced for input (4) including those that have a lower rank. A^*-based pruning keeps this number down to 11. Candidate (5) realizes half the input, (6) realizes 4 out of 6 attribute-value pairs and (7) 5 out of 6. Thus, the candidates exhibit a lack of completeness with respect to the input: The instance-based ranker tends to choose small subsets of the input semantics that have been expressed by some training set instance. In principle, being similar to the corpus sentences is a desirable property and a general characteristic of corpus-based approaches – the goal is to mimic some aspects of the corpus. However, only maximizing this similarity can be a problem. Candidates (5-7) could in fact be generated by a system that simply has a complex template corresponding to each instance. In other words, the creativity of the rule-based grammar has not been exploited by the instance-based generator. The grammar rules might be able to generate previously unseen candidates that better express the input. However, these are not preferred by the ranker. (Furthermore, if grammar interpreter and ranker are interleaved as in our approach, one tends to end up with only very short candidates that have a high score since these can be used by the A^*-search algorithm to prune away longer but lower scoring candidates.)

One obvious first attempt to address this problem is to extend the definition of a 'candidate' from any edge of syntactic category S to also require the complete realization of the input. This yields the following best candidate for input (4) (tags are shown as well):

(8) [IN_FULLNAME James L Pate], [IN_AGE 54] years old, [IN_OTHERPOST_NODET executive vice president], was elected a [POST_INDEF director] of this [COMP_DESCR oil concern], expanding the board to [BOARD_INCR 14] members .

This sentence has a cosine score of 0.95 and has therefore previously lost out against its shorter but higher scoring competitors (5-7). However, it cannot generally be assumed that the generator is always able to express the entire input semantics in one sentence although in this case we were lucky. For example, there are 5 tags in the test set that do not occur in the training set (overall, there are 74 different tags in the corpus annotation). Furthermore, there is no guarantee that a given combination of tags known to the grammar can be expressed in a single sentence. This is part of a general problem that has been described as the "generation gap" (Meteer, 1990). The fundamental question is how a generation input can be constructed by some external component without knowing exactly what the grammar is able to express.

3.1 Combining cosine score and coverage

Here, we are not trying to decide whether or not the generation gap can be bridged but rather address the practical question how we can influence the coverage of the input semantics in an instance-based approach to candidate ranking without hard-wiring a completeness constraint. Our first proposal is to use a linear combination of the cosine score and a coverage score to compute a new edge score:

$$(9) \quad (1 - \lambda) \cdot cos(E, I) + \lambda \cdot \frac{|Sem_E|}{|Sem_{Input}|} \quad [\lambda \in \{0...1\}]$$

The cosine between edge E and some instance I (to the left of 9) is combined with a coverage score which is computed by dividing the size of the semantics of E by the size of the input.[2] In principle, the coverage score can take into account different weights for parts of the semantic input. A high value of λ places more emphasis on completeness, a low value yields candidates that more closely resemble instances.

Choosing different values for λ results in candidates of different sentence lengths and input coverages. Generating the complete candidate (8) for input (4) requires a λ of at least 0.2. The nearest-neighbour of (8) is a training set template that lacks the IN_OTHERPOST_NODET tag:

(10) [IN_FULLNAME William C. Ballard Jr.], [IN_AGE 48] years old, was elected a [POST_INDEF director] of this [COMP_DESCR distilled beverages concern], expanding the board to [BOARD_INCR 11] members.

The linear combination of cosine score and coverage score effectively 'squeezes' the missing IN_OTHERPOST_NODET tag into the nearest-neighbour template.[3]

There are always certain values of λ at which the system 'switches' between different best candidates. One can only obtain short candidates (5,6) if λ=0.0. In our experience, the switching points and number of best candidates for different settings of λ can vary greatly for different inputs. For example, the system finds only two best candidates for a longer input of 10 tags:

(11) λ >= 0.05: Michael Henderson , 51 -year-old group chief executive , was named chairman of this metals and industrial materials maker , succeeding Ian Butler , 64 . (coverage: 7/10)

(12) λ < 0.05: Michael Henderson was named chairman of this metals and industrial materials maker . (coverage: 3/10)

---

[2] When we talk about the 'length' of an edge in this paper, we are referring to the size of the semantic set. This broadly correlates with the length of the surface string (see section 5).

[3] However, it should be noted that the system does not directly adapt templates to new inputs but rather uses grammar rules derived from an analysis of the instances to produce the output candidates.

One cannot know ahead of time how many different candidates there are for all values of λ, where the switching points are, or whether increasing λ would result in larger coverage.

On the other hand, it seems possible to choose a 'reasonable' setting for λ (0.2, for example) if the goal is simply to express as much input as possible without compromising fluency (i.e. similarity between candidate and nearest-neighbour) too much. In fact, the use of a linear combination suggests that there is a trade-off between fluency and completeness in the sense that the cosine decreases as the coverage weight λ increases. This is indeed what we find (see section 5).

3.2 Generating candidates into several ranked lists

We would like to take a step back at this point and ask what we actually expect the output of a realizer to be in the context of an overgeneration approach. For text generation, or any input for which we are unsure whether it can really be expressed in a single sentence, there is a possibility that we do not want to express a maximal number of tags in the first sentence since we do not know whether the remaining semantics can be expressed in the follow-up sentences. We would rather aim at generating a globally best text than a locally best first sentence.

We therefore propose that a realizer used in the context of a text generation system produce several ranked candidate lists, each one for a different subset of the input semantics. Follow-up sentences would be generated for the remaining semantics of these candidate lists which again return a set of ranked lists. The ranked lists could be organized in a lattice or similar data structure. The text generator would then choose a path though this data structure.

In this paper, we concentrate on sentence realization but believe that maintaining several dynamically created candidate lists for different semantic indices are a useful output of a sentence realizer if it is unclear how much semantics should be expressed in the individual sentences.

As an example, we take again input (4). The system dynamically creates 12 ranked lists ('bins') according to the semantic indices of the candidates that have been produced (the indices can be found in first column of input 4). In addition to bins that contain (5-7) as their best candidate, the following sentences are the best candidates in their respective bin:

(13) sem=0 2 3 4 5: James L Pate, executive vice president, was elected a director of this oil concern, increasing the number of seats to 14 . (cos=0.96, cov=5/6)

(14) sem=0 1 2 3 4 5: James L Pate, 54 years old, executive vice president, was elected a director of this oil concern, expanding the board to 14 members . (cos=0.95, cov=6/6)

(15) sem=0 1 3 4: James L Pate, 54 years old, was named a director of this oil concern . (cos=0.73, cov=4/6)

(16) sem=0 2 3 4: James L Pate, executive vice president, was named a director of this oil concern . (cos=0.70, cov=4/6)

(17) sem=0 1 2 3 4: James L Pate, 54 years old, executive vice president, was named a director of this oil concern . (cos=0.61, cov=5/6)

(18) sem=0 1 3: James L Pate, 54, was elected a director . (cos=0.47, cov=3/6)

(19) sem=0 1 2 3: James L Pate, executive vice president, 54, was elected a director . (cos=0.43, cov=4/6)

(20) sem=0 2 3: James L Pate, executive vice president, was elected a director . (cos=0.39, cov=3/6)

Behind every shown candidate there is a list of lower ranked candidates that express the same semantics (which are not shown). The number of bins depends on the size of the input and the combinations of tags that can be produced by the grammar. For example, the input of 10 tags that yields (11) and (12) results in 50 different bins.

Generating candidates into different bins also solves another problem of being too similar to the corpus: if there is a good match for a long candidate, the system might not be able to find a shorter, lower scoring one.

3.3 Effects on A^*-search algorithm

A linear combination of cosine and coverage score affects the computation of the expectation: we need to be optimistic about the coverage score (as we are about candidate-instance similarity) and assume that the expectation for the coverage score is always maximal (1.0). We have to make this assumption since we do not know what combinations of tags are licensed by the grammar – again a manifestation of the fundamental problem of expressibility. The combined expectation therefore is higher than the cosine alone, making it easier for edges to pass the threshold set by the cosine of already existing candidates. This extends the search space but does not lead to problems in practice except for extreme values of λ (see below).

The use of multiple rankings allows candidates only to compete within their bins. Since there are fewer candidates in each separate bin than overall, less pruning takes place. Again, this does not pose a serious practical problem.

4 Faithfulness problems

Realizing the generation input only partially can result in non-intended interpretations. An example of a faithfulness error due to the omission of certain tags:

(21) George L Manzanec, senior vice president, was named a group vice president of this natural-gas-pipeline concern.

(22) George L Manzanec, senior vice president of Texas Eastern Corp., was elected a group vice president of this natural-gas-pipeline concern .

Only the second realization is faithful to the input since Manzanec is senior vice president at another company, not the natural-gas-pipeline concern as (21) implies.

Solutions to this faithfulness problem need to consider that the only semantic information available to the system is the annotation of the corpus sentences. We do not have a deep semantic model that is able to reason about possible interpretations of candidates.

4.1 Cooccurrence constraints

One approach to address this problem is to manually specify hard cooccurrence constraints on semantic tags that effectively act as additional filters on candidates. For example, they can state that both IN_OTHERCOMP and IN_OTHERPOST_NODET need to be expressed together. Tag cooccurrence constraints can be applied to two or more tags which can be regarded as single tags that are allowed to be realized discontinuously.

An efficient implementation of tag cooccurrence constraints is possible by only checking for those semantic indices that are actually present in the input. The constraints are specialized to semantic indices when the input is given to generator. This means that constraints that do not apply (since not all tags that need to cooccur are present in the input) need never be checked during generation. For example, consider the following constraints:

```
coocurrenceCst(["IN_OTHERCOMP","IN_OTHERPOST_NODET"])
coocurrenceCst(["IN_OTHERCOMP","IN_OTHERPOST_DEF"])
coocurrenceCst(["INPERSON_PREVIOUSCOMP",
                "INPERSON_PREVIOUSPOST_NODET"])
```

When input (1) is given to the generator, only the first constraint is translated into requiring either both indices 1 and 2 or none of these. The other constraints do not apply, avoiding unnecessary overhead when individual candidates are checked.

4.2 Improving the annotation scheme

A related faithfulness problem can be traced back to an ambiguity in the annotation scheme. In the following example, the best candidate places the post of the incoming person at the main company:[4]

(23) William J Russo was named senior vice president public affairs and advertising, of this financial and travel services concern .

---

[4]The main company is assumed to have been introduced into the discourse context already, usually by a headline to the article about the management succession.

| λ | cosine av. | coverage | | | length (words) | | $|\bar{e}|$ | $\bar{t}$ secs | Fluency errors | Syntax errors | Sem. errors | correct cands. |
		av.	stdev	% compl.	av.	stdev						
0.0	0.97	0.63	0.19	10.0	16.8	4.2	1315	6.9	0	0	3	92.5 %
0.2	0.97	0.68	0.18	12.5	17.8	4.2	1338	8.8	0	0	3	92.5 %
0.4	0.93	0.75	0.18	25.0	19.9	5.7	1259	10.6	3	0	5	80.0 %
0.6	0.81	0.87	0.17	47.5	22.2	5.7	1338	11.8	10	2	5	62.5 %
0.8	0.69	0.92	0.13	70.0	23.8	5.3	1530	12.3	12	4	5	57.5 %
1.0	0.69	0.92	0.13	70.0	23.8	5.3	976	13.1	12	4	5	57.5 %

Figure 2: Evaluation results for instance-based generator using linear combination

However, the original sentence specifies that the post is located at a subsidiary (we only show the annotation of the relevant parts of the sentence):

(24) William J. Russo was named senior vice president, public affairs and advertising, for this [COMP_DESCR financial and travel services concern]'s [COMP_SUBSIDIARY American Express Bank Ltd.] [COMP_SUBSIDIARY_DESCR_NODET subsidiary].

The annotation does not clearly specify the location of the post, and in a sense the ranker does not choose an "unfaithful" candidate. The problem can be solved by marking the post location explicitly, for example by adding '*' to each tag specifying the location of the main post. If this is done consistently for the entire corpus, the system will be able to clearly distinguish between COMP_DESCR – as we will find it in the revised annotation of (24) – and COMP_DESCR* for corpus sentences like (2). This example demonstrates that we can influence the output of the generation system by changing the annotation that forms the basis of the ranker (and the grammar, of course).

5 Evaluation

We evaluated our system along several dimensions which can be divided into those that require human judgement, and those that do not. With respect to the linear combination, the trade-off between cosine and coverage as well as different candidate lengths (in terms of surface words and semantic tags) and basic efficiency metrics can be measured automatically. On the other hand, the reduction of faithfulness errors and other problems of the output need to be observed by human judges.

We divided the annotated corpus into a random selection of 104 articles for training and 40 for testing. Figure 2 shows average cosine and coverage values for 6 settings of λ. The results clearly indicate a trade-off between the two. As intended by the use of the linear combination, coverage and average sentence length in words grow as λ increases. For higher values of λ, the average candidate length approaches the one of the original test set sentences (average length in words: 26.1; standard deviation 7.4). The number of complete candidates ('% compl.') increases from 10% for $\lambda=0$ to 70% for $\lambda=0.8$.

Figure 2 also shows the results of the manual evaluation of the system output for which we employed two human judges. Semantic errors include faithfulness errors and other semantic problems. We distinguish syntax errors from fluency errors. Some examples:

(25) *Fluency:* Francis D. John, *35 years old, president,* was elected to ... .

(26) *Syntax:* This bank holding company said its board *elected* John W. Day.

(27) *Semantics:* ... was named vice president,..., *the new post/both new posts* ...

We observed an increase in the number of fluency and syntactic errors for lower average cosine scores – confirming our basic assumption that cosine similarity is a good indicator of fluency. As a result of tag cooccurrence constraints and improved annotation, the number of faithfulness errors is relatively small, even for short candidates. An unmodified version of the system yields 7 faithfulness errors due to omissions of tags for $\lambda=0.2$ (versus 3 faithfulness errors for the new system). The overall number of correct sentences is maximal for $\lambda=0.0$ and $\lambda=0.2$ and decreases when more weight is given to completeness. Although the number of test set inputs is small, the results seem to indicate the validity of our approach.

The column headed by $|\bar{e}|$ shows the average number of edges per test set input; column $\bar{t}$ shows the average run-time. Obviously, the latter is influenced by the programming language (Java) and the machines available (Sun Enterprise 220). For high values of λ (>0.8), the A^*-search algorithm tends to search for ever larger candidates so that we need to apply a time limit (60 seconds).

For reasons of space we cannot give detailed efficiency figures for generation into multiple ranked lists. The average number of bins for 4 input tags is 4.6, resulting in the generation of 133 candidates per input (without bins and $\lambda=0.6$, the system produces 137 candidates on average). The average number of bins for inputs of size 7 increases to 18.7, resulting in 469 candidates per input on average (in contrast to 473 for generation without bins and $\lambda=0.6$).

6 Discussion

Ranking models for NLG that are based on corpus similarity alone (even if the corpus contains additional syntactic or semantic information) make the assumption that all candidates are (equally) complete and (equally) faithful. If this assumption is relaxed – as we have done in this paper – the possibility of a trade-off between fluency and other goals arises. This does not depend on search strategy or generation architecture and also applies to systems in which grammar interpreter and ranker are non-interleaved.

In instance-based ranking, the tendency towards replicating training set sentences is easily observable because the original examples are available as nearest-neighbours which can be imitated directly. In statistical approaches, the corpus material is generally not accessible. However, in case statistical NLG systems do not ensure completeness, they can face similar problems of comparing candidates. For example, ngram models tend to prefer sequences of words that are much shorter than any corpus sentence. (Knight and Hatzivassiloglou, 1995) describe a function which increases with sentence length to counter this behaviour. Avoiding to minimise sentence length can be seen as an indirect way to approximate a completeness criterion.

The A^*-search algorithm can be extended so that n best candidates are found. This can be implemented by keeping the threshold as low as the score of the nth candidate in the ranked list, or at 0.0 as long as there are less than n candidates. Keeping n best candidates in each bin when generating into multiple ranked lists might be useful if macroscopic properties are scored in text generation (Oberlander and Brew, 2000). For example, one might want to approximate a certain global distribution of verbs in the text, and this might require the system to choose candidates that are non-optimal according to the sentence-based nearest-neighbour ranker. Similarly, issues of discourse coherence in multi-sentence generation may require a re-ranking of the locally best sentences. Alternatively, discourse coherence could be ensured by means of grammar constraints. In other words, additional knowledge needed to be added to either the ranker or the rule-based part of the hybrid generation system. However, this is beyond the scope of the present paper.

7 Conclusion

We have described the tension between the goals of fluency and completeness in Natural Language Generation and presented two techniques to deal with it. These were implemented in an instance-based generation system. Using a linear combination of cosine and coverage score allows one to trade candidate-instance similarity against candidate-input similar-ity (in semantic terms). Maintaining multiple candidate lists circumvents the problem of choosing a candidate that expresses the right portion of the input altogether by delaying the final decision until a larger context is known. We also demonstrated that faithfulness problems can arise if parts of the generation input are omitted, and proposed tag cooccurrence constraints as well as refining the annotation scheme as potential solutions.

References

Srinivas Bangalore and Owen Rambow. 2000. Exploiting a Probabilistic Hierarchical Model for Generation. In *COLING-00*.

Charles L. Forgy. 1982. Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem. *Artificial Intelligence*, pages 17–37.

Kevin Knight and Vasileios Hatzivassiloglou. 1995. Two-level, Many-paths Generation. In *ACL-95*.

Irene Langkilde and Kevin Knight. 1998. Generation that Exploits Corpus-based Statistical Knowledge. In *COLING/ACL-98*.

M. W. Meteer. 1990. *The Generation Gap – The Problem of Expressibility in Text-Planning*. Ph.D. thesis, University of Massachusetts, Amherst, MA, USA.

Jon Oberlander and Chris Brew. 2000. Stochastic Text Generation. In *Philosophical Transactions of the Royal Society of London, Series A*, volume 358, pages 1373–1385.

Adwait Ratnaparkhi. 2000. Trainable Methods for Surface Natural Language Generation. In *NAACL-00*.

James Shaw and Vasileios Hatzivassiloglou. 1999. Ordering among Premodifiers. In *Proceedings of ACL-99*.

Sebastian Varges and Chris Mellish. 2001. Instance-based Natural Language Generation. In *NAACL-01*.

Jürgen Wedekind. 1988. Generation as Structure Driven Derivation. In *COLING-88*.