An Architecture for Text Revision

Inui Kentaro, Tokunaga Takenobu, Tanaka Hozumi

Department of Computer Science, Tokyo Institute of Technology
2-12-1 Ôokayama Meguro Tokyo 152 Japan
{inui,take,tanaka}@cs.titech.ac.jp

Abstract

In order to generate good text, many kinds of factors should be taken into account. In order to generate better text, many researchers have spent much time to search for the architecture which decides the necessary factors in proper order. However, there are certain kinds of problems in text which are difficult to detect until the text is actually generated. These problems can be easily detected and solved by introducing a revision phase after generation. In this paper, we argue the importance of text revision with respect to natural language generation, and propose a computational model of text revision. We also describe an experimental Japanese text generation system WEIVER, which incorporated a revision module.

1 Introduction

During the course of text generation, many kinds of decisions should be made. Traditionally, these decisions are classified into two categories; *decisions on what-to-say*, that is, the topic selection and the topic organization, and *decisions on how-to-say*, that is, decisions on grammatical choices and lexical choices. Many of the text generation systems proposed so far make these decisions sequentially in a fixed order. The order is usually arranged from decisions on what-to-say to decisions on how-to-say.

Some researchers, however, have claimed these decisions are dependent on each other and the versatile architecture are required to handle these interactions.[2,4,10] For example, the number of propositions contained in a sentence is constrained by both the rhetorical relations among the propositions and the complexity of the sentence. The former is the constraint of what-to-say and the latter is the one of how-to-say, and the both are dependent on each other. Furthermore, within each of them, there are interactions among decisions.

If one makes decisions sequencially in a fixed order, since the information flow is limited to one direction, that is from what-to-say to how-to-say, it is difficult to realize the interactions among the decisions as mentioned above. Some researchers have developed devices which allow versatile interactions among decision modules.[1,4,10,11] For example, Hovy proposed an architecture which can decide the order of decisions dynamically during the generation process.[4]

One of the limitations of these approaches is that the system still must foresee how the current decisions constrain the subsequent decisions, since the system does not change the decisions once made. However, there are certain kinds of problems which are difficult to find until a text is actually generated. For example, one cannot detect ambiguities of modification relations between phrases until s/he decides lexical choice, word order, punctuation and so on.

This leads us to the idea of revising text once generated. Mann,[7] Gabriel[3] and Shibata[12] partially incorporated this idea into their systems. We follow this line and aim to realize a revision mechanism on computers which is similar to the one of humans'. We consider text revision as a process necessary after text generation[1]. Throughout revision process, since text is always realized, problems such as ambiguities of modification relations are easy to detect. Further, by repeating the revise operations, the quality of the text is improved.

The importance of text revision is also supported by the observation of human writing. From a psycholinguistic viewpoint, Yazdani argued that the largest part of human text writing is devoted to revision rather than initial generation.[13] This is easy to imagine when we introspect on the way we write text.

[1] We call this process *initial generation* in order to emphasize revision process.

2 Introduction to Japanese

For the reader's convenience in understanding the examples in this paper, we provide a brief introduction to Japanese in this section.

A simple Japanese sentence consists of a sequence of postpositional phrases (PPs) followed by a predicate (a verb or an adjective). A PP consists of a noun phrase (NP) followed by a postposition which indicates the case role of the NP. In this case we say "each NP modifies the predicate" and call NPs the *modifiers* and the predicate the *modifiee*. For example, both "太郎は" and "東京に" modify "住んでいる" in sentence (1). Here, the second line indicates the translations of NPs and the third line indicates grammatical relations.

(1) 太郎は 東京に 住んでいる.
Taro Tokyo lives.
NOM in

The order of PPs are not strictly fixed, so it is possible to scramble PPs without changing the meaning of the sentence. For example, a sentence "東京に 太郎は 住んでいる." has the same meaning as the sentence (1). One of the important constraints in Japanese is that no two modification relations cross each other.

When a sentence has only a modifiee, the modification relation can be determined uniquely. However this is not always the case. Sentence (2) is an example in which the noun "花子" is modified by the verb "去っていく (departing)". This is similar to a relative clause in English. In this example, "ポチと (with Poti)" can modify either "去っていく (departing)"or "見ていた (looked at)".

(2) 太郎は ポチと 去っていく 花子を 見ていた.
Taro Poti departing Hanako looked at
NOM with ACC

Depending on which verb the "ポチと (with Poi)" modifies, this sentence gives two interpretations[2]:

- Taro looked at [Hanako departing with Poti]. (where "ポチと" modifies "去っていく")
- With Poti, Taro looked at Hanako departing. (where "ポチと" modifies "見ていた")

Both the interpretations are equally acceptable. To determine the correct modification relation is one of the important goals of natural language understanding. From the viewpoint of natural language generation, the goal is to avoid generating such an ambiguous sentence.

3 Why Revision?

Following the previous section, we discuss the decision order in the generation process and the necessity of revision. In particular, we concentrate on the problems of Japanese sentences with ambiguous modification relations.

Consider sentence (2) again. Assume we want to generate a sentence which has the second meaning, that is "With Poti, Taro looked at Hanako departing." If the generation system would generate the word order of sentence (3) , then "ポチと (with Poti)" can only modify "見ていた (looked at)", and we would obtain a desirable sentence. This is because modifiers can only modify succeeding elements in Japanese.

(3) 太郎は 去っていく 花子を ポチと 見ていた.
Taro departing Hanako Poti looked at
NOM ACC with
(With Poti, Taro looked at Hanako departing.)

Also the ambiguity of sentence (2) can be solved by inserting a comma as follows:

(4) 太郎は ポチと、去っていく 花子を 見ていた.
Taro Poti departing Hanako looked at
NOM with ACC
(With Poti, Taro looked at Hanako departing.)

Furthermore, this ambiguity never happens if the sentence is realized with two sentences. That is:

(5) 花子は 去っていった.
Hanako (NOM) departing
(Hanako was departing.)

太郎は それを ポチと 見ていた.
Taro (NOM) it (ACC) Poti (with) looked at
(Taro looked at it with Poti.)

From these examples, we can see that there are several possibilities to avoid ambiguity; proper decision on word order, lexical choice, punctuation, syntactic structure, and so on. However, it is obviously impossible to foresee what kind of syntactic structure will not cause ambiguities before deciding word order and words. Thus we can find ambiguities of modification only after deciding all these factors.

The problems in the length of sentences and in the depth of embedding are difficult to find before

[2]It is interesting that English translation also causes the ambiguity. We make the difference clear by using brackets and inversion.

text is actually realized as well. These two also affect the quality of the text. The point is whatever the order of the decisions we adopt, there is the possibility that some kinds of problems will still need to be solved. Seeing only examples like the sentence (3), one may think that the ambiguity of modification can be solved by deciding the word order at the end. However this strategy does not work due to the following reasons:

- It is not always possible to find a proper word order which does not cause ambiguities.
- Word order should not be decided only with respect to avoiding ambiguities of modification. There are many fctors on deciding word order, such as old/new information, constraints, focusing constraints and so on.

Even if the system has the ability to decide the order of decisions dynamically, it does not help these problems. By introducing the revision process, we can easily find and solve such kind of problems. In the next section, we describe our approach in detail.

4 Our approach

We consider text generation as a process consisting of two phases; *initial generation* followed by *revision*. The revision phase solves the problems which are difficult to consider in initial generation as we discussed in section 3.

Figure 1 illustrates our model of text generation. At the moment we focus on how-to-say issues, so we assume that the input to the system is a *rough* rhetorical structure, which we call *Pre-rhetorical structure* (PRS). "Rough" means that this structure does not provide the orders among the blocks and the relations among the blocks can be transformed later. *Surface Generator* decides on transformations of rhetorical structure, grammatical choices, lexical choices, and so on. In the initial generation, Surface Generator generates a text from the input PRS. We call this text *draft*, which means that it may not yet be the final version because of the problems mentioned in section 3. Drafts are represented with *draft descriptions*, each of which consists of a PRS and a syntactic structure.

The revision process is realized as a repetition of the *revision cycle*, which consists of *evaluation*, *planning* and *surface change*. In each revision cycle, Evaluator first evaluates the current draft description to detect some problems. Then Planner selects one of them and suggests a change which will solve the problem. After planning, Surface Generator actually changes the surface structure according to the grammar. The revision cycle ends when the surface change succeeds. By repeating revision cycles, the draft description is gradually improved into the final version of the text.

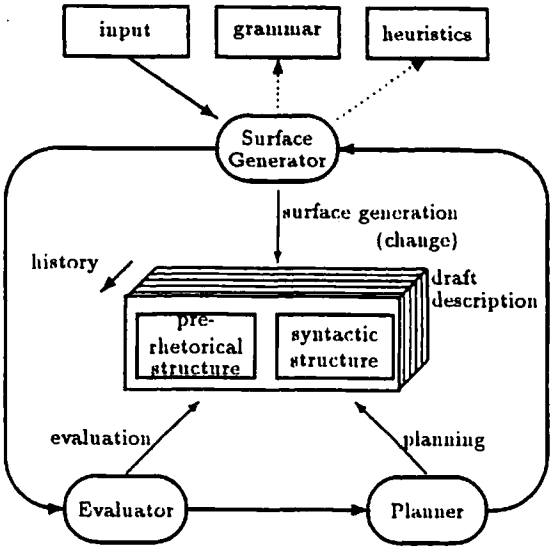

Figure 1 Model of text generation

Evaluation

There is little research on the evaluation of text from the viewpoint of natural language generation. Here we focus on the following problems, as mentioned in section 3, which are difficult to solve before text is actually realized. We call these problems *surface problems*.

- ambiguities of modification relations between phrases
- the length of sentences, clauses and phrases
- the depth of embedded sentences
- center embedding

Planning

In each revision cycle, Planner selects a problem from the ones detected by Evaluator, and suggests a change to solve the problem using heuristics. We call these heuristics *planning rules*. Then Planner sends a *message* which specifies the change to Surface Generator. When the change can not be realized due to some

reason, for example, due to grammatical constraints. Surface Generator requests alternatives from Planner. If Planner cannot suggest alternatives for the problem, it tries the other problem. During the revision cycle, Planner manages the history of draft descriptions and changes to them. By monitoring the history, Planner keeps the revision cycle from falling into an infinite loop.

This model repeats the revision cycle until text becomes good enough. Therefore, even if a surface change introduces new problems, they can be detected and solved in the subsequent cycles. This means that Planner need not consider the side effects caused by each change seriously. At this point, our model is significantly different from previous one-pass generation models, in which each process has to foresee the effects of decisions on the subsequent processes.

Surface Change

Surface Generator is used both in initial generation and in revision. Surface Generator uses a set of heuristics to make decisions on how-to-say. At any point in text generation, there are several choices to take, the heuristics put preference on these choices with respect to various kinds of resources, such as syntactic, semantic and pragmatic ones.

In the revision phase, Surface Generator realizes various kinds of changes on the surface structure, which are proposed by Planner. The change also must satisfy syntactic, semantic and pragmatic constraints.

5 Implementation

The generation model described in the previous section has been partially implemented as WEIVER. We describe some implementation issues of WEIVERin this section, in particular, we focus on realization process of surface change.

5.1 Using Phrasal Lexicon

The generation process can be considered as a set of decisions, each of which is made at a choice point in a decision tree. From this point of view, we can regard a surface change as a change of a decision. So it is desirable that both grammatical and lexical choices are described in uniform representation.

In order to fulfill this requirement, we adopt a *phrasal lexicon* to represent linguistic knowledge. The phrasal lexicon integrates the grammar and the lexicon into a unified representation.[9] In addition, the phrasal lexicon contributes to generating fluent text.[6] We basically follow Jacobs' representation, which represents the phrasal lexicon as a collection of PC pairs (Pattern-Concept pairs).[5] Each PC pair defines a mapping from a part of the semantic structure to syntactic/lexical fragments, and the generator realize surface sentences by constructing these fragments. Using PC pairs, decisions on how-to-say can be considered as choices among PC pairs. From the viewpoint of revision, changes can be realized as replacing PC pairs with alternatives.

```
CONTEXT: [ ]
MAP-FROM: [
    [con:@absorb,
        required-slots:[agt:@CO2],
        optional-slots:[obj:@heat],
        constr:[type:sentence, tense:past] ...],
    [con:@CO2, ...],
    [con:@heat, ...] ]
MAP-TO: [
    [lex:"吸収する",
     con:@absorb,
        slots:[agt:@CO2, obj:@heat],
        constr:[type:sent, tense:past,
                pos:V, connect:period] ...],
    [con:@CO2,
        constr:[pos:NP, case:ga] ...],
    [con:@heat,
        constr:[pos:NP, case:wo] ...] ]
EFFECTS: [...]
```

Figure 2 An example of extended PC pair

We made some extensions on PC pairs in order to realize revision process. Figure 2 shows an example of our PC pair.

A PC pair consists of four parts; CONTEXT, MAP-FROM, MAP-TO and EFFECTS, and defines a mapping from the sub tree specified in MAP-FROM to the sub tree specified in MAP-TO. CONTEXT and MAP-FROM define the conditions by which to apply the PC pair. CONTEXT specifies the constraints on sub tree which dominates the sub tree specified in MAP-FROM. The Surface generator can apply a PC pair, only if both CONTEXT and MAP-FROM unify a subtree of the current structure. EFFECTS defines the effects which are achieved when the PC pair is applied, for example, *make-it-concise*, *pronominalize*, *make-it-formal*, and so forth.

Significant improvements on Jacobs' are as follows. First, our PC pair maps from tree structures to

tree structures, while Jacobs' maps conceptual structures to linear strings. If the generator makes many kinds of decisions on this tree structure, such as lexical, grammatical and even textual, it can treat them in uniform manner, since the result of mapping is always the uniform representation. And the tree structures provide more information for evaluation than strings do. Furthermore, the generator can easily construct the structures larger than a single sentence because of the uniform representation. Jacobs' PC pair is only for single sentence generation.

Secondly, the EFFECT part is newly introduced. The EFFECT part enables the system to take into account more than semantic constraints, say pragmatic constraints, in selecting PC pairs. The EFFECT part is also used by Planner to suggest the decisions to be changed (see the next subsection).

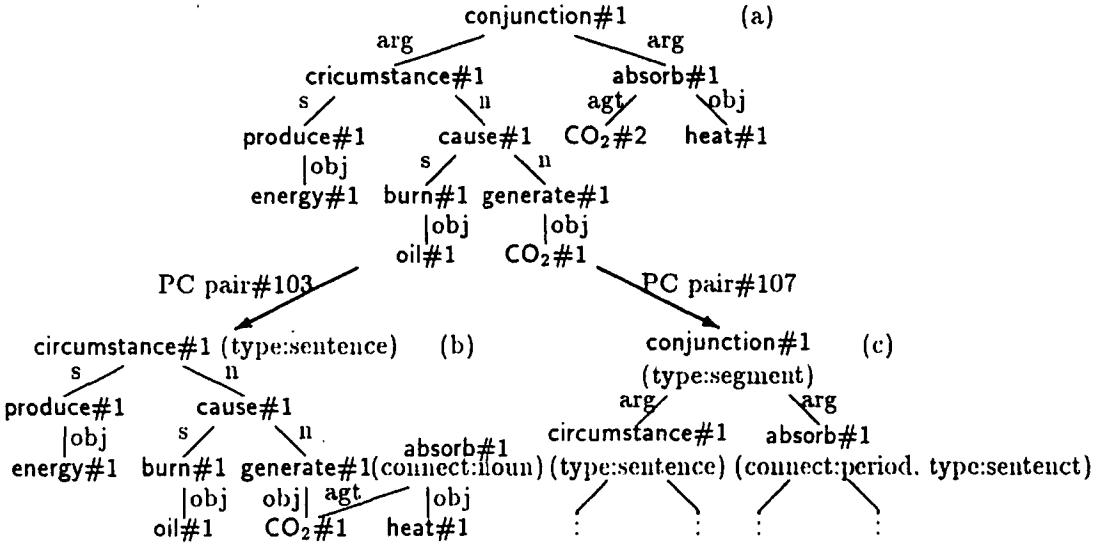

Figure 3 Rewrite of Structure

5.2 An Example

In this subsection, we discuss three significant features of our method through an example.

Consider draft (1). We assume this draft was generated from a PRS. Figure 3 (a) is a part of the PRS, and (b) is an intermediate structure derived from (a) by applying PC pair#103. These structures correspond to the first sentence in draft (1).

If we consider only syntactic and semantic constraints in the first sentence, "燃やすため (due to burning (oil))" can modify either "吸収する (absorb)" or "発生する (is generated)." This gives rise to the two interpretations: "burning oil causes heat to be absorbed by CO_2" and "burning oil causes CO_2 to be generated." Pragmatic knowledge is necessary to solve this ambiguity, i.e. to reject the former interpretation. However, readers are likely to believe the former interpretation before their eyes get to "CO_2." This is similar to the garden-path sentence and brings the system to the revision process. In the following we demonstrate the revise operations used to solve this problem.

Draft (1)

エネルギーの生成過程では、 石油を燃やすため 熱を吸収する CO_2が 発生する.
In producing energy due to burning oil heat absorbing CO_2 is generated.
(Burning oil to produce energy generates CO_2 which absorbs heat.)

エネルギーの消費が増えると CO_2の吸収する 熱が 大きくなり 気温が上昇する
Consuming more energy absorbed by CO_2 heat increases temperature rises.
(Consuming more energy causes the temperature to rise, because CO_2 absorbs more heat.)

Suggesting a choice to change

First, the Planner has to suggest an appropriate change and send the message to Surface Generator. In this example, the planning rule below is applied.

> IF a modifier X can modify two verbs V_1 followed by V_2, and X intends to modify V_2, and V_1 modifies a noun N, THEN remove the PC pair which makes the sub tree whose root corresponds to V_1 be a modifier of N.

Generally, the planning rules suggests PC pairs to be removed or replaced in terms of the features specified in "constr" slots of PC pairs. Then Planner searches for the PC pair to be changed in the *PC pair history*. PC pair history is the history of the PC pairs which are used in the last revision cycle. Assume PC pair#103 specifies the value of the feature "connect" of absorb#1 to "noun", which means absorb#1 should modify a noun. Since PC pair#103 satisfies the requirement of the above rules, Planner sends a message to remove PC pair#103 to Surface generator.

Planner may refer to the EFFECTS part in PC pairs in the history as well, when it searches for the PC pair to be changed. This is the case when things other than grammatical features, for example, pragmatic constraints, are involved.

Draft (2)

エネルギーの生産過程では　石油を燃やすため　CO_2が　発生する.
In producing energy　due to burning oil　CO_2　is generated.
(Burning oil to produce energy causes CO_2 to be generated.)

また　CO_2は　熱を吸収する.
And　CO_2　absorbs heat
(CO_2 absorbs heat).

エネルギーの消費が増えると　この熱が　大きくなり　気温が上昇する.
Consuming more energy　the heat　increases　temperature rises.
(Consuming more energy causes the temperature to rise, because the heat (absorbed by CO_2) increases.)

Internal/external dependency

Next, Surface generator actually changes the draft with respect to the message. In this example, Surface generator replaces PC pair#103 with another PC pair on the choice point shown in figure 3 (a). We assume that PC pair#107 was chosen. This choice leads to generating draft (2). The first sentence in draft (1) was divided into two sentences. At the same time, the value of the feature "connect"in absorb#1 changes from "noun" to "period." This change of the feature value causes the changes of the conditions to apply PC pairs to absorb#1. So the PC pairs which was applied to absorb#1 in draft (1) may not be applicable anymore. The dependency among the applied PC pairs can be determined by refering to the PC pair history.

For example, applying PC pair (a) changes the part of the current intermediate structure. This may enable some other PC pairs to be applicable. And if some of them, say PC pair (b) and (e), are actually applied, these PC pairs are dependent on PC pair (a). We call this the *internal dependency*. Figure 4 illustrates this situation. These dependencies, in general, are represented as a directed acyclic graph, which we call *dependency graph*. If PC pair (b) is replaced, the PC pairs dependent on (b) (enclosed by the dashed line in figure 4) may also be replaced.

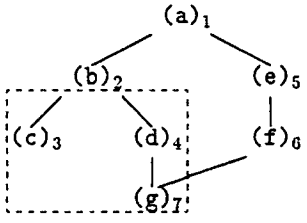

Figure 4　A dependency graph

On the other hand, "CO_2の吸収する 熱が (the heat absorbed by CO_2)" is also changed to just "この 熱が (the heat)". This is because when the reader has read to the second to the last sentence, the focus is on "熱 (heat)"in draft (2), while the focus is on "CO_2" in draft (1). That is, the second sentence in

draft (1) is dependent on the first one in terms of the focusing heuristics.

We call this kind of dependency *external dependency* in order to distinguish it from the internal dependency. In figure 4, the PC pairs enclosed by the dashed line is dependent internally on the PC pair (b). On the other hand, the PC pairs are dependent externally on (b) if they are outside of the dashed line and their surface strings do not proceed that of (b), in this example (e) and (f). External dependencies are caused not by the grammatical constraints, but by the heuristics of focusing, the reader's model, and so on.

Minimal Change Priciple

Note that some parts are intact through this revision in this exapmle. From the viewpoint of efficiency, it is desirable to preserve the parts which are independent of the changes as much as possible. We call this *minimal change principle*.

According to minimal change principle, Surface Generator prefers the PC pairs used in the last revision cycle. Surface Generator checks every conditions in terms of dependencies. With regard to the internal dependency, Surface Generator checks the condition of both PC pairs and heuristics used in the last generation. On the other hand, with regard to the external dependency, it checks only the conditions of the heuristics.

5.3 Advantages over Backtracking

Our approach may seem similar to a naive depth-first search with backtracking. But it is very different from backtracking in the following two respects. Consider figure 4 again. The numbers associated with the nodes indicate the order of the application of PC-pairs.

First, when the planning rules suggest to change a decision, WEIVERtries to change the decision directly. On the other hand, in case of backtracking all decisions which have made since the suggested decision was made should be canceled before making an alternative decision. For example, when the planning rules suggests to change decision (b), WEIVERtries to directly replace the PC-pair (b). On the other hand, by backtracking all decisions (g) through (c) would be canceled.

Secondly, when WEIVERtries to replace a PC-pair, it tries to reuse the PC-pairs which are subordinate to the PC-pair to change as much as possible. For example, when WEIVERtries to replace (b) in figure 4, the system prefers to use (c), (d) and (g) which were used with (b) in the last revision cycle. In backtracking, the system will first try the same PC-pairs as were tried when (b) was adopted.

6 Related works

There is little research on text revision. This is mainly because the methodology of evaluating text is not established yet. Mann and Moore's KDS is one of the oldest systems which incorporate an element of revision.[8] KDS is different from WEIVERin two significant points. One is that KDS never reverts the decisions on rules once made, and the other is that KDS's evaluation is not on surface text but on the intermediate expressions. Due to these difference, KDS has difficulties in detecting and solving the surface problems mentioned in section 3.

Gabriel's *Yh* is another example which incorporates a revision module, however, *Yh* revises the intermediate expression only once.[3] So *Yh* also suffers from the same short comings that of KDS does.

Unlike the above two systems, Mann's Penman evaluates the surface text once generated.[7] Penman's approach is quite similar to ours. However, Mann's main concern in the Penman project is in implementing a broad coverage grammar (called NIGEL) and in providing a mechanism for organizing the rhetorical structure of text. The revision module has not been fully discussed and the details are not clear.

The importance of text revision is emphasized in Meteer's PhD work.[11] She collected more than 500 changes on a published paper which were made by a professional edior and categorized them into 16 types. Based on the result, Meteer proposed a data structure called Text Structure, which aims to provide a data structure versatile enough for the revision operation, and bridging between the how-to-say and the what-to-say modules. Unfortunately, Meteer has not yet given a concrete operational model of text revision using Text Structure. It is not clear to what extent Text Structure contributes to computer text revision.

7 Concluding remarks

Text revision does not oppose to (initial) text generation. They are both complementary. We have pointed out that there are problems which are difficult to consider in initial generation. Our claim is

that text generation should consist of two phases: the initial generation and the revision.

It will be efficient if all the decisions can be made only in the initial generation with one-pass process as Meteer claims.[11] However, there must be a limitation in the quality of text only with this approach. It is obvious that the better the initial generation is, the more efficiently we will be able to obtain the final text through revision. If we start revision with a very poor text, it may take a long time to revise into a good text. So there must be a compromise in sharing the burden between two phases in order to obtain a good text efficiently. In this paper, we proposed a model to realize this idea and described its implementation WEIVER.

There is little research on text revision so far. In particular, this is due to the difficulty of text evaluation. There is no consensus on criteria in evaluating and changing text. Observing the human writing process may provide valuable insight into this problem, as Meteer has shown in her research.[11] We have also conducted a psychological experiment to extract humans' criteria to evaluate and improve text. The collected data is now under analysis.

Eventually, we will analyze the results of the experiment and feed this information back into the system. At the same time we will extend the current system and evaluate its performance with more examples.

Acknowledgements

The authors would like to thank Mr. Craig P. Hunter for his contribution in improving English of the paper.

References

[1] D. E. Appelt. TELEGRAM: A grammar formalism for language planning. In *the Proceedings of the International Joint Conference on Artificial Intelligence*, pages 595–599, 1983.

[2] L. Danlos. Conceptual and linguistic decisions in generation. In *the Proceedings of the International Conference on Computational Linguistics*, pages 501–504, 1984.

[3] R. P. Gabriel. Deliberate writing. In D. D. McDonald and L. Bolc, editors, *Natural Language Generation Systems*, chapter 1, pages 1–46, Springer-Verlag, 1988.

[4] E. H. Hovy. *Generating Natural Language under Pragmatic Constraints*. Lawrence Erlbaum Associates, 1988.

[5] P. S. Jacobs. PHRED: A generator for natural language interfaces. In D. D. McDonald and L. Bolc, editors, *Natural Language Generation Systems*, chapter 7, pages 256–279. Springer-Verlag, 1988.

[6] K. Kukich. Fluency in natural language reports. In D. D. McDonald and Leonard Bolc, editors, *Natural Language Generation Systems*, chapter 8, pages 280–311. Springer-Verlag, 1988.

[7] W. C. Mann. An overview of the Penman text generation system. In *the Proceedings of the National Conference on Artificial Intelligence*, pages 261–265, 1983.

[8] W. C. Mann and J. A. Moore. Computer generation of multiparagraph English text. *American Journal of Computational Linguistics*, 7(1):17–29, 1981.

[9] C. Matthiessen. Lexico(grammatical) choice in text generation. In *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, chapter 10, pages 249–292, Kluwer Academic Publishers, 1991.

[10] K. R. McKeown and M. Elhadad. A contrastive evaluation of functional unification grammar for surface language generation: A case study in choice of connectives. In C. L. Paris, W. R. Swartout, and W. C. Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*, chapter 14, pages 351–396, Kluwer Academic Publishers, 1991.

[11] M. W. Meteer. *The Generation Gap: The Problem of Expressibility in Text Planning*. PhD thesis, University of Massachusetts, 1990.

[12] S. Shibata, M. Fujita, and K. Mazeki. Text generation algorithm for well-formed sentences. In *Proceedings of the 41st Annual Convention IPS Japan*, pages 3:177-3:178, 1990. (in Japanese).

[13] M. Yazdani. Reviewing as a component of the text generation process. In G. Kempen, editor, *Natural Language Generation*, chapter 13, pages 183–190, Martinus Nijhoff, 1987.