Text Revision: A Model and Its Implementation

Kentaro Inui, Takenobu Tokunaga and Hozumi Tanaka

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

Abstract

To generate good text, many kinds of decisions should be made. Many researchers have spent much time searching for the architecture that would determine a proper order for these decisions. However, even if such an architecture is found, there are still certain kinds of problems that are difficult to consider during the generation process. Those problems can be more easily detected and solved by introducing a revision process 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 discuss its implementation issues and describe an experimental Japanese text generation system, WEIVER.

1 Introduction

During the course of text generation, many kinds of decisions should be made. These decisions are generally classified into two categories: *decisions on what-to-say*, that is, topic selection and topic organization, and *decisions on how-to-say*, that is, decisions on grammatical choices and lexical choices. Many of the text generation systems proposed thus far make these decisions in a fixed sequential order. The order usually begins with decisions on what-to-say and ends with decisions on how-to-say.

These decisions, however, are interdependent, and a more versatile architecture is required to handle these interactions [4, 8, 16]. For example, the number of propositions contained in a sentence is constrained by 2 sets of decisions, the rhetorical relations among the propositions (what-to-say) and the complexity of the sentence (how-to-say). These decisions are interdependent. Furthermore, within each of the sets, there are other interactions among decisions. To account for this, some researchers have developed devices that allow interactions among decision modules [1, 8, 16, 17]. For example, Hovy proposed an architecture that can dynamically decide the order of decisions during the generation process [8].

One limitation to these approaches is that the system needs to foresee how a generation decision constrains subsequent decisions. There are, however, certain kinds of problems that are difficult to detect before the text is actually generated. Structural ambiguities, for example, are difficult to detect before lexical choice, word order, and punctuation decisions are all made. We call these kinds of problems *surface problems*.

Whatever decision order we adopt, there is the possibility that surface problems will still remain (see Sect. 3).

This leads us to the idea of revising text once generated. We introduce a text revision process that solves surface problems. 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 incorporates a revision module.

Since our target language is Japanese, we provide a brief introduction to Japanese in the next section. In Sect. 3 we give examples illustrating why the revision process is necessary. In Sect. 4 we provide a summary of our approach to solve the problems presented in Sect. 3. The implementation issues are discussed in Sect. 5.

2 Brief Introduction to Japanese

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. We say "each PP modifies the predicate" and call PPs the *modifiers* and the predicate the *modifiee*. For example, both "*John-ga*"[1] and "*Tokyo-ni*" modify "*sundeiru*" in sentence (1).

(1) *John-ga* *Tokyo-ni* *sundeiru.*
John-NOM in Tokyo lives.

The order of PPs is not strictly fixed, so it is possible to scramble PPs without changing the meaning of the sentence[2]. For example, the sentence "*Tokyo-ni John-ga sundeiru.*" has the same meaning as that of sentence (1). As with prepositional phrase attachments in English, one of the important constraints in Japanese is that no two modification relations cross each other[3].

When a sentence has only one modifiee, the modification relation can be uniquely determined. However, this is not always the case. Sentence (2) is an example in which the noun "*Mary*" is modified by the verb "*satteiku*." In this example, "*Poochie-to*" can modify either "*satteiku*" or "*miteita*."

(2) *John-ga* *Poochie-to* *satteiku* *Mary-wo* *miteita.*
John-NOM with Poochie departing Mary-ACC look at-PAST

Depending on which verb "*Poochie-to*" modifies, this sentence gives two interpretations[4]:

- John looked at [Mary departing with Poochie].
  (the case "*Poochie-to*" modifies "*satteiku*")

[1] We denote PP in "NP-postposition" form for convenience of explanation.
[2] Of course, scrambling may change nuances and the naturalness of the sentence.
[3] There are some exceptions but this is generally considered a reasonable constraint.
[4] It is interesting that the translation in English is also ambiguous. We make the difference clear by using brackets and inversion.

– With Poochie, John looked at Mary departing.
(the case "*Poochie-to*" modifies "*miteita*")

Given no contextual information, both interpretations are equally acceptable. Determining the correct modification relation is one of the major problems of natural language understanding. In natural language generation, it is important to avoid generating such an ambiguous sentence.

3 Why Revision?

In this section, we argue that to solve surface problems, text generation should exploit the revision process.

Consider sentence (2) again. Assume that we want to generate a sentence representing the second meaning, "With Poochie, John looked at Mary departing." If the system made a decision on the word order as that in sentence (3), then "*Poochie-to*" could only modify "*miteita*," and we would obtain a desirable sentence. This is because modifiers can only modify succeeding elements in Japanese.

(3) *John-ga* *satteiku* *Mary-wo* *Poochie-to* *miteita.*
John-NOM departing Mary-ACC with Poochie look at-PAST
(With Poochie, John looked at Mary departing.)

The ambiguity of sentence (2) could also be resolved by inserting a comma (sentence (4)). This is because a modifier preceding a comma will not modify the element immediately following the comma.

(4) *John-ga* *Poochie-to,* *satteiku* *Mary-wo* *miteita.*
John-NOM with Poochie departing Mary-ACC look at-PAST
(With Poochie, John looked at Mary departing.)

In the same way, we can avoid structural ambiguities through proper decisions on word order and punctuation. This seems to imply that you have only to make these decisions at the end of the process. This strategy, however, does not work for the following reasons:

– It is not always possible to find a proper word order that does not cause any ambiguities.
– Word order should not be decided only with respect to avoiding structural ambiguities. There are many factors that decide word order, such as old/new information, focus, etc.

The following example demonstrates this difficulty[5].

(5) *John-wa,* [[*Tom-ga* *Φ* *Φ* *yôkyûsita* *node*]
John-TOP Tom-NOM (John-DAT) (position of president-ACC) request-PAST because

[5] We enclose subordinate clauses with brackets. *Φ* denotes an ellipsis.

kare-ni	*syatyô-wo*	*yuzuru-to*]	*kôhyôsita.*
him-DAT (Tom)	position of president-ACC	transfer-COMP	anounce-PAST

Due to the structural ambiguity, sentence (5) also has two possible interpretations:

- John made an announcement that in response to Tom's request, John would transfer the position of president to Tom.
  (the case "*Tom-ga yôkyûsita node*" modifies "*yuzuru*")
- John made an announcement, in response to Tom's request, that John would transfer the position of president to Tom.
  (the case "*Tom-ga yôkyûsita node*" modifies "*kôhyôsita*")

Assume the first interpretation is the intended meaning. To avoid the ambiguity, try an alternative word order, as in sentence (6), taking the same tactic as in sentence (3). Note that "*kare-ni*" has been moved in front of "*Tom-ga*," and compare with sentence (5).

(6) *John-wa,* John-TOP	[*kare-ni* him-DAT (??)	[*Tom-ga* Tom-NOM	*Φ* (position of president-ACC)	*yôkyûsita* request-PAST	*node*] because

syatyô-wo position of president-ACC	*yuzuru-to*] transfer-COMP	*kôhyôsita.* announce-PAST

This word order, however, causes another problem. Although "*kare*" should refer to "*Tom*" as in sentence (5), "*kare*" in sentence (6) instead refers to "*John.*" This problem arises because all of the lexical choices had already been decided before deciding on the word order.

In this example, the intended meaning can be attained, however, by adopting another syntactic structure, like that in sentence (7). The subordinate clause in sentence (5) is realized as a coordinate clause.

(7) *John-wa,* John-TOP	[*Tom-ga* Tom-NOM	*Φ* (position of president-ACC)	*yôkyûsita* request-PAST	*node*] because

syatyô-wo position of president-ACC	*yuzurukoto-wo* transferring-ACC	*kim* decide	*e,* and	*sore-wo* it-ACC	*kôhyôsita.* announce-PAST

From this we can see that, for certain sentences, some syntactic structures inherently have structural ambiguities. This suggests that at the decision point on the syntactic structure you have to foresee whether, or not, your decision will cause any structural ambiguities. This task, however, is very difficult unless the text is actually generated. In this sense, it is a surface problem.

The complexity of a sentence, e.g. length, depth of embedding, etc., is also a kind of surface problem. For example, consider the case when a proposition "My car is

new." is embedded into the other proposition "My car is French." You can produce "*My new car is French.*" as well as "*My car, which is new, is French.*" [19]. There is no syntactic embedding in the first sentence. Therefore, the depth of (syntactic) embedding can be measured only after lexical and grammatical choices are made.

Even if the system has the ability to decide the order of decisions dynamically, it does not alleviate this problem. By introducing the revision process, we can easily find and solve such kinds of problems. This is one of the most important motivations of our research.

4 Our Approach

Text generation consists of two processes: *initial generation process*, followed by *revision process*. The revision process solves the surface problems, which are difficult to consider during the initial generation, as we discussed in Sect. 3.

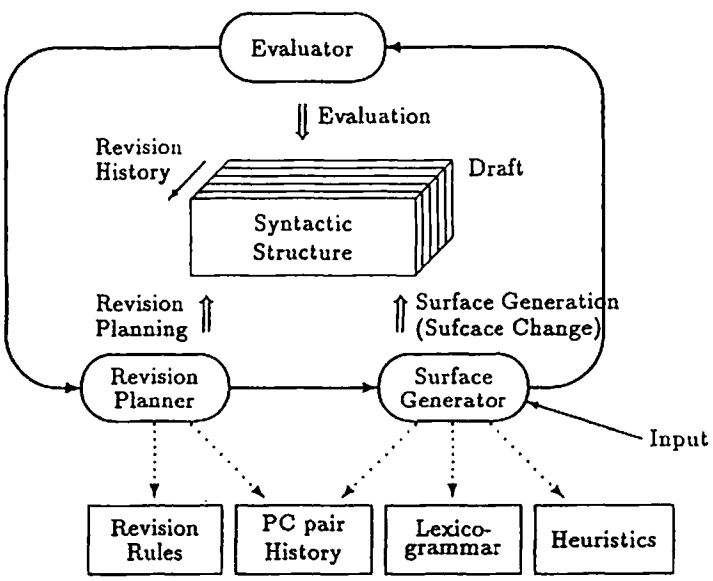

Fig. 1. Model of text generation

Figure 1 illustrates our model of text generation. At the moment we focus on how-to-say decisions, so we assume that the input to the system is a rhetorical structure, that represents what-to-say information. The system has three modules: *surface generator*, *evaluator* and *revision planner*.

In the initial generation, the surface generator makes decisions on grammatical and lexical choices to generate a text from the input by referring to the two kinds of resources: the *lexico-grammar* and the *heuristics*. The lexico-grammar provides the grammatical and lexical constraints [14]. The heuristics provide the pragmatic and

textual preferences, such as referring expressions and topicalization. The heuristics are used at each decision point where the surface generator has more than one choice that satisfies the grammatical and lexical constraints. The output of the surface generator is called the *draft*, because it may not be the final (optimal) version of the text, as a result of surface problems.

The revision process consists of repetitions of the *revision cycle*. Each revision cycle consists of *evaluation*, *revision planning*, and *surface change*. In a revision cycle, first the evaluator evaluates the current draft to detect surface problems. Then, the revision planner selects one of them, and suggests a change that will solve the problem. The surface generator actually changes the draft. A revision cycle ends when the surface change is successfully accomplished. By repeating revision cycles, the draft can be gradually improved into the final version of the text.

4.1 Evaluation

Here we focus on the following two surface problems.

- structural ambiguity
- sentence complexity
  - the length of sentences, clauses and phrases
  - the depth of embedding and modification relations
  - the depth of center embedding

The evaluator evaluates the draft, which is an actual text rather than some intermediate representation. The draft is represented as its syntactic structure, which provides helpful information for the evaluation.

Even with the state-of-the-art techniques of natural language understanding, it is difficult to find structural ambiguities. The evaluator has to detect the structural ambiguities that even humans cannot disambiguate. For example, the examples shown in Sect. 2 and 3 may not be ambiguous if proper contexts are given. At the moment, our system requires human assistance to detect structural ambiguities.

The complexity of the sentence mainly affects the readability. Several criteria for measuring the complexity of the Japanese sentence have been proposed in research on machine translation systems and text revision support systems. We focus on the criteria itemized above regarding sentence complexity. In the current system, we establish upper and lower bounds for each criteria. An example of the values is shown in Table 1.

Table 1. An example of the criteria regarding sentence complexity

Criteria	Upper bound	Lower bound
Length of sentence (words)	40	5
Depth of clause embedding	3	0
Depth of modification relation in NP	3	0
Depth of center embedding	1	0

4.2 Revision Planning

The revision planner selects one of the problems detected by the evaluator, and suggests a change to solve it using heuristics. We call these heuristics the *revision rules*. The rules are assigned static preference. The revision planner chooses the most preferable rule from the rules of which preconditions are satisfied. Then the revision planner sends the surface generator a *message* that describes the change in the chosen revision rule. When the change cannot be realized because of a reason such as grammatical constraints, the surface generator requests an alternative message from the revision planner. If the revision planner cannot suggest any alternative change for the problem, it tries another problem. The revision planner manages the history of the drafts and the changes (*revision history*). By monitoring the history, the revision planner keeps the revision process from falling into an infinite loop.

Our model repeats the revision cycle until we produce an acceptable text. Therefore, even if a surface change introduces new problems, these can be detected and solved in the subsequent cycles. This means that the revision planner need not seriously consider the side effects of the changes. At this point, our model is significantly different from previous one-pass generation models, in which the system has to foresee the effects of each decision on the subsequent processes.

4.3 Surface Change

In addition to the initial generation, the surface change is also handled by the surface generator. The examples in Sect. 3 show that solving the surface problems involves various kinds of surface changes, such as changes of word order, punctuation, lexical choice, syntactic structure, and so on. Moreover, each surface change should produce an alternative draft that both satisfies the grammatical and lexical constraints, and is supported by the pragmatic and textual preferences as well as the initial draft. Otherwise, the draft may becomes worse due to the changes.

5 Implementation

The generation model described in the previous section has been partially implemented as WEIVER. In this section, we discuss implementation issues of WEIVER. In particular, we focus on the surface change.

5.1 Lexico-grammar

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 decisions. The system is required to change various kinds of decisions on grammatical and lexical choices to solve the surface problems. It is thus desirable that both grammatical and lexical knowledge are described in a uniform representation.

We therefore adopt a *phrasal lexicon*, which integrates the grammar and the lexicon into a unified representation [14]. In addition, the phrasal lexicon contributes

to the generation of fluent text [11]. Basically, we follow Jacobs' representation, which represents the phrasal lexicon as a collection of PC pairs (Pattern-Concept pairs) [10]. In Jacobs' framework, each PC pair defines a mapping from a part of the semantic structure to the syntactic/lexical fragments, and the generator constructs surface sentences from these fragments. Decisions on how-to-say can be considered as choices among PC pairs. From the viewpoint of revision, a surface change can be attained by replacing PC pairs with alternatives.

```
CONTEXT: [ ],
MAP-FROM: [ [$1, con:@action,
                 required-slots:[agt:$2, obj:$3], optional-slots:[ ],
                 constr:[type:sentence] ],
            [$2, con:@animate],
            [$3, con:@concrete-object],
MAP-TO: [ [$1, con:@action,
              slots:[agt:$2, obj:$3],
              constr:[type:sentence, voice:passive, pos:VP, connect:period] ],
          [$2, con:@animate],
          [$3, con:@concrete-object,
              constr:[type:TOP, pos:NP, connect:V],
EFFECTS: [current-focus($3), ...]
```

Fig. 2. An example of extended PC pair

To realize the revision process, we made some extensions on PC pairs. Each PC pair defines a tree-to-tree mapping. By applying PC pairs, the surface generator incrementally transforms the input rhetorical structure into a syntactic structure (see Fig. 3). We call the structures that are partially transferred the *intermediate structures*. Figure 2 shows an example of our PC pair [6]. A PC pair consists of four parts: CONTEXT, MAP-FROM, MAP-TO and EFFECTS. A PC pair can be applied only if both its CONTEXT and MAP-FROM unify with a subtree of the rhetorical/intermediate structure. If more than one PC pair is applicable at a decision point, the surface generator refers to the heuristics to choose one of them. After choosing a PC pair, the surface generator replaces the subtree of the rhetorical/intermediate structure, specified by MAP-FROM, with the MAP-TO subtree. EFFECTS defines the effects that will be achieved when the PC pair is applied. The PC pair in Fig. 2 is applied to realize a proposition as a passive sentence when an object of an action (node $3) is the current local focus.

The extensions to Jacobs' are as follows. First, our PC pair maps from tree structures to tree structures, while Jacobs' maps conceptual structures to linear strings. Because the result of applying our PC pairs is always a uniform representation, the generator can treat the various kinds of decisions, such as lexical, grammatical and, even textual ones, in a uniform manner. Jacobs' PC pair is only for single sentence

[6] "$N" and "@X" are typed variables to unify with an identifier of a node and a concept, respectively.

generation. Furthermore, the tree structures provide more information for evaluation than strings do. Secondly, in selecting PC pairs, the newly introduced EFFECTS part enables the system to take into account not just semantic constraints but also pragmatic preferences. The EFFECT part is also used by the revision planner to suggest the decisions to be changed (see Sect. 5.4).

5.2 An Example

In this section, we show another brief example. In the following sections we discuss several features of WEIVER using this example.

The Initial Generation. The rhetorical structure shown in Fig. 3 (a) is the input[7]. It has four propositions. At the first step of the initial generation, the surface generator divides the input propositions into some sentences. This is realized by applying PC pairs to the subtree near the root node of the input rhetorical structure. If there is more than one applicable PC pair, the generator chooses one of them by referring to the heuristics. Scott and Souza [20] proposed the heuristics of organizing propositions. Their heuristics guide the linguistic realization of the rhetorical structure including embedding and coordination. Our heuristics follow them.

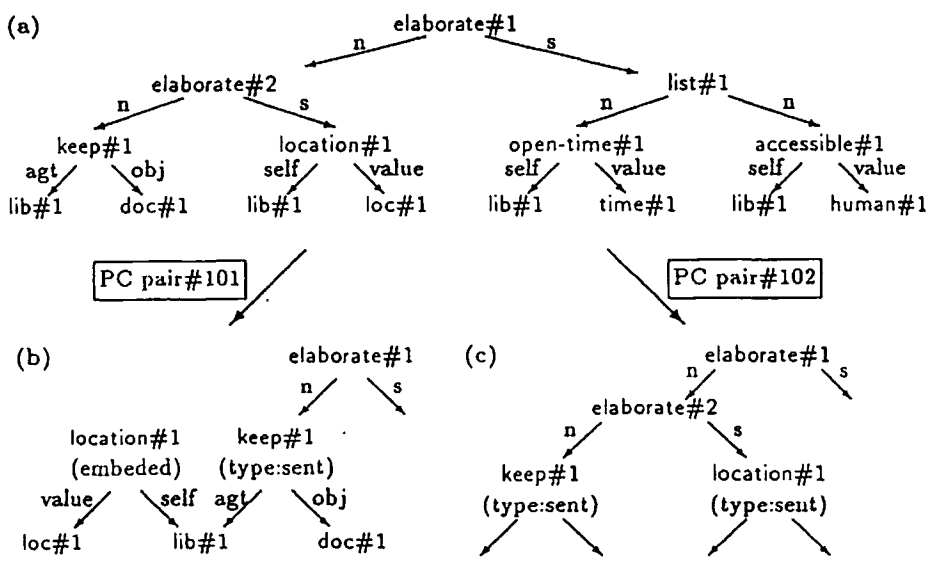

Fig. 3. Transformation of Structure

At the decision point (a) in Fig.3, there are two applicable PC pairs, PC pair#101 and PC pair#102. Assume PC pair#101 is chosen by referring to the heuristics. As

[7] Actually, the input includes the other information, such as focus.

a result, proposition location#1 is embedded to proposition keep#1 (Fig. 3 (b)), and these two propositions will be realized as one sentence, as in the first sentence in draft (8).

(8) *siryô-wa,* [*tonari-no tatemono-no 4kai-no itiban'oku-ni aru*]
document-TOP in the most inner part of the next building on the 4th floor

siryôsitu-ni hokansareteimasu. siryôsitu-wa 9zi-kara 5zi-made
library-LOC keep-PASS Library-TOP from 9 to 5

aitei te darede-mo tukaemasu.
open and everybody accessible.

(The document is kept in the library in the most inner part of the next building on the 4th floor. The library is open from 9 to 5 and is open to the public.)

The Revision Cycle. After the initial generation, WEIVER enters the first revision cycle. First, the evaluator evaluates draft (8) and detects a surface problem; the modification of the noun phrase "*siryôsitu,*" that is "*tonari-no ... aru,*" is too deep according to the criterion shown in Table 1[8]. Next, the revision planner suggests that PC pair#101 should be replaced in order to solve the problem. Following this suggestion, the surface generator actually replaces it with PC pair#102 at the decision point (a) in Fig. 3. As a result, the proposition location#1, which was embedded in draft (8), will be realized as a separate sentence. The new draft is shown in draft (9).

(9) *siryô-wa siryôsitu-ni hokansareteimasu. siryôsitu-wa*
document-TOP library-LOC keep-PASS library-TOP

tonari-no tatemono-no 4kai-no itiban'oku-ni arimasu.
in the most inner part of the next building on the 4th floor

Φ 9zi-kara 5zi-made aitei te darede-mo tukaemasu.
(siryôsitu-TOP) from 9 to 5 open and everybody accessible

This is the end of one revision cycle. In the following process, WEIVER repeats revision cycles as well. In the following sections, we explain the following points using the above example:

- how to find the PC pair to be replaced,
- how to change the draft with its intended meaning preserved,
- how to change the draft without degrading the quality of the draft.

[8] This problem is difficult to avoid in the initial generation, because it is difficult to foresee that the referring expression of loc#1 in Fig. RS (a) will be such a long NP when one decides the embedding.

5.3 PC Pair History

First, we introduce the *PC pair history* as a resource for the revision planning and the surface change.

In the initial generation, the surface generator keeps track of the process as a PC pair history. It is represented as a data structure similar to the dependency network used in the truth maintenance system (TMS) [5]. From the viewpoint of TMS, we can roughly regard the input (rhetorical structure) as a set of facts, each decision (PC pair) as an assumption, and the resultant subtree as a conclusion.

Figure 4 (a) shows a fragment of a sample PC pair history. It shows that PC pair#72 (see Fig. 2) was applied under the precondition represented as the nodes above and, as a result, produced the several features represented as the nodes below. "n_i" denotes the identifier of the node in the rhetorical/intermediate structure. The surface generator records the PC pair history by gathering subnetworks, like those shown in Fig. 4 (a). In general, the resultant network is a directed acyclic graph (see Fig. 4 (b)). We also call this the *dependency network*. In the dependency network, each node has a state, either *in* or *out*. The states propagate along the links as they do in TMS. This mechanism enables us to reconstruct the intermediate structure, which is the result of removing a PC pair (see Sect. 5.6).

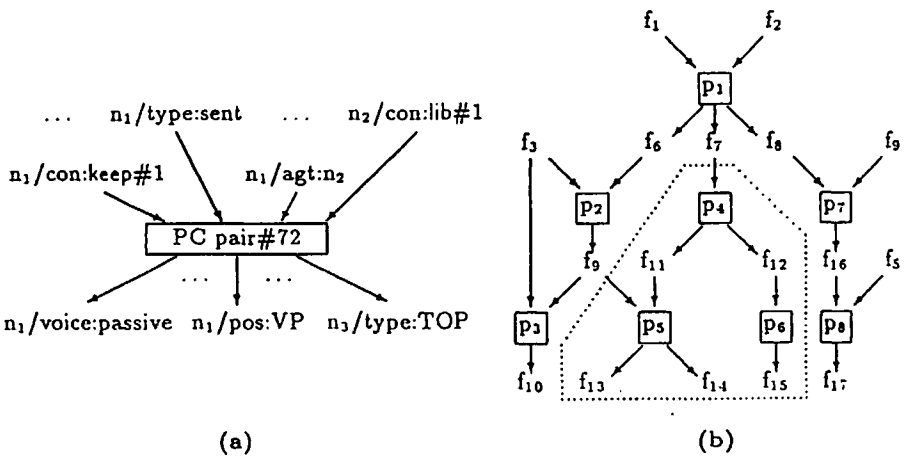

Fig. 4. Dependency network

The dependencies among the PC pairs exist not only because of their preconditions but also because of the heuristics. The PC pair history holds both kinds of dependencies. A number of heuristics on how-to-say have been proposed in the previous research on text generation, although at the moment we have implemented only a few of them in WEIVER. The heuristics are used to generate anaphora [3], ellipsis [9], exophora [2, 19], connectives [6], etc. They are also used to make the pragmatic decisions (formality, etc.) [8].

Most heuristics are context sensitive. Consider the focusing heuristics as an example. Generating an anaphora is partially dependent on the current focus state

described as in the rule below [15].

> If a noun phrase denotes the "given" information and
> it was the previous current discourse focus,
> then pronominalize it.

Thus, most heuristics can be represented as production rules. In the PC pair history, the preconditions of the applied heuristics are also recorded to represent these heuristics dependencies.

5.4 Suggesting a PC pair to Change

The revision planner suggests an appropriate change and sends the message to the surface generator. In the above example, the following revision rule will be applied.

> If an adjective phrase M, which modifies a noun phrase, is too deep, and
> there exists a node in M whose semantic node has a feature "*embedded*,"
> then remove the feature.

Generally, the revision rules suggest which feature should be removed from the current draft. Sometimes they even suggest which feature should be added in place of the removed feature. After choosing an applicable revision rule, the revision planner searches the PC pair history for the PC pair that introduced the features in question. That is easy because the PC pair history holds the dependencies among the PC pairs and the features. In this example, the feature "*embedded*" of location#1 is in question. Fig. 3 shows "*embedded*" is introduced by PC pair#101. Therefore, the message "*Remove PC pair#101.*" is sent to the surface generator. The revision planner further refers to the EFFECTS part in the PC pairs if pragmatic constraints are involved in addition to grammatical features.

5.5 Internal/external Dependencies

Next, the surface generator actually changes the draft with respect to the message. In this example, the surface generator replaces PC pair#101 with PC pair#102 at the choice point shown in Fig. 3 (a). As a result, the feature "*embedded*" in location#1 will be removed, and the first sentence in draft (8) will be divided into two separate sentences.

Because of this change, some of the PC pairs that were applied after the decision point (a) to generate draft (8) may now become inapplicable. For example, the PC pair that realized location#1 as an adjective phrase is not applicable any longer. Thus, there are dependencies among the PC pairs that are applied together to generate a draft. Such dependencies can be found by referring to the PC pair history. Here, we define two types of dependencies: *internal* and *external dependencies*.

Internal Dependency. Assume PC pair p_4 is replaced with PC pair p_4' in Fig.4 (b). Since the features f_{11} and f_{12} may not be produced by p_4', both p_5 and p_6 may not be applicable anymore. In other words, p_5 and p_6 are dependent on p_4. We call these *internal dependencies*. The internal dependencies can be found by traversing the dependency network.

External Dependency. Note that the topicalized subject "*siryôsitu-wa*" was omitted in draft (9). Such ellipses are introduced not because of the grammatical constraints, but because of the textual preferences. The heuristic applied in this example is "The topic in the current sentence should be omitted when it is the same as one in the latest sentence." Thus, some PC pairs are dependent on the replaced PC pair not because of the preconditions of the PC pairs, but because of the heuristics. We call these *external dependencies*. The external dependencies can be found by referring to the preconditions of the heuristics. In general, A PC pair p_i depends externally on the replaced PC pair p_j only when p_i does not depend internally on p_j, and when p_j is included in the preconditions of the heuristics applied to choose p_i.

5.6 Generating Alternatives

In this section, we show how the surface generator realizes the surface changes with respect to the internal and external dependencies. The procedure is as follows:

1. receive the message from the revision planner,
2. make the state of the PC pair in question *out*, spread the state over the dependency network, and construct the intermediate structure (S) from the features whose current states are *in*,
3. invoke the ordinary generation process (apply the PC pairs to S until all the required decisions are made),
4. make the collection of the PC pairs (C) which depend externally on the changed parts,
5. if C is empty, terminate the process, otherwise, choose and remove a PC pair from C that corresponds to the left most part in the draft,
6. if the PC pair chosen in 5 still satisfies the heuristics, go to 5, otherwise go to 2.

For example, assume the PC pair to be replaced is p_4 in Fig. 4 (b). The PC pairs and the features enclosed by the dotted line are set to *out*, and the intermediate structure is constructed of the features with *in* states. Then the surface generator applies the alternative PC pairs to generate an alternative draft. Our procedure ensures that the new draft does not degrade, by also considering the external dependencies as described in steps 4 through 6.

Although this procedure often works successfully, it still has drawbacks. Assume PC pair p_5 was introduced in the previous revision cycle, that is, at that time WEIVER replaced a PC pair with PC pair p_5 to solve a surface problem. If WEIVER now removes p_4, p_5 may also be removed and the previous surface problem may be introduced again. To avoid this problem, step 3 in the above procedure should be extended as follows:

3'. if the PC pairs depending internally on the PC pair to be replaced include a PC pair (p_i) that was introduced in the previous revision cycle, upgrade the priority of p_i and invoke the generation process, otherwise, invoke the ordinary generation process.

In our method, because the surface generator generates an alternative draft from the same rhetorical structure, it is ensured that the new draft keeps the intended

meaning intact. Moreover, because the surface generator also considers the external dependencies, the new draft satisfies not only the grammatical and lexical constraints, but also the pragmatic and textual preferences. Thus, it is guaranteed that the surface change will not worsen the quality of the draft.

5.7 From the Viewpoint of Backtracking

Our method is very similar to a dependency-directed backtracking (DDB) in a justification-based TMS (JTMS). Actually, we believe that the text revision can be realized naturally in this line. As mentioned in Sect. 5.3, each decision can be regarded as an assumption that is justified by its preconditions. That is, replacing a PC pair can be seen as retracting an assumption and adding an alternative one. Thus, with respect to efficiency, our method has the advantage over a naive depth-first backtracking, as discussed in Doyle's paper [5].

However, there needs to be some extensions to JTMS for text revision. Because the heuristics make it difficult to specify the dependencies (see Sect. 5.3), one cannot utilize the JTMS algorithm for the dependency propagation. In our method, the surface generator starts the generation process from an intermediate structure, and then sees if the heuristics are still satisfied. Furthermore, the surface generator prefers the previous surface changes, as discussed in Sect. 5.6.

6 Related Work

We have discussed text revision as a means of solving surface problems. The importance of text revision has also been argued from other points of view. Those arguments are summarized by Wong as follows [21]:

- Revision has psychological reality [22].
- Revision enables feedback from how-to-say to what-to-say. This is important because some factors are detectable only after realization [18].
- With revision, the giant step of the whole generation task can be divided into smaller tasks [22].

Our model also inherits these advantages.

Mann and Moore's KDS [13] and Gabriel's *Yh* [7] are examples of systems that have implemented the revision component. They are different from WEIVER in two ways. First, they never change the decisions, and second, instead of the surface text they evaluate the intermediate representation. Because of these differences, they have difficulties in detecting and solving the surface problems.

Several models have been proposed that evaluate the surface text in revision. Mann's Penman [12] and Wong's blackboard model [21] evaluate the draft, referring to information such as the syntactic structure. Meteer [18] and Yazdani [22] propose models that parse the draft. Our model is very similar to these four models. However, the implementation issues of these models have not yet been fully discussed.

7 Concluding Remarks

In this paper we argued the importance of text revision with respect to natural language generation, and proposed a model that incorporated a revision module. Although we focused on the revision in terms of solving the surface problems, we believe that the revision plays important roles in the other aspects of improving text, as discussed by Meteer [18]. We also discussed the implementation issues of our model. We adopted the JTMS-like approach to keep several constraints consistent in changing the draft.

At present, there is little documented research on text revision. This is due, in particular, to the difficulty of text evaluation. There is no consensus on criteria for evaluation and improvement. Observing the human writing process may provide valuable insight into this problem, as Meteer has indicated [17]. We have also conducted a psychological experiment to extract criteria for the evaluation and improvement. The collected data is now under analysis. We will feed the result back into the system. At the same time, we will extend the current system, and evaluate its performance with more examples.

Acknowledgments

The authors would like to thank Dr. Christian Matthiessen for his fruitful discussions on realization of lexico-grammar, and the reviewers for their helpful comments on the early version of this paper. The authors owe a great debt to Craig Hunter and Megan Withycombe, who patiently read the draft and contributed to improving the how-to-say of this 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. D. E. Appelt. Planning natural-language referring expressions. In D. D. McDonald and L. Bolc, editors, *Natural Language Generation Systems*, chapter 3, pages 69–97. Springer-Verlag, 1988.
3. R. Dale. The generation of subsequent referring expressions in structured discourse. In M. Zock and G. Sabah, editors, *Advances in Natural Language Generation*, chapter vol. 2, 4, pages 58–75. Ablex Publishing Corporation, 1988.
4. L. Danlos. Conceptual and linguistic decisions in generation. In *the Proceedings of the International Conference on Computational Linguistics*, pages 501–504, 1984.
5. J. Doyle. A truth maintenance system. *Artificial Intelligence*, pages 231–272, 1979.
6. M. Elhadad and K. R. McKeown. Generating connectives. In *the Proceedings of the International Conference on Computational Linguistics*, pages 3:97–101, 1990.
7. 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.
8. E. H. Hovy. *Generating Natural Language under Pragmatic Constraints*. Lawrence Erlbaum Associates, 1988.

9. S. Ishizaki. Generation Japanese text from conceptual representation. In D. D. McDonald and L. Bolc, editors, *Natural Language Generation Systems*, chapter 7, pages 256–279. Springer-Verlag, 1988.
10. 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.
11. 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.
12. 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.
13. W. C. Mann and J. A. Moore. Computer generation of multiparagraph English text. *American Journal of Computational Linguistics*, 7(1):17–29, 1981.
14. 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.
15. M. Maybury. Using discourse focus, temporal focus, and spatial focus to generate multisentential text. In *the Proceedings of the Fifth International Workshop on Natural Language Generation*, pages 70–78, 1990.
16. 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.
17. M. W. Meteer. *The Generation Gap: The Problem of Expressibility in Text Planning.* PhD thesis, University of Massachusetts, 1990.
18. M. M. Meteer (Vaughan) and D. D. McDonald. A model of revision in natural language generation. In *the Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 90–96, 1986.
19. E. Reiter. Generating discriptions that exploit a user's domain knowledge. In R. Dale, C. Mellish, and M Zock, editors, *Current Research in Natural Language Generation*, chapter 10, pages 257–286. Academic Press, 1990.
20. D. R. Scott and C. S. Souza. Getting the message across in rst-based text generation. In R. Dale, C. Mellish, and M Zock, editors, *Current Research in Natural Language Generation*, chapter 3, pages 47–74. Academic Press, 1990.
21. W. C. Wong and R. F. Simmons. A blackboard model of text production with revision. In *the Proceedings of the AAAI Workshop on Text Planning and Realization*, pages 99–106, 1988.
22. 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.

This article was processed using the LaTeX macro package with LMAMULT style