Sentence Compression by Removing Recursive Structure from Parse Tree

Seiji Egawa[1], Yoshihide Kato[2], and Shigeki Matsubara[3]

[1] Graduate School of Information Science, Nagoya University
[2] Graduate School of International Development, Nagoya University
[3] Information Technology Center, Nagoya University

Furo-cho, Chikusa-ku, Nagoya, 464-8601, Japan
[1] egawa@el.itc.nagoya-u.ac.jp

Abstract. Sentence compression is a task of generating a grammatical short sentence from an original sentence, retaining the most important information. The existing methods of removing the constituents in the parse tree of an original sentence cannot deal with recursive structures which appear in the parse tree. This paper proposes a method to remove such structure and generate a grammatical short sentence. Compression experiments have shown the method to provide an ability to sentence compression comparable to the existing methods and generate good compressed sentences for sentences including recursive structures, which the previous methods failed to compress.

Key words: sentence compression, text summarization, phrase structure, recursive structure, maximum entropy method

1 Introduction

Sentence compression is a task of summarizing a single sentence. It is useful for automatic text summarization and other applications such as generating subtitles or reducing messages for mobile devices.

Several sentence compression algorithms have been proposed so far. These algorithms produce a summary of a single sentence, which is called *compression*. Compression should satisfy the following conditions:

- It should be grammatical.
- It should retain the most important information of the original sentence.

In previous works, the problem of sentence compression have been simplified to removing redundant words or phrases from the original sentence. To generate a compression, the algorithms utilize syntactic information, such as phrase structure, dependency structure, part-of-speech and so on. Most of the algorithms only remove some redundant words or phrases, then the compression is a subsequence of the original sentence.

Knight and Marcu have proposed a probabilistic method of removing redundant constituents from the parse tree of the original sentence[2]. The probabilities

of removing constituents are estimated from a compression parallel corpus consisting of the pairs of original sentences and the corresponding compressions. Turner and Charniak have proposed an alternative method to approximate such probabilities without compression parallel corpora to overcome the lack of compression corpora[7]. Unno et al. have proposed a method of using maximum entropy method[1] so that more various features are dealt with, while Knight and Marcu have used only simple PCFG[8]. Vandeghinste and Pan have proposed a method of combining a probabilistic approach like above and a rule-based approach to avoid generating ungrammatical sentences[9].

These methods have only one operation of removing a constituent from a parse tree. However, the operation is not enough to compress any kind of sentences. The parse trees of some compressions have quite different structure from those of the original sentences so that it is impossible to obtain the compressed version of parse trees by removing constituents.

To solve the problem, this paper proposes an operation of transforming parse trees for sentence compression. We focus on recursive structures, which frequently appear in parse trees and represent adjuncts, coordinations, embedded sentences and so on. The operation removes recursive structures from the parse tree while preserving its grammaticality. Our method models sentence compression as a process of removing constituents and recursive structures from the parse tree of an original sentence. The model is probabilistic and learned from a compression parallel corpus. Our method can compress sentences including recursive structures.

Experimental results have shown that our method is comparable with the existing methods and that removing recursive structure from parse trees is effective for compressing certain sentences.

The organization of this paper is as follows: We review the previous methods of removing constituents in section 2. Section 3 describes our method which deal with recursive structures in parse trees for sentence compression. Section 4 presents some experimental evaluation of our method compared to the previous methods. Section 5 concludes this paper and presents future works.

2 Sentence Compression by Removing Constituents

In previous works, given an input sentence l, a compression s is formed by removing words from l. No rearranging words or no adding new words take place. The $2^{|l|}$ compression candidates exist and the problem of sentence compression can be formalized as determining which candidate is the best compression.

Knight and Marcu[2] tackled this problem by presenting a noisy channel model. The method finds the compression s which maximizes the conditional probability $P(s|l)$. The model $P(s|l)$ is decomposed into two models: the source model $P(s)$ and the channel model $P(l|s)$. That is, the compression s' is defined as follows:

$$s' = \underset{s}{\operatorname{argmax}}\, P(s|l) = \underset{s}{\operatorname{argmax}}\, P(s)P(l|s)$$

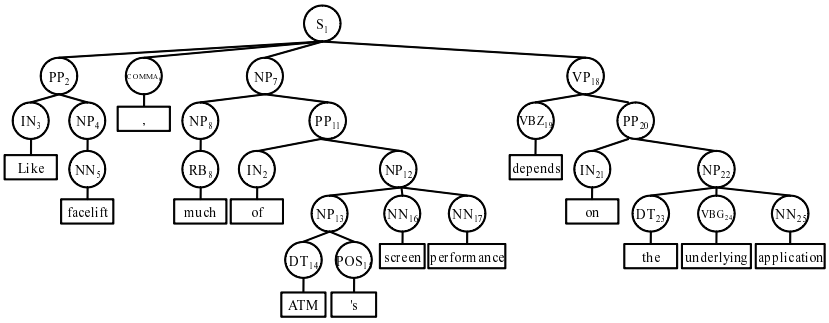

Fig. 1. Parse tree of sentence (1)

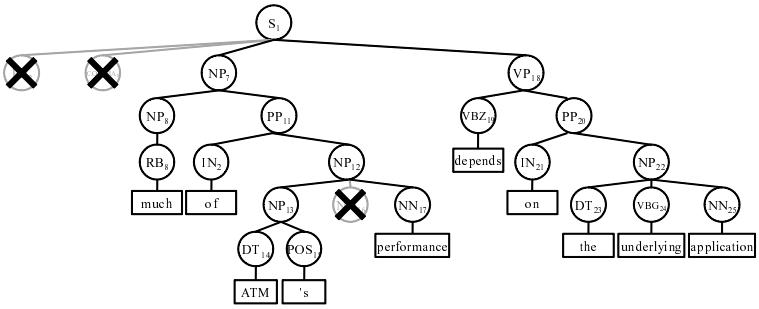

Fig. 2. Parse tree of sentence (2) created from (1) by Knight and Marcu's method

The source model $P(s)$ evaluates the gramaticality of s. The channel model $P(l|s)$ determines which parts in l are redundant.

$P(s)$ and $P(l|s)$ are calculated based on the parse tree. As an example, let us consider the following original sentence (1) and its compression (2):

(1) Like facelift, **much of ATM's** screen **performance depends on the underlying application.**
(2) **Much of ATM's performance depends on the underlying application.**

The original sentence (1) is parsed into a tree shown in Fig. 1. The parse tree of compression (2) is created by removing some constituents from the original tree, that is, removing nodes "$\mathbf{PP_2}$", "$\mathbf{COMMA_6}$" and "$\mathbf{NN_{16}}$". These nodes respectively correspond to word sequences "Like facelift", "," and "screen" which do not appear in compression (2). In this case, the probability $P(s)$ is high because the parse tree of (2) is grammatical.

$P(l|s)$ is learned from a compression parallel corpus consisting of pairs of sentences and compressions. A parse tree is assigned to every sentence and every compression. The method finds the correspondence between the nodes in the

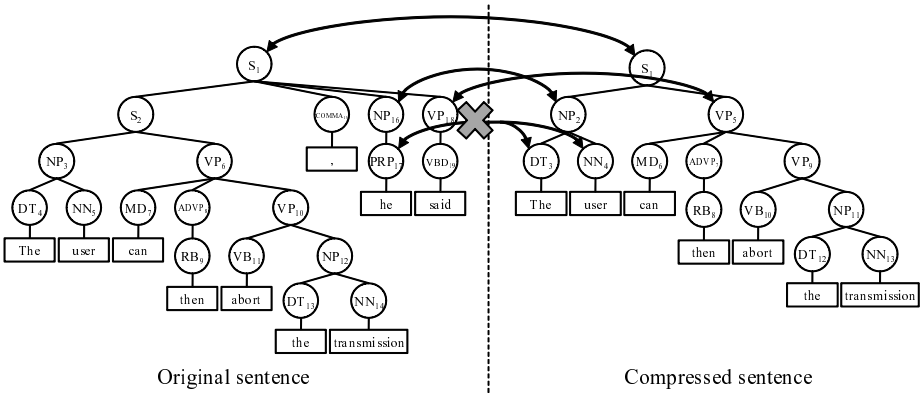

Fig. 3. Mismatch between parse trees of original sentence and compression

original parse tree and the compressed one in top-down fashion, and identifies the constituents removed from the original parse tree. For example, there exists a correspondence between the parse trees shown in Fig. 1 and Fig. 2, and nodes $\mathbf{PP_2}$, $\mathbf{COMMA_6}$ and $\mathbf{NN_{16}}$ are identified with removed constituents.

As the following example shows, however, there is certain cases where the method cannot find the correspondence between original and compressed parse trees. (see Fig. 3).

(3) **The user can then abort the transmission**, he said **.**
(4) **The user can then abort the transmission.**

In this example, the method first finds the correspondence between "$\mathbf{S} \rightarrow \mathbf{NP\ VP}$" in the compressed parse tree and "$\mathbf{S} \rightarrow \mathbf{S\ COMMA\ NP\ VP}$" in the original parse tree. In the next stage, the method finds no correspondence because the child "**PRP**" of **NP** in the original parse tree does not match the children "**DT**" and "**NN**" in the compressed parse tree.

On the contrary, Unno et al.[8] have proposed a method for finding correspondences between both parse trees in a bottom-up fashion. The method parses only original sentences and extracts compressed parse trees from the original parse trees as in Fig. 4. Even though finding the correspondence always succeeds, the compressed parse trees become sometimes ungrammatical. Unno et al. directly estimate probabilities of removing constituents and do not evaluate the grammaticality of the compression.

As an example, let us consider a sentence (5) and its compression (6).

(5) It is likely that **a Macintosh version will be available** soon**.**
(6) **A Macintosh version will be available.**

The parse tree of (5) is shown in Fig.5. To obtain the compression, the method should remove nodes $\mathbf{NP_2}$, $\mathbf{AUX_5}$, $\mathbf{JJ_7}$, $\mathbf{IN_9}$ and $\mathbf{ADVP_{21}}$. Since the removal

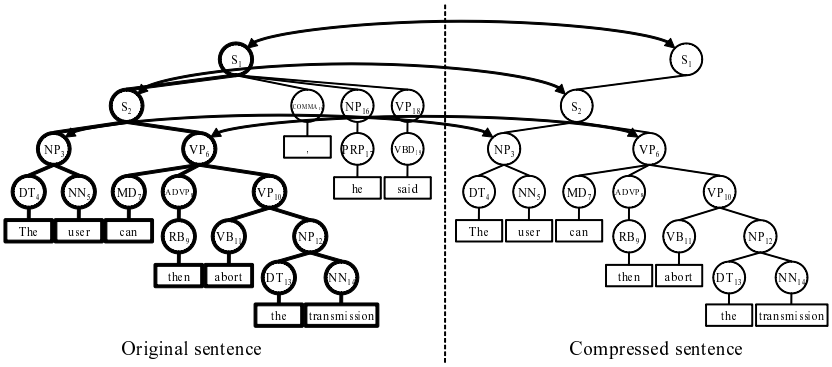

Fig. 4. Matching parse trees of original sentence and compression

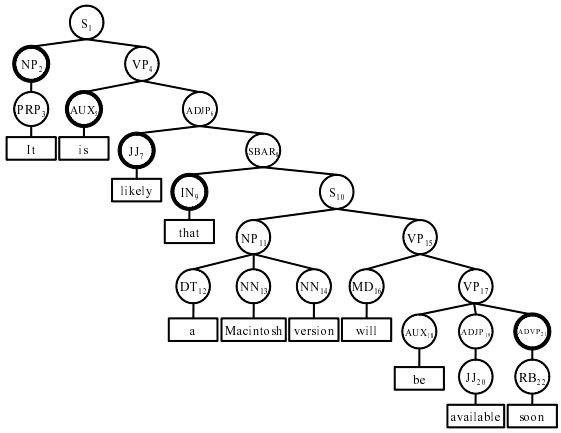

Fig. 5. Parse tree difficult to compress by previous methods

operations are assumed to be independent, it is difficult to compress such sentence. The same can be said for a sentence (7) and its compression (8).

(7) **The CAKE in CAKEware** is an acronym which **stands for computer-assisted knowledge engineering.**

(8) **The CAKE in CAKEware stands for computer-assisted knowledge engineering.**

3 Method of Removing Recursive Structures

This section describes our algorithm for sentence compression. We introduce a new operation: removing recursive structures from parse trees. At first, we describes the basic idea of our approach.

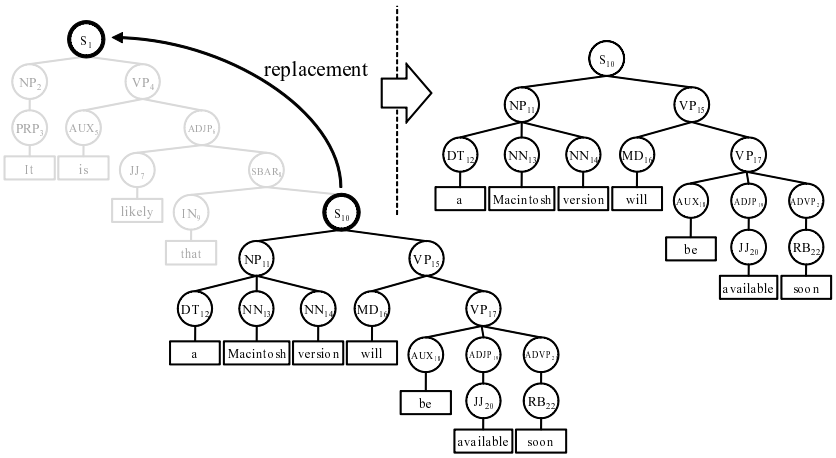

Fig. 6. Removing recursive structure for sentence compression

As an example, let us consider the parse tree shown in Fig. 5. The parse tree has a recursive structure in which node $\mathbf{S_1}$ includes node $\mathbf{S_{10}}$. If we replace $\mathbf{S_1}$ with $\mathbf{S_{10}}$, we obtain a parse tree (see Fig. 6). The parse tree is grammatical because $\mathbf{S_{10}}$ plays the same syntactic role as $\mathbf{S_1}$. Our method introduces such operation. This operation can capture the dependence among removed constituents. For example, the operation captures the dependence between $\mathbf{NP_2}$, $\mathbf{AUX_5}$, $\mathbf{JJ_7}$ and $\mathbf{IN_9}$ removals, because they are removed by one operation.

In order to confirm the validity of this method, we investigated how often such operation occurs in human compression. It occurs 579 times in compressing 943 sentences which are included in the compression parallel corpus used in Knight and Marcu[2]. 110 operations out of 579 are particularly difficult to emulate for previous methods because there are some other nodes on the path from the ancestor node to the descendant node with the same syntactic category and the multiple nodes have a dependence.

Although the problem still remains whether important information of the original sentence is retained or not, it can be solved by training probabilities of removal operations from a compression parallel corpus.

3.1 Elementary Unit

This section gives some definitions for explanations of our method.

Definition 1 (Recursive node) *Let T be a parse tree, η be a node in T and X be the label of η. We call η recursive if there exists a node η' satisfying the following conditions:*

1. *η' is a descendant of η.*
2. *The label of η' is X.*

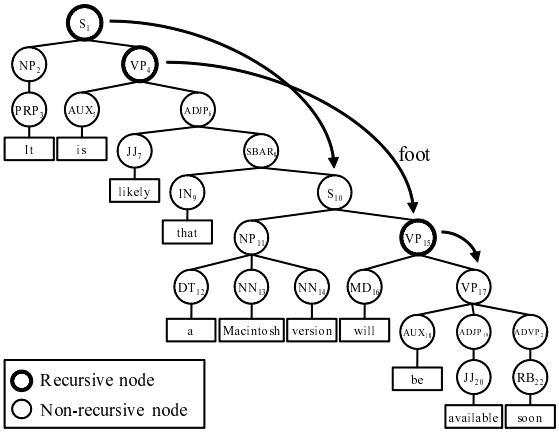

Fig. 7. Recursive node and non-recursive node

We call η non-recursive if η is not recursive.

For example, there are three recursive nodes, $\mathbf{S_1}$, $\mathbf{VP_4}$ and $\mathbf{VP_{15}}$ in Fig. 7. The node η' which is the nearest to η is called *foot*. The path from η to η' is called *minimal recursive path* (*MRP*). We say that the root of the MRP is η. An MRP corresponds to a recursive structure in a parse tree. Fig. 8 shows the MRP whose root is $\mathbf{S_1}$.

3.2 Removing Elementary Units from Parse Tree

Our proposed algorithm removes constituents and MRPs from the parse tree of an input sentence to generate the compression. We use two types of operations:

removeConst operations remove a non-recursive node η and all descendants of η from parse tree.

removeMRP operations remove a recursive node η and all descendants of η, and replace the position of η with the foot of η

By applying these operations to the parse tree of the input sentence, we can obtain the compressed version of it. However, we need to choose the operations to generate a compression which is grammatically correct and preserves the important information of the original sentence. For this purpose, our method learns the process of applying operations from a compression parallel corpus.

The compression parallel corpus consists of pairs of original sentences and their compressions. Our method first assigns the parse tree only to original sentences. For each pair of original parse tree and its compression, we determine which operations are applied to the parse tree. Next, we count the frequency of applying operations and estimate the probabilities.

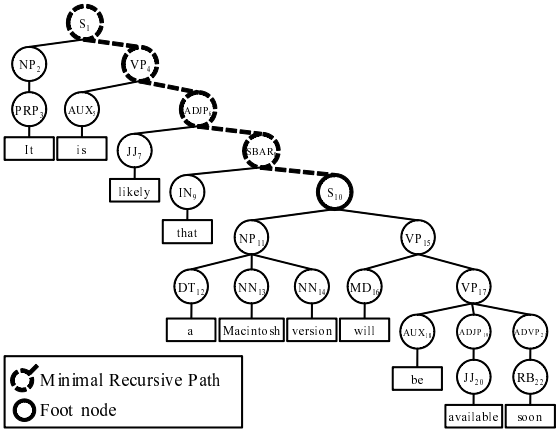

Fig. 8. Minimal recursive path

Our method determines which operations are applied as follows: For each node in the parse tree, the operation is applied, if it does not remove any words in the compression. The operations are applied in a top-down fashion.

As an example, let us consider the input sentence (5) and its compression (6). Fig. 9 shows the parse tree of (5). For each terminal node, it is marked with bold line if it exists in the compression (6).

At first, the procedure tries to apply removeMRP since the root $\mathbf{S_1}$ is recursive. Because the word sequence "It is likely that", which is removed by the operation, do not overlap the compression, so this operation is applied to $\mathbf{S_1}$. Note that nodes $\mathbf{NP_2}$, $\mathbf{PRP_3}$, $\cdots$, $\mathbf{IN_9}$ are removed by the operation to $\mathbf{S_1}$. Next $\mathbf{S_{10}}$ is non-recursive. Applying removeConst, all words in the original tree are removed. Because some of these words exist in compression (6), this operation is not applied. For each node from $\mathbf{NP_{11}}$ to $\mathbf{NN_{14}}$, removeConst operation is not applied for the same reason. $\mathbf{VP_{15}}$ is recursive. The removeMRP operation is not applied for this node because this operation deletes the word "will", which appears in the compression (6). For each node from $\mathbf{MD_{16}}$ to $\mathbf{JJ_{20}}$, which are non-recursive, the removeConst operations are not applied. $\mathbf{ADVP_{21}}$ is non-recursive. The removeConst operation is applied, since its descendant, "soon", does not exist in the compression (6).

As the above, applying each corresponding operation to $\mathbf{S_1}$ and $\mathbf{ADVP_{21}}$, we obtain the parse tree of (6). This tree (shown in Fig. 10) is grammatical as opposed to the one generated by the previous method.

After determining, for each node, whether the operation is applied or not, we estimate its probability by maximum entropy method using the features:

a. the removal operation type (removeConst or removeMRP)
b. the current node label
c. the parent node label

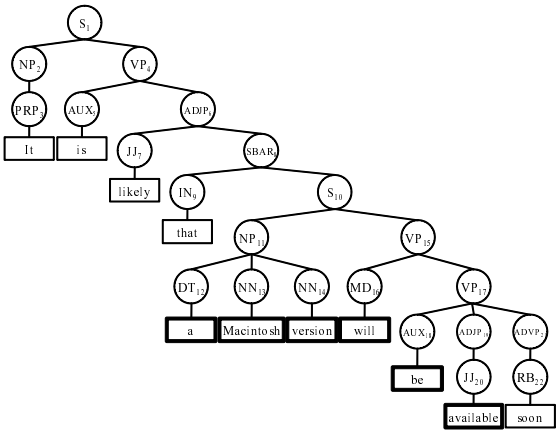

Fig. 9. Parse tree of sentence (5)

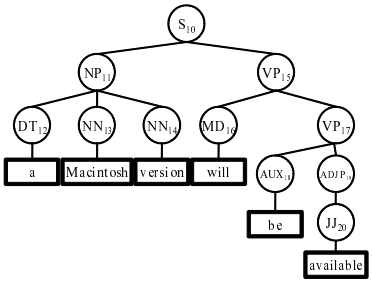

Fig. 10. Parse tree of sentence (6) by our method

d. the daughter node labels
e. the left sibling node labels and which siblings are removed (only if the operation type is removeConst)
f. the node labels on MRP
g. the daughter node labels of nodes on MRP
h. the foot node label

3.3 Probabilistic Sentence Compression Model

This section describes how to calculate the compression probability by using removal probabilities.

We define the probability of compressing a long sentence l to a short sentence s as the probability of generating the compressed version of the parse tree from the parse tree of l by removal operations. The probability is calculated by the product of the removing probabilities, that is,

$$P(s|l) = \prod_{\eta \in N} P(a_\eta|\eta, l)$$

where N is the set of nodes remaining in the parse tree of s or to which operations are applied. N does not have any node which is removed by applying a removal operation to an ancestor. a_η is 1 if an operation is applied to η and 0 if not.

For example, the probability of the compressing sentence (5) to the sentence (6) is $P(1|\mathbf{S_1})\ P(0|\mathbf{NP_{11}})\ P(0|\mathbf{DT_{12}})\ P(0|\mathbf{NN_{13}})\ P(0|\mathbf{NN_{14}})\ P(0|\mathbf{VP_{15}})\ P(0|\mathbf{MD_{16}})\ P(0|\mathbf{VP_{17}})\ P(0|\mathbf{AUX_{18}})\ P(0|\mathbf{ADJP_{19}})\ P(0|\mathbf{JJ_{20}})\ P(1|\mathbf{RB_{21}})$. For simplicity, we abbreviate l.

3.4 Computing Scores

Using the model described in the previous section, we compute the compression score for every compression candidate s.

$$Score(s) = length(s)^\alpha \cdot \log P(s|l)$$

This score is proposed by Unno et al. and the compression model $P(s|l)$ is replaced with ours. α is a length parameter which controls the average length of outputs. Our method formalize the sentence compression problem as finding the compression which maximizes the score.

4 Experiments

To evaluate our algorithm, we conducted experiments. We use the compression parallel corpus used in Knight and Marcu[2]. This corpus consists of sentence pairs extracted from the Ziff-Davis corpus, which includes news articles on computer products. 32 sentence pairs in this corpus are used for evaluation in Knight and Marcu's experiment. We also use these sentences as a test set. Our model is trained on 943 sentences pairs, where each word in a compression corresponds to only one word in the original sentence, in the rest of the compression corpus.

4.1 Comparison with Original Results

At first, we compared our model with noisy-channel model in Knight and Marcu[2]. We evaluated both methods using four measures, compression rate, word F-measure, word bigram F-measure and BLEU score[6]. These measures except compression rate represent the similarity between sentences and we evaluate a compression with the degree of similarity to human compression. The BLEU score is a measure for machine translation quality. We used from unigram to 4-gram precisions for the BLEU score as in Unno et al.[8]. The value of the length parameter α for our method is determined by using 50 sentence pairs randomly extracted from training set. In this experiment, $\alpha = -0.43$.

The results are shown in Table 1. Our method achieved comparable accuracy with Knight and Marcu's method.

Table 1. Comparison with Knight and Marcu

Method	compression	F-measure	bigram F-measure	BLEU
Knight and Marcu	70.4%	71.9%	58.5%	48.9%
Our method	50.7%	68.4%	58.5%	52.8%
Human	53.3%			

4.2 Examples of Compressions

Table 2 shows three sentences with compressions by human, the previous methods and our method. These sentences are used in the literature [8]. The first sentence is accurately compressed by a bottom-up method of Unno et al. while Knight and Marcu failed. Our method has also generated a correct compression. The second sentence has some recursive structures in its parse tree and both previous methods can not correctly compress it. Removing one of the recursive structures, Our method generated proper compression. Although all of the compressions generated for the third sentence are different from the one generated by human, our method seems to be superior to the others from the viewpoints of grammaticality and meaning.

5 Conclusion

We proposed a probabilistic method for sentence compression to remove recursive structures in the parse trees of original sentences. While recursive structures frequently appear in the parse tree, the previous methods do not deal with such the structure. Our method accurately compress such sentences applying a removal operation of the recursive structure. The experimental results show that our model has comparable power for sentence compression with other methods, and correctly compresses certain sentences which those methods cannot deal with. Evaluating our method only using three measures in this paper, we will evaluate our method by human judgments in terms of grammaticality and retention of important information.

Acknowledgements. The authors would like to thank Prof. Kevin Knight and Prof. Daniel Marcu for providing their parallel corpus and the experimental results. This research was partially supported by the Grant-in-Aid for Scientific Research (B) (No. 20300058) of JSPS.

References

1. Berger, A.L., Della Pietra, V.J., Della Pietra, S.A.: A Maximum Entropy Approach to Natural Language Processing. Computational Linguistics, vol. 22, num. 1, pp. 39–71 (1996)

Table 2. Examples of compressions

Original	The user can then abort the transmission, he said.
Human	The user can then abort the transmission.
Knight	The user can abort the transmission said.
Unno	The user can then abort the transmission.
Our method	The user can then abort the transmission.
Original	It is likely that both companies will work on integrating multimedia with database technologies.
Human	Both companies will work on integrating multimedia with database technologies.
Knight	It is likely that both companies will work on integrating.
Unno	It is will work on integrating multimedia with database technologies.
Our method	Both companies will work on integrating multimedia with database technologies.
Original	A file or application "alias" similar in effect to the MS-DOS path statement provides a visible icon in folders where an aliased application does not actually reside.
Human	A file or application alias provides a visible icon in folders where an aliased application does not actually reside.
Knight	A similar in effect to MS-DOS statement provides a visible icon in folders where an aliased application does reside.
Unno	A file or application statement provides a visible icon in folders where an aliased application does not actually reside.
Our method	A file or application "alias" similar in effect to the MS-DOS path statement provides a visible icon in folders.

2. Knight, K., Marcu, D.: Statistics-Based Summarization – Step One: Sentence Compression. In: AAAI/IAAI-2000, pp. 703–710, MIT Press (2000)
3. Knight, K., Marcu, D.: Summarization beyond Sentence Extraction: A Probabilistic Approach to Sentence Compression. Artificial Intelligence, vol. 139, pp. 91–107 (2002)
4. Mani, I.: Automatic Summarization.: John Benjamins, Philadelphia (2001)
5. Nguyen, M.L., Horiguchi, S., Shimazu, A., Ho, T.B.: Example-Based Sentence Reduction Using the Hidden Markov Model. ACM Trans. on Asian Language Information Processing, vol. 3, num. 2, pp. 146–158 (2004)
6. Papineni, K., Roukos, S., Word, T., Zhu, W.J.: BLEU: A Method for Automatic Evaluation of Machine Translation. In: ACL-2001, pp. 311–318, ACL, Morristown (2001)
7. Turner, J., Charniak, E.: Supervised and Unsupervised Learning for Sentence Compression. In: ACL-2005, pp. 290–297, ACL, Morristown (2005)
8. Unno, Y., Ninomiya, T., Miyao, Y., Tsujii, J.: Trimming CFG Parse Trees for Sentence Compression Using Machine Learning Approaches. In: COLING/ACL-2006, pp. 850–857, ACL, Morristown (2006)
9. Vandeghinste, V., Pan, Y.: Sentence Compression for Automated Subtitling: A Hybrid Approach. In: ACL-2004 Workshop on Text Summarization, pp. 89–95, ACL, Morristown (2004)