Sentence Compression by Structural Conversion of Parse Tree

Seiji Egawa
Graduate School of Information Science
Nagoya University
Furo-cho, Chikusa-ku, Nagoya,
464-8601, Japan
egawa@el.itc.nagoya-u.ac.jp

Yoshihide Kato
Graduate School of International Development
Nagoya University
Furo-cho, Chikusa-ku, Nagoya,
464-8601, Japan
yosihide@gsid.nagoya-u.ac.jp

Shigeki Matsubara
Information Technology Center
Nagoya University
Furo-cho, Chikusa-ku, Nagoya,
464-8601, Japan
matubara@nagoya-u.jp

Abstract

Sentence compression is the task of generating a grammatical short sentence from an original sentence, retaining important information. The existing methods of only removing the constituents in the parse tree of an original sentence cannot emulate human compression that changes structures of the parse tree. This paper proposes a method to remove recursive structures, one example of such structural conversions, and generate a grammatical short sentence. In order to remove a recursive structure, our method detects the constituents forming the structure and removes them as a unit. Compression experiments have shown that our method generates more grammatical compressed sentences than the previous method.

1. Introduction

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

The output of sentence compression is called the *compression*. A compression should satisfy the following conditions:

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

Several sentence compression algorithms have been proposed so far [3, 4, 5, 6, 8, 9]. In previous works, the task of sentence compression has been simplified to removing redundant words or phrases from the original sentence, then the compression is a subsequence of the original sentence. To generate a compression, most of the algorithms only remove some redundant words or phrases based on parse trees.

Knight and Marcu have proposed a probabilistic method of removing redundant constituents from the parse tree of the original sentence [5]. The probabilities of removing constituents are estimated from a compression parallel corpus consisting of the pairs of original sentences and the corresponding compressions.

While Knight and Marcu have used only simple PCFG, Unno et al. have proposed a method of using maximum entropy method [1] so that more various features, such as its parent node and sibling nodes, are treated [9].

These methods have only one operation of removing a constituent from a parse tree in common. However, the operation is not sufficient for compressing every kind of sentence. The parse trees of some compressions have quite different structures from those of the original sentences so that it is impossible to obtain the compressed version of parse trees by simply removing constituents.

To solve the problem, this paper proposes a method 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. We introduce the operation of remov-

ing recursive structures from 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.

Experimental results have shown that our method is superior to the existing method in terms of grammaticality.

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 removes recursive structures in parse trees for sentence compression. Section 4 presents experimental evaluations of our method compared to the previous methods. Section 5 concludes this paper and presents future works.

2 Sentence Compression by Removing Constituents

In most previous works, given an input sentence l, a compression s is formed by removing words from l, that is, no rearranging words or no adding of new words takes place. Although the $2^{|l|}$ compression candidates exist, most of them are not grammatical or do not preserve important pieces of information. Then, the task of sentence compression can be formalized as determining which candidate is the best compression.

Knight and Marcu [5] have proposed a parse-tree-based method for sentence compression. This method parses an input sentence and generates a parse tree of the compression by removing constituents from the tree of the input sentence. To determine which constituents are removed, they present a noisy-channel model. That is, the compression of l, s', is defined as follows:

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

$P(s)$ is calculated based on CFG and evaluates the grammaticality of s. $P(l|s)$ evaluates the redundancies of words and phrases which are removed in compression from l to s. It is learned from a compression parallel corpus. Decomposing $P(s|l)$ into $P(s)$ and $P(l|s)$, the problems of grammaticality and preserving information are dealt with independently. In what follows, we focus on training from a compression parallel corpus to discuss the problematic point in Knight and Marcu. In their method, the original sentence and the corresponding compression in a corpus are parsed and nodes in both of trees are matched in a top-down fashion. The nodes in the original tree which have no corresponding node in the compression tree are considered to be redundant, and $P(l|s)$ are estimated from that frequency.

As an example, let us consider the following original sentence (1) and its compression (2) in a compression parallel corpus. The removed words are represented with bold fonts:

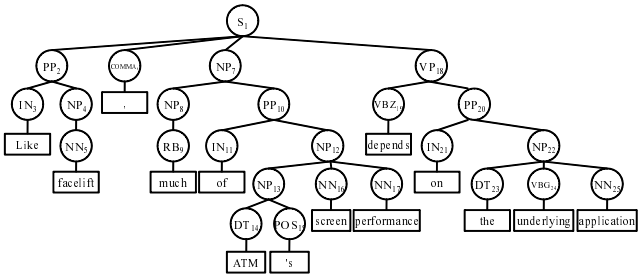

Figure 1. Parse tree of sentence (1)

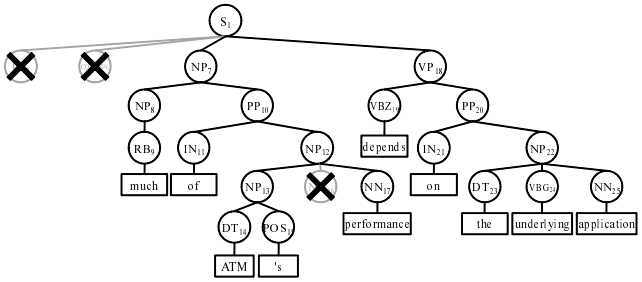

Figure 2. Parse tree of sentence (2)

(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.

Figure 1 shows the tree of the original sentence (1) and Figure 2 the compression (2). Matching both trees, we find that nodes $\mathbf{PP_2}$, $\mathbf{COMMA_6}$ and $\mathbf{NN_{16}}$ have no corresponding node and therefore they are redundant. In this way, we can determine the redundant constituents.

As the following example shows, however, this method cannot find the correspondence between original and compressed parse trees in the case where the process of sentence compression does not consist of only removing constituents. As an example, let us consider the following original sentence (3) and its compression (4):

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

Figure 3 shows the trees of these sentences. In this example, the method first finds the correspondence between "S → **S COMMA** NP VP" in the original parse tree and "S → NP VP" in the compressed parse tree. In the next stage, however, the method finds no correspondence between "NP →

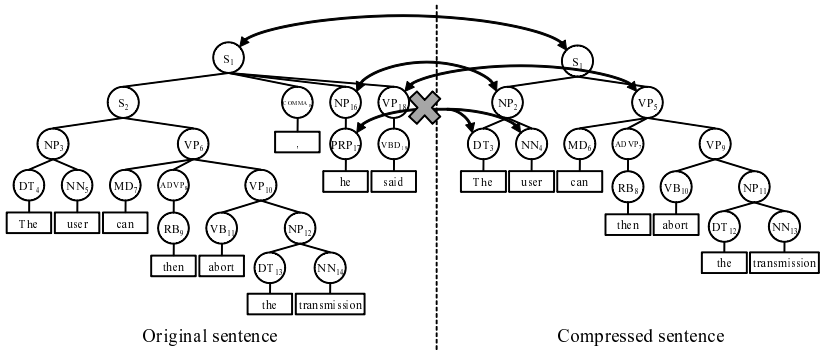

Figure 3. Mismatch between parse trees of original sentence and compression

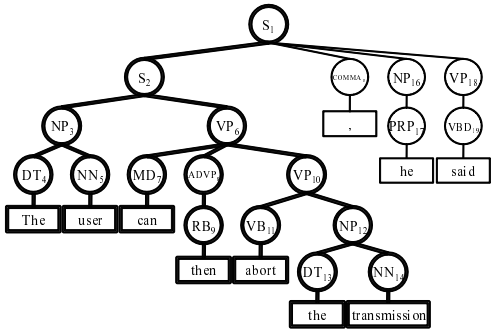

Figure 4. Extraction of parse tree of compression from that of original sentence

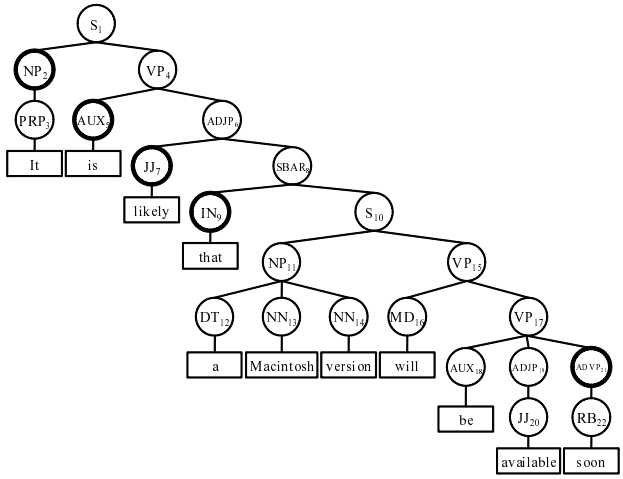

Figure 5. Parse tree difficult to compress by previous methods

PRP" and "NP $\rightarrow$ *DT NN*" because of a mismatch between their right sides.

On the contrary, Unno et al. [9] have proposed a method of parsing only the original sentence and finding correspondences between the parse tree and the compressed sentence in a bottom-up fashion. This method extracts compressed parse trees from the original parse trees.

As an example, let us consider sentences (3) and (4) again. Figure 4 shows the parse tree of the sentence (3) where the words in the compression (4) are marked with double lines. We can extract the parse tree of the compression by marking nodes recursively if they have at least one marked child nodes.

Even though finding the correspondence always succeeds, when the compressing process does not consist of only removing constituents, the compressed parse trees become ungrammatical. As a typical example, let us consider the original 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 Figure 5. As a result of bottom-up matching, the nodes $\mathbf{NP_2}$, $\mathbf{AUX_5}$, $\mathbf{JJ_7}$, $\mathbf{IN_9}$ and $\mathbf{ADVP_{21}}$ are considered as redundant nodes. The compressed tree extracted from the original sentence tree is ungrammatical. Even though the method can learn the compressing process, it is the unnatural process in which the constituents $\mathbf{NP_2}$, $\mathbf{AUX_5}$, $\mathbf{JJ_7}$ and $\mathbf{IN_9}$ are removed independently. As a result, the method has possible danger of generating the ungrammatical compression such as removing only $\mathbf{JJ_7}$.

3 Sentence Compression by Removing Recursive Structures

In the previous section, we described the sentence compression which cannot be resolved by existing methods of removing only constituents from parse trees. This section

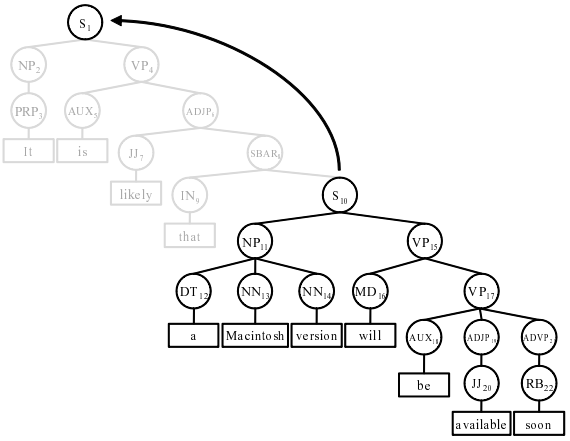

Figure 6. Removing recursive structure in sentence compression

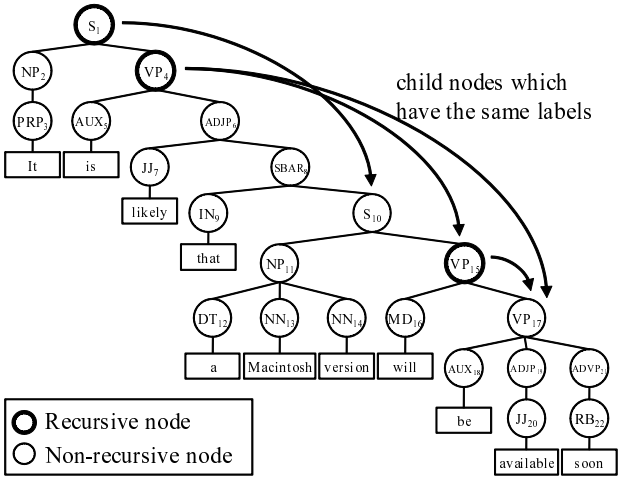

Figure 7. Recursive node and non-recursive node

describes our algorithm for sentence compression which not only removes constituents but also transforms original trees. We introduce a new operation: removing recursive structures from parse trees. At first, we describe the basic idea of our approach.

If we obtain a grammatical parse tree of the compression from the original parse tree through a certain process, the process is natural. In the example of the previous section, we can obtain the grammatical parse tree of the sentence (6), replacing $\mathbf{S_1}$ with $\mathbf{S_{10}}$ as shown in Figure 6. If we introduce such an operation, we can model the natural process for sentence compression. The words "It is likely that" are removed as one phrase, and then incorrect processes such as removing only $\mathbf{JJ_7}$ are not learned. The key point is that $\mathbf{S_1}$ and $\mathbf{S_{10}}$ have the same syntactic category. That is, a node is grammatically replaced by the node which has the same category. This is the reason why we focus on recursive structures.

3.1 Elementary Unit

In this section, we define a new elementary unit for removing a recursive structure. Our method determines whether or not to remove each elementary unit. As a preparation for defining an elementary unit, we first define a recursive node.

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.*

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 Figure 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*). η is called the root of the MRP. Figure 8 shows the MRP whose root is $\mathbf{S_1}$.

Removing an MRP corresponds to removing a recursive structure in a parse tree. In our method, the removal elementary units are the constituent and the MRP.

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

removeMRP operation replace a recursive node η with corresponding foot node η'.

By applying these operations to the parse tree of the input sentence, we can obtain the compressed version of the parse tree. The algorithm applies removeConst operation to non-recursive nodes and removeMRP to recursive nodes. However, we need to choose the operations properly 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.

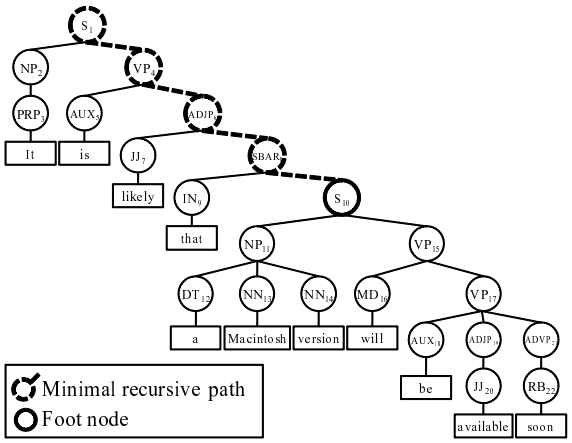

Figure 8. Minimal recursive path

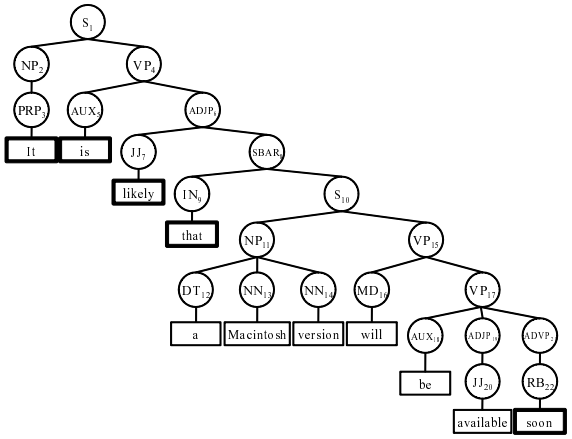

Figure 9. Parse tree of sentence (5)

Our method first assigns the parse tree only to original sentences [2]. For each pair of original parse trees and their compression, we determine which operations are applied to the parse tree. Next, we count the frequency of applying operations and estimate the probabilities.

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 word 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). Figure 9 shows the parse tree of (5). For each terminal node, it is marked with bold line if it does not exist in the compression (6). At first, the procedure tries to apply removeMRP to the root node $\mathbf{S_1}$ since $\mathbf{S_1}$ is recursive. Because the words "It", "is", "likely" and "that", which is removed by the operation, do not overlap the compression, 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

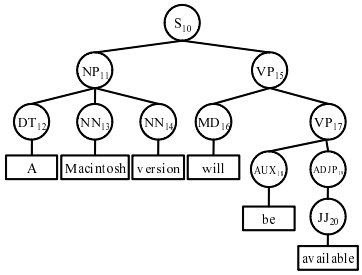

Figure 10. Parse tree of sentence (6) compressed by our method

$\mathbf{S_1}$. Next $\mathbf{S_{10}}$ is non-recursive. Applying removeConst, all words in the original tree are removed. Because some of these words exist in the 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 to this node because this operation deletes the word "will", which appears in the compression. 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.

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

3.3 Estimating Probabilities by Maximum Entropy Method

After determining whether the operation is applied to each node or not, we estimate its probability by using the maximum entropy method. We use the following features:

a. removal operation type (removeConst or removeMRP)

b. current node label

c. parent node label

d. daughter node labels

e. left sibling node labels and which siblings are removed (only if the operation type is removeConst)

f. node labels on MRP

g. daughter node labels of nodes on MRP

h. foot node label

3.4 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 subset of nodes in the parse tree of l, which includes the nodes not applied removal operation to (= nodes remaining in the tree of s) and the nodes applied removal operation to. N does not include any node which is removed by applying a removal operation to one of its ancestors. a_η is 1 if an operation is applied to η and 0 if not.

According to this model, the probability of compressing the sentence (5) to the sentence (6) is calculated as follows. For simplicity, we abbreviate l.

$$P(1|\mathbf{S_1})\, P(0|\mathbf{S_{10}})\, P(0|\mathbf{NP_{11}})\, P(0|\mathbf{DT_{12}})\, P(0|\mathbf{NN_{13}})$$
$$P(0\,|\,\mathbf{NN_{14}})\; P(0\,|\,\mathbf{VP_{15}})\; P(0\,|\,\mathbf{MD_{16}})\; P(0\,|\,\mathbf{VP_{17}})$$
$$P(0|\mathbf{AUX_{18}})P(0|\mathbf{ADJP_{19}})P(0|\mathbf{JJ_{20}})P(1|\mathbf{ADVP_{21}})$$

3.5 Scoring

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. We should normalize compression probabilities using lengths of compressions because the shorter the compression is, the higher the probability becomes. α is a parameter for such normalization. Our method formalizes the sentence compression problem as finding the compression which maximizes the score.

4 Experiments

To evaluate our compression algorithm, we conducted experiments. We use the compression parallel corpus used in Knight and Marcu [5]. This corpus consists of sentence pairs extracted from the Ziff-Davis corpus, which includes news articles on computer products and their summaries. 32 sentence pairs in this corpus are used for evaluation in Knight and Marcu's experiments. 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.

Table 1. Results of human judgments

	Compression	Grammar	Importance
Knight	70.4%	4.05	3.80
Our method	50.7%	4.20	3.41
Human	53.3%	4.44	3.67

We compared our model with noisy-channel model of Knight and Marcu by human judgments. We determined the value of the length parameter α by adjusting experiments on 50 sentence pairs which are extracted from the training set at random. That is $\alpha = -0.43$.

We presented each original sentence in the test corpus, and three compressions of it, to four judges: the compressions generated by humans, the outputs of Knight and Marcu's algorithm and the outputs of our algorithm. The judges thought of all compressions as the ones generated automatically.

The results are shown in Table 1. Our method is superior to Knight and Marcu's method in terms of grammaticality of the compressions. Although Knight and Marcu's method is better at retaining information than human compression and our method, it is significantly affected by compression rates, therefore, we cannot make a simple comparison.

Table 2 shows three sentences with compressions by humans, the previous methods and our method. These sentences are used in the literature [9]. 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 humans, 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 structures. Our method accurately compresses such sentences by applying a removal operation of recursive structures.

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
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.

The experimental results have shown that our method compresses sentences more grammatically than the previous methods of removing only constituents from parse trees.

Even though we adjusted the compression rate of proposed method to be equivalent to that of human compression in this paper, the compression rate has an affect on the preservation of information, therefore we will research their relationship.

Acknowledgment

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. 20300092) of JSPS and by The Asahi Glass Foundation.

References

[1] A. L. Berger, V. J. Della Pietra, and S. A. Della Pietra, "A Maximum Entropy Approach to Natural Language Processing, " Computational Linguistics, vol. 22, num. 1, pp.39-71, 1996.

[2] E. Charniak, "A Maximum-Entropy-Inspired Parser, " Proc. 6th ANLP, pp.132-139, 2000.

[3] J. Clarke, and M. Lapata, "Models for Sentence Compression: A Comparison across Domains, Training Requirements and Evaluation Measures, " Proc. 44th ACL, pp.377-384, 2006.

[4] H. Jing, "Sentence Reduction for Automatic Text Summarization, " Proc. 6th ANLP, pp.310-315, 2000.

[5] K. Knight, and D. Marcu, "Statistics-Based Summarization – Step One: Sentence Compression, " Proc. 17th AAAI, pp. 703-710, 2000.

[6] K. Knight, and D. Marcu, "Summarization beyond Sentence Extraction: A Probabilistic Approach to Sentence Compression, " Artificial Intelligence 139, pp.91-107, 2002.

[7] I. Mani, Automatic Summarization, John Benjamins, 2001.

[8] J. Turner, and E. Charniak, "Supervised and Unsupervised Learning for Sentence Compression, " Proc. 43rd ACL, pp.290-297, 2005.

[9] Y. Unno, T. Ninomiya, Y. Miyao, and J. Tsujii, "Trimming CFG Parse Trees for Sentence Compression Using Machine Learning Approaches, " Proc. 44th ACL, pp.850-857, 2006.