Rewriting Saves Extracted Summaries

Satoshi Sato
JAIST / PRESTO, JST
Asahidai 1-1, Tatsunokuchi, Nomi
Ishikawa, 923-1292, JAPAN
sato@jaist.ac.jp

Madoka Sato
Kanazawa Gakuin University
Suemachi 10, Kanazawa
Ishikawa, 920-1392, JAPAN
madoka@kanazawa-gu.ac.jp

Abstract

In automated editing, text summarization plays an important role when creating the digests of contents; the digest helps users find the desired information quickly. In our automated editing system that makes FAQ-like information packages from the USENET articles, summaries are extracted from original articles. Extracted summaries are useful for selecting the desired information, however, they can be improved into more appropriate summaries to the package. In this paper, we discuss the limitations of the current summary-extraction method, and propose employment of rewriting as a solution.

Introduction

Automated editing (Sato 1997; Sato & Sato 1997) is the new and ambitious idea that the automated system edits information packages from electronic materials. Our first application aimed to edit Frequently Asked Questions (FAQ) from the USENET articles automatically; at this stage, we made the system that generates Questions and Answers Package (QA-Pack)—a substitute of FAQ—that is updated everyday.

The basic structure of QA-Pack is the classification tree (concept tree), in which a node represents a classification concept; that is, QA-Pack consists of hierarchically classified threads[1]. QA-Pack provides two major navigation mechanisms: the table-of-contents and a digest page per node (concept).

Figure 1 shows the table of contents of Sun QA-Pack in English[2]. The table-of-content page shows the first two levels of the concept tree, and provides users with its overview. Figure 2 shows a digest for a concept—*serial port*. This type of pages is called *node page* because it corresponds to a node of the concept tree. In this page, each thread is represented by a group of a headline and a summary. A headline denotes the thread's topic and a summary complements the headline; a headline has a hyperlink to the related thread page (Figure 3).

To generate QA-Pack from the USENET articles, multitudinous editing works are needed. Among them, there are two central tasks (Figure 4).

1. Determine what concept node a thread should belong to.
2. Generate a headline and a summary from each thread.

Obviously, the first task is an issue of text classification; the second task is an issue of text summarization.

In this paper, we discuss text summarization in the automated editing system of QA-Pack. First we overview the summary-extraction method of the current system. Then, we discuss the limitations of the method, and propose a solution.

Summary Extraction in QA-Pack Editing

The word *summarize* is too general to implement a summarization module; before implementation, we have to clarify what kind of summary is requested in the specific application. Usually, (1) what we use summaries for, and (2) how we use summaries, are two basic questions that we must find the answer at this stage.

For our application, we first designed the style of the node page (Figure 2), where summaries are used. The purpose of the node page is to provide an overview of the questions or the problems related to certain specific concept; the node page helps readers find the similar

[1] A thread is a group of articles that share the same subject line (Subject:) in their headers. In Q&A newsgroup, such as comp.sys.sun.admin and fj.sys.sun, the first article of a thread is a question article about Sun workstations and the rest are response articles.

[2] There is also a Japanese Sun QA-Pack. The English QA-Pack is made from articles of comp.sys.sun.admin and the Japanese QA-Pack is made from articles of fj.sys.sun. Both QA-Packs are open at http://www-sato.jaist.ac.jp:8000/faq/.

File Edit View Go Bookmarks Options Directory Window Help

Sun QA-Pack

Questions and Answers about Sun Workstations from comp.sys.sun.admin
June 25, 1997. Edited by our automatic-editing system, *APG v3.30*

[Table of Contents | Complete TOC | Section Finder | How to Use | About QA-Pack | Mail to Us]

Topic (582)	Hardware (427)	Software (983)	Misc (604)
Audio (3)	CD-ROM (48)	Command (37)	
Backup (146)	Floppy Disk (4)	Editor (11)	
Boot (91)	Hard Disk (118)	Language (50)	
Diskless (3)	Keyboard (10)	Library (14)	
Japanese (1)	Memory (18)	Network Software (423)	
Mail/Sendmail (82)	MO (10)	OS (314)	
Multimedia (2)	Modem (20)	Software Tools (14)	
Netnews (9)	Monitor (49)	X (79)	
Porting Software (2)	Mouse (7)		
PPP (45)	Network Hardware (56)		
Printing/Printer (87)	Parallel Port (6)		
Postscript (10)	SCSI (33)		
Remote Login (44)	Serial Port (16)		
Shutdown (8)			
System Trouble (18)			
Web (26)			

Figure 1: Table-of-contents page of Sun QA-Pack. This page shows the first two levels of the classification tree.

File Edit View Go Bookmarks Options Directory Window Help

Sun QA-Pack: Questions and Answers about Sun Workstations from comp.sys.sun.admin
June 25, 1997. Edited by our automatic-editing system, *APG v3.30*
[Table of Contents | Complete TOC | Section Finder | How to Use | About QA-Pack | Mail to Us]

Go to General [Hardware (427) < Top]

Serial Port Japanese

Questions/Problems:

- **Console routing** -- *SunOS 4.1.3 / SS5* -- Apr 19, 96 (91)
  What needs to be done to route the messages that go to the console window in a *SS5* runing *SunOS 4.1.3* to an asyncronous port lets say ttya or ttyb.
- **Routing console to asyncronous port** -- Apr 18, 96 (20)
  How do you route the console window to an asyncronous port, ttya or ttyb.
- **Need *IPX* Serial Port Pinout** -- Jun 22, 97 (820)
  I just bought a used *IPX* and have been trying to get it working. I have no monitor so I've been trying to get it going through the serial port.
- **Setting serial port speed on *Solaris 2.5.1*** -- May 13, 97 (455)
  What should I do?
- **/dev/term** -- *Solaris 2.5.1 / IPX* -- May 6, 97 (214)
  I have been having some problems getting a printer to print from the serial port on one of my SUN's (*IPX - Solaris 2.5.1*). I determined that the problem was not with the printer but with the serial ports so I did as the man from SUN suggested which was delete "a" and "b"

Figure 2: Example node page of Sun QA-Pack. This is a list of threads' digests.

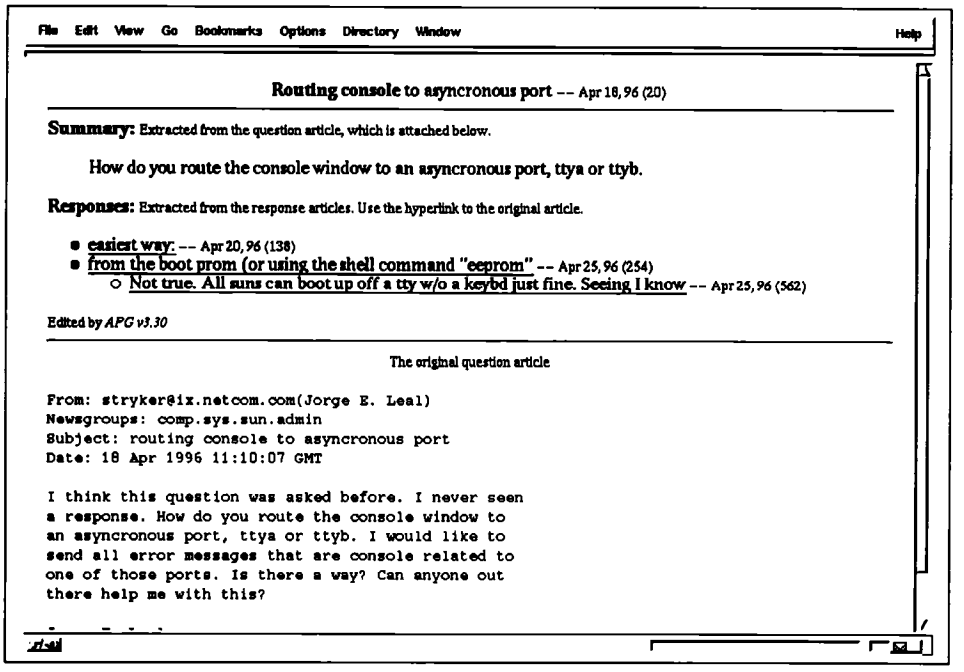

Figure 3: Example thread page of Sun QA-Pack. This page comprises a headline, a summary, leads, and the original question article.

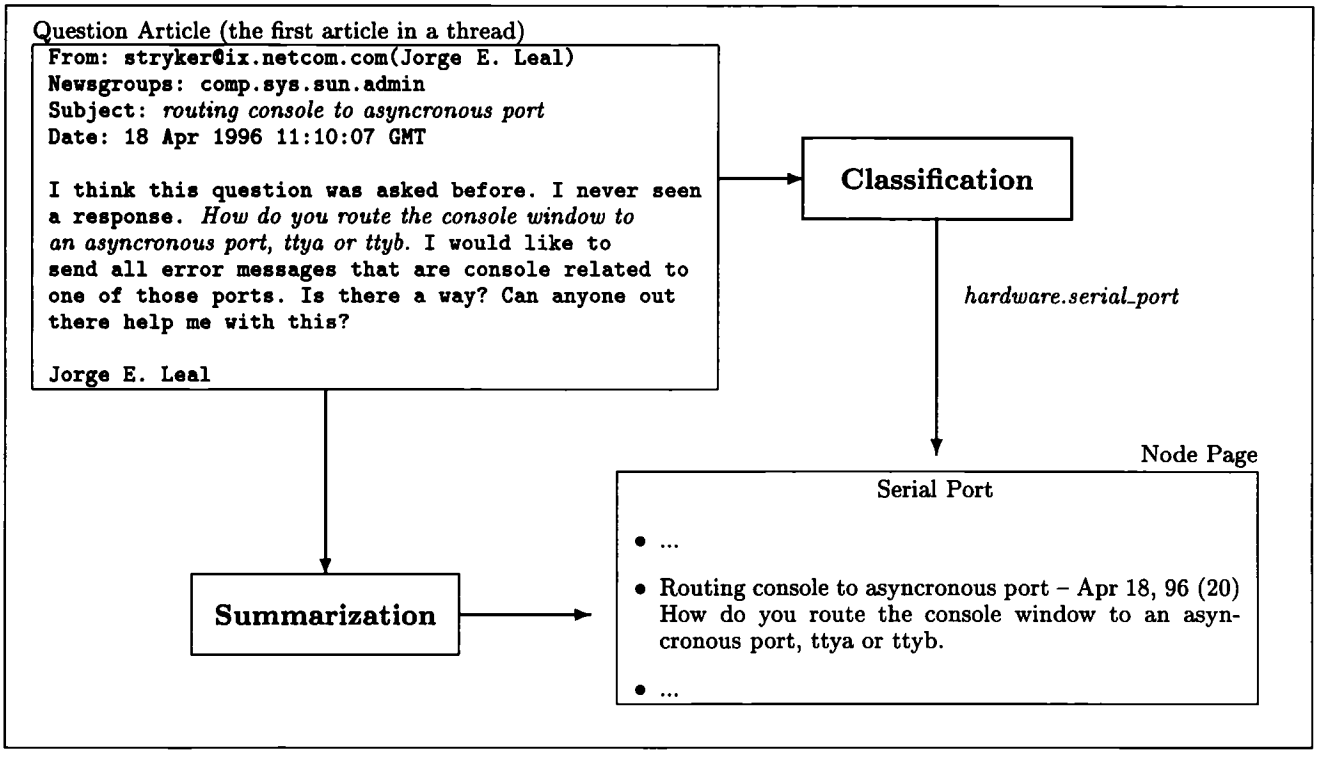

Figure 4: Text classification and text summarization. Text summarization makes a two-layer summary and text classification determines the page that the summary should be in.

questions or problems to theirs. We derived, from the node page's design, two requirements for good summaries as follows:

1. Content:
   A summary should contain the question or the problem that is discussed in the thread.
2. Length:
   A summary should be as short as possible.

To fulfill the above requirements, we employ the two-layer summary—a group of a headline and a summary. A headline, which denotes the thread's topic, is made from the subject line in the question article's header. In many cases, however, a headline is too short, so it does not describe what the question or the problem is sufficiently. Therefore, the summary that describes a little more details about the question or the problem is necessary to complement the headline.

Our system makes a summary from a question article's body by the summary-extraction method that was developed for extracting the most important sentence from a Japanese USENET article in the newsgroup fj.wanted[3](Sato & Sato 1996). This method is based on two basic assumptions.

1. Types of information to be extracted as summary are limited.
2. A particular type of information is presented with the particular syntactic signs or cue phrases (Edmundson 1969; Paice & Jones 1993).

The first assumption says that we may concentrate on a small set of information types in the summary-extraction process. In QA-Pack generation, the information types that we concentrate on are *a question about something*, *a description of a problem*, *a goal of a questioner*, etc. The second assumption says that we can detect the information types by the particular signs; an example follows:

(1) I *want to* connect an ISDN line so that I may have a permanent connection from my PC at home to the server at the office.

In this example, the sign *want to* indicates that the sentence may represent the questioner's goal. We formalize these signs as features, which are defined as regular expressions—for example, the regular expression of the *want* feature is as follows:

(*want to*|*wish to*|*would like to*|*I want*)

[3]The newsgroup fj.wanted consists of articles for selling, buying, and requiring information.

Table 1: Features for English summary extraction.

Type	s/w	Feature Name	Expression
Goal	G	want	*want to*
	G	need	*I need*
Situation	S	I'm doing	*I'm attempting to*
	S	try	*when I try*
At a Loss	K	at a loss	*at a loss*
Negation	N	got a message	*got – message*
	N	negative verb	*crash*
	N	not+verb	*cannot work*
	N	fine until	*fine until*
	N	only able to	*only able to*
	n	not	*not*
	n	however	*however*
Problem	P	problem	*my problem*
	P	verb+problem	*having problems*
Question	Q	question	*my question is*
	Q	Can I	*Can I*
	Q	Do you	*Do you*
	Q	Is there any	*Is there any*
	Q	Is ?	*Is –?*
	Q	anybody	*does anybody*
	Q	5W1H	*What*
	Q	wondering	*I'm wondering*
	Q	teach	*please teach me*
	Q	advice	*Any advice on*
	q	interrogative	*–?*
	q	Can	*Can*

Table 1 shows example features that are used for summary extraction in English. In this table, the *expression* column shows example expressions. The *s/w* column indicates the power of the feature: the capital letters denote strong (reliable) features, and the lower case letters denote weak features.

The algorithm of summary extraction consists of two stages: the feature-detection stage and the sentence-extraction stage. The first stage scans the first ten sentences of a question article, and detects all features by string-pattern matching. The second stage determines the sentences to be extracted as a summary using detected features. Unlike the first stage that every sentence is processed independently, the second stage uses inter-sentence relations. The following is the procedure that extracts the summary sentences from an English question article:

1. Extract the first sentence that has one or more strong features as the summary sentence.
2. If the extracted sentence has neither Q feature nor q feature, and the next sentence has either Q feature or q feature, extract the next sentence and add to the summary.
3. If the extracted sentence has neither N feature nor n feature, and the next sentence has either N feature or n feature, extract the next sentence and add to

Table 2: Results of summary-extraction experiments for English articles.

	Known set		New set			
			Blind tests		After modification	
GOOD	133	75.1%	152	71.4%	173	81.2%
OK	8	4.5%	6	2.8%	13	6.1%
NG	29	16.4%	46	21.6%	23	10.8%
FAIL	7	4.0%	9	4.2%	4	1.9%
Total	177	100%	213	100%	213	100%

the summary.

The additional extraction steps (step 2 and 3) contribute to generation of more informative summaries. The following is the summary example that is generated by this procedure.

(2) I have problems connecting a printer to a Solaris 2.5 machine (Intel-based). I use /dev/lp1; however, the spooler (lpsched) seems to change the data; the printer prints incorrectly.

Obviously, the second sentence that was added by the step 3 (because the sentence contains *however*) makes the summary more precise and informative.

For Japanese articles, we use the different set of features that are defined by Japanese regular expressions. The extraction procedure is basically the same as it of English; however, it is slightly more complicated.

Table 2 shows an experimental results of summary extraction in English. We used two sets of articles: *Known set* consists of 177 articles, which we studied before system construction, and *New set* consists of 213 articles, which were collected from USENET after system construction. The sum of GOOD[4] and OK[5] is high in spite of low quality of the USENET article's texts[6]. Even though the system manages some of the typical spelling mistakes and grammatical errors, it is difficult to handle all these low-quality texts. The accuracy of Japanese summary extraction is almost the same as it of English.

Extracted Summaries Need Rewriting

Our current method is based on the following *opportunistic* assumption.

[4]The extracted sentences are the most appropriate as the article's summary.

[5]The extracted sentences are not the most appropriate as the summary, but denote the article's essence.

[6]One tester categorized the extracted summaries into four groups (GOOD, OK, NG, and FAIL).

There are distinctive sentences that describe the essence of the original document; we can detect and extract the sentences. These sentences are good enough as a summary without any modification.

As a result, we learned that the extracted summaries can be useful to some degree, but not high in quality; they can be improved into more appropriate summaries to the package's objectives. A solution is rewriting.

Why aren't extracted summaries high in quality? The keywords to answer this question are *document type* and *sentence style*. In our application, the original document is a USENET article, which a writer asks for advice for a question or a problem. In this type of documents, there are certain sentence styles that users find preferable—polite, indirect, detailed, and so on. For example, the following is a typical question sentence that is extracted as the summary sentence:

(3) Could anyone tell me where I can find the GNU gcc binaries?

The phrase *could anyone tell me* adds politeness and indirectness to the sentence.

On the other hand, in our application, the target document where summaries are presented is a node page, which users search for the similar questions and problems to theirs. Quick and easy scanning is the very important property that the node page should provide, so simple, short and direct sentences are preferable in the node page. For example, the following short and direct sentence is more preferable than (3).

(4) Where can I find the GNU gcc binaries?

From these examples, we surmise that preferable sentence styles differ according to the document types. To change the sentence styles, rewriting is necessary.

What Style Is Preferable and What Kinds of Rewriting Are Necessary?

Unfortunately there is a few studies about rewriting (or paraphrasing) in natural language processing (e.g., (Yamamoto, Masuyama, & Naito 1995; Robin & McKeown 1996; Dras 1997)), so intensive studies are needed for all aspects of rewriting.

We have to start from making a list of preferable properties for summary sentences; these properties are derived from consideration on the function of the target document, and the requirements for high-quality documents.

Simple, Short and Direct The function of a node page is to show the list of questions and problems related to certain specific topic. To place a larger number of summaries in a limited space and to assist quick

scanning, simpler, shorter and more direct summary is preferable.

There are various ways of rewriting that make a sentence simpler, shorter, and more direct. The followings are examples of desirable rewritings.

In the first example, rewriting changes directness of the sentence.

(5) *Can anyone tell me where to find* the gcc binaries for SunOS 4.1.3?
⇒ Where can I find the gcc binaries for SunOS 4.1.3?

This rewriting is implemented by the following rewriting rule.

(6) `Can anyone tell me where *verb`
`--> Where can I *verb`

In this rule, `*verb` indicates a variable that matches any verb.

The second example is slightly complicated than the first one.

(7) *Does anyone know of a reliable place from where I can obtain* a pkgadd-able copy of gcc 2.7.2 for Solaris 2.5?
⇒ Where can I obtain a pkgadd-able copy of gcc 2.7.2 for Solaris 2.5?

The third example needs rewriting twice.

(8) I would like to know how to create an icon image and bind it to my application *so that when I iconize my appl, the icon image is used instead of the default one.*
⇒ *I would like to know how to create* an icon image and bind it to my application. ___
⇒ How do I create an icon image and bind it to my application?

The first rewriting is to drop the part that starts from *so that*, and the second is to convert the sentence into the question. The second rewriting is implemented by a rewriting rule, like the previous two examples; i.e.,

(9) `I would like to know how to *verb`
`---> How do I *verb`

The first rewriting, however, needs the syntactical information; the part that starts from *so that* is an omittable subordinate clause: parsing is required to implement this type of rewritings.

The last example is difficult even if full parsing is available.

(10) I've just installed Solaris 2.5.1 x86, I can log in as root and the os seems to be functioning, but I can't create directories or new users (or install the sysadmin answerbook).
⇒ After I installed Solaris 2.5.1 x86, I can't create directories or new users.

Consistent and Standardized Consistent presentation is preferable in any kinds of technical documents: The similar matter should be presented in the same form. The easiest way to keep consistent presentation is the use of standard representations. Itemization, the presentation style of summaries in a node page, also requires uniforming of presentation in every item.

Standardization is necessary in various levels: word level, phrase level and sentence level. In our application, standardization of each technical word (hardware name, product name, software name, OS name, machine name, etc.) is very important. For example, the following rewriting is standardization of OS name, which is implemented in the current system[7].

(11) Can someone tell me how to configure sendmail.cf on *SunOS 5.5* to allow full email domains to pass thru.
⇒ Can someone tell me how to configure sendmail.cf on Solaris 2.5 to allow full email domains to pass thru.

Another example of word-level standardization uses the standard verb in the specific situation.

(12) Where can I *obtain* a pkgadd-able copy of gcc 2.7.2 for Solaris 2.5?
⇒ Where can I find a pkgadd-able copy of gcc 2.7.2 for Solaris 2.5?

Basically word-level standardization is implemented by using a thesaurus, which is a list of presentational variations.

Phrase-level standardization needs rewriting rules that allows variables. For example,

(13) Does anyone have *gcc/gzip running on a Solaris 2.5.1 machine*?
⇒ Does anyone have gcc/gzip for Solaris 2.5.1?

This rewriting can be implemented by the following phrasal rewriting rule.

(14) `*software running on a *OS machine`
`--> *software for *OS`

[7]The current system standardizes only OS names and machine names.

In this rules, both `*software` and `*OS` are variables.

Sentence-level standardization can be also implemented by rewriting rules. Let us assume that we use the following sentence skeleton to express the question that asks the place where we can obtain a software.

(15) Where can I find *software?

Under the assumption, we should rewrite the sentence as follows,

(16) *Is there an FTP site which has gcc* precompiled for us Solaris 2.x folks who don't have the cc compiler?
⇒ Where can I find gcc precompiled for us Solaris 2.x folks who don't have the cc compiler?

using the following rule.

(17) `Is there an FTP site whch has *software ---> Where can I find *software`

Example (13) is also rewritten as follows.

(18) *Does anyone have gcc/gzip for Solaris 2.5.1?*
⇒ Where can I find gcc/gzip for Solaris 2.5.1?

The rewriting of (18) can be viewed as paraphrasing.

Complement Headline In a node page, we employ two layered presentation—headline and summary—so that a summary can complement a headline. Ideally, a headline gives an overview and a summary gives a little more about the details.

(19) Ultra150 and internal array issue - Aug 14, 97 (317)
I am unable to configure the four 4.2gb hard drives with the format command or view them or the controller in soltice disksuite 4.0.

However, the current system has no method for coordinating a headline and a summary. Therefore, the system sometimes make redundant headline-summary group, such as follows.

(20) Add a new external hard disk to SPARC box under Solaris 2.5.1 - Jun 2, 97 (42)
I'm trying to add a 9 GB external hard disk to SPARC box under Solaris 2.5.1.

In the example (20), the only piece of information carried by the summary is that the size of the external hard disk is 9GB—it is not important.

We do not have the solid idea for solving this problem; however, we assume that a part of problem will be solved by dropping duplicated parts.

	general	domain-specific
thesaurus	1	2
words-to-words rewriting rules	3	4
tree-to-tree rewriting rules	5	6

Figure 5: Classification of rewriting. We categorized rewritings into six classes.

Our Implementation Plan

Currently we are studying what kinds of rewriting are necessary for our application in both of English and Japanese; and constructing the rewriting module that we attach to the system. From the study until now, we have learned that the rewriting module should provide three different methods of rewriting; (1) rewriting by using a thesaurus, (2) rewriting by words-to-words rewriting rules, and (3) rewriting by tree-to-tree rewriting rules.

Because the current system already provides the first method, i.e., rewriting by using a thesaurus, the remaining work is collecting the presentational variations for each technical term; however it is not easy. Automatic collection of variations is desirable.

The second method, rewriting with words-to-words rewriting rules, requires morphological analysis as a preprocessing. For Japanese, we decided to use JUMAN, a Japanese morphological analyzer developed by Kyoto University, and have implemented the rewriting system.

The third method, rewriting with tree-to-tree rewriting rules, requires parsing as a preprocessing. For Japanese, we decided to use KNP (Kurohashi & Nagao 1994), and we are now implementing a rewriting system like Grade (Nakamura, Tsujii, & Nagao 1985), which was used in the MU machine translation project.

The rewritings that are executed in each method can be divided into two classes: domain-specific rewritings and general rewritings. Thus, there are six rewriting classes (Figure 5). The domain-specific rewritings are exclusive to each application; the general rewritings are applicable to many applications.

References

Dras, M. 1997. Reluctant paraphrase: Textual restructuring under an optimization model. In *Proceed-*

ings of Pacling97, 48–55.

Edmundson, H. 1969. New method in automatic extracting. *Journal of the Association for Computing Machinery* 16(2):264–285.

Kurohashi, S., and Nagao, M. 1994. KN parser: Japanese dependency/case structure analyzer. In *Proceedings of The International Workshop on Sharable Natural Language Resources*, 48–55.

Nakamura, J.; Tsujii, J.; and Nagao, M. 1985. Grammar writing system (GRADE) of mu-machine translation project and its characteristics. *Journal of Information Processing* 8(2):93–100.

Paice, C. D., and Jones, P. A. 1993. The identification of important concepts in highly structured technical papers. In *Proceedings of Sixteenth Annual International ACM SIGIR conference on Research and Development in Information Retrieval*, 69–78.

Robin, J., and McKeown, K. 1996. Empirically designing and evaluating a new revision-based model for summary generation. *Artificial Intelligence* 85(1–2):135–179.

Sato, S., and Sato, M. 1996. Automatic digesting of network newsgroup fj.wanted. (in Japanese). *Natural Language Processing* 3(2):19–32.

Sato, M., and Sato, S. 1997. Natural language processing in the automated editing system for the USENET articles. In *Proceedings of Natural Language Processing Pacific Rim Symposium 1997*, 261–266.

Sato, M. 1997. *Automated Editing of Electronic Texts.* Ph.D. Dissertation, School of Information Science, Japan Advanced Institute of Science and Technology.

Yamamoto, K.; Masuyama, S.; and Naito, S. 1995. Green: An experimental system generating summary Japanese editorials by combining multiple discourse characteristics, (in Japanese). *Natural Language Processing* 2(1):39–52.