A Constraint Solving Web Service for Recognizing Historical Japanese KANA Texts

Kazuki Sando, Tetsuya Suzuki and Akira Aiba
Graduate School of Engineering and Science, Shibaura Institute of Technology, 307 Fukasaku, Minuma-ku, 337-8570, Saitama-shi, Saitama, Japan

Keywords: Natural Language Processing, Morphological Analysis, Constraint Solving, Web Service, Reprint.

Abstract: One of the first steps for researching Japanese classical literature is reading Japanese historical manuscripts. However the reading process is not easy, and time-consuming since a set of characters used in those manuscripts contain different characters from those currently used. There have been several attempts to read Japanese historical manuscripts. We proposed a framework to assist the human process for reading Japanese historical manuscript. It formulates the process as a constraint satisfaction problem, and a constraint solver in the whole system, was experimentally implemented as a UNIX command. In this paper, we added a Web service layer to the solver to realize loose coupling between the solver and the other subsystems. Thanks to the loose coupling, any programming language can be used for implementation of other parts of the whole system. In addition, the constraint solving Web service can be public through the Internet. We experimentally confirmed the solver as a Web service is faster than the that as a UNIX command if both the solver and a client are connected to a same local area network.

1 INTRODUCTION

Before printing technologies became popular, all texts were hand-written in Japan, and copied by hand. Through a series of the copying processes, texts were often modified accidentally or intentionally. As a result, there were several versions of the same text. For example, Kokin waka shu which was the first anthology edited by an imperial order in 905 AD has more than 10 variants.

One of the first steps of researching Japanese classical literature is comparing those variants to determine the standard text. To do this, one has to read these manuscripts, but this is time-consuming and requires training since they are hand-written, and may contain characters different from those currently used. That is why we can not use an automatic character recognition system for current texts.

We proposed a framework for assisting the human process for reading Japanese historical manuscripts by employing constraint solving (Arai et al., 2013). A sequence of characters is constrained to form a valid word in a dictionary for historical Japanese. We experimentally implemented a backtrack-based constraint solver for reading Japanese historical manuscripts using one kind of Japanese characters called hiragana. The solver is a part of the whole system based on the framework. Because of the insufficient pruning, the backtrack-based constraint solver can not suppress combinatorial explosion.

We then proposed a minimum-cost-method-based constraint solver as a successor of the backtrack-based solver (Watanabe et al., 2015). To suppress combinatorial explosion, the solver uses the A^* algorithm.

In this paper, we propose a constraint solving Web service. We added a Web service layer to the minimum-cost-method-based constraint solver to realize loose coupling between the constraint solver and the other subsystems. Thanks to the loose coupling, suitable programming languages can be used to implement other parts of the whole system. In addition, the constraint solving Web service can be available to other researchers concerning reprinting of Japanese historical text through the Internet.

In this paper, we use the word “kana” as well as “hiragana” for the sake of convenience though hiragana is a part of kana in general.

In section 2, we introduce hiragana, and we summarize existing research in section 3. We redefine the constraint satisfaction problem (CSP) for reading Japanese historical manuscripts in section 4, and ex-

Sando, K., Suzuki, T. and Aiba, A.
A Constraint Solving Web Service for Recognizing Historical Japanese KANA Texts.
DOI: 10.5220/0006709702570265
In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART 2018) - Volume 2*, pages 257-265
ISBN: 978-989-758-275-2

plain the constraint solving process of the minimum-cost-method-based constraint solver briefly in section 5. In section 6, we describe a constraint solving Web service. Experimental results are summarized in section 7. Section 8 states concluding remarks.

2 READING HISTORICAL JAPANESE CHARACTERS

2.1 Japanese Classical Manuscripts and Hiragana

After introducing Chinese characters to Japan in the 4th, or 5th century, a method to represent Japanese sentences using kanji was invented by the 8th century. This is called Man'yo-gana. They were unmodified Chinese characters and used to represent Japanese syllables according to their pronunciation. In the 8th century, it is said that there were 88 syllables in Japanese language different from about 50 syllables at present, and over 900 Chinese characters were used to represent them. During the 8th and 9th century, a new kind of character called hiragana was invented based on a cursive form of hand-written Chinese characters. Hiragana was gradually accepted by the end of the 9th century (Frellesvig, 2010).

2.2 Reading Hiragana in Japanese Historical Texts

Hiragana used in historical texts, which we will call historical hiraganas hereafter, is quite different from that currently used:

1. A set of historical hiragana contains different characters from those currently used. There are several characters to represent one syllable.
2. A special symbol called odori-ji is used to represent a repetition of an immediately previous character.
3. Especially in hand-written texts, each occurrence of the identical hiragana may have a different shape, according to each author or text.

Because of such characteristics, reading Japanese historical text is difficult even for Japanese. The Fig.1 shows a fragment of historical Japanese text written in hiragana taken from Tale of Ise, originally written in 1234 AD, and copied in 1547 AD (Reizei, 1994). In this fragment, the 1st, 3rd, 5th and 6th characters are easy to recognize for us since they are quite similar to that hiragana currently used.

Figure 1: A fragment of historical Japanese text which reads (mu)-(ka)-(si)-(o)-(to)-(ko).

Actually, they are (mu), (si), (to), and (ko), respectively. However, the 2nd, and 4th hiraganas are difficult to read since they are not currently used.

In the following, we will briefly explain how a human tries to read them. When we focused on the 2nd character, we may suspect that this would be (tu), or (ka) by its shape. We can determine that the 2nd character is (ka) since (mu-ka-si) is a valid Japanese word meaning 'the past'. By applying the similar inference, we can determine that the 4th hiragana is (o) since (o-to-ko) is a valid word meaning 'a male'. By observing this small example, it is clear that the knowledge on shapes of historical hiragana is insufficient, but the knowledge on historical Japanese words is necessary for recognizing Japanese historical manuscripts.

3 EXISTING RESEARCH

Yamamoto and Osawa proposed an optical character recognition (OCR) method for kana and kanji characters in cursive style (Yamamoto and Osawa, 2016). It is based on elastic pattern matching for character recognition (Terasawa and Kawashima, 2011). According to the paper, the accuracy of pattern recognition is more than 80%.

Hayasaka et al. proposed a recognition method based on a convolutional neural network (Hayasaka et al., 2017b). Their method is demonstrated on a website (Hayasaka et al., 2017a).

Yamada et al. proposed a system for reprinting characters in historical manuscripts by a combination of OCR and character n-gram (Yamada and Shibayama, 2003). Given unreadable characters, OCR outputs the results. These results are associated with prepared n-gram information to output candidates of recognition. Even though n-gram is used, n is 2 or 3.

We proposed a framework for assisting the human process for reading Japanese historical manuscripts by employing constraint solving (Arai et al., 2013). A sequence of characters is constrained to form a valid word in a dictionary for historical Japanese. The overall structure of the framework is shown in Fig.2. It consists of two major subsystems: a character recognizer and a constraint solver. The character recognizer segments an input image into characters, determines hiragana candidates for the segmented characters, and constructs a CSP based on the determined candidates. Then the constraint solver solves the CSP by assigning possible reading using a word dictionary. Then the result of the constraint solving is returned to the character recognizer to revise recognition. This feedback is repeated if necessary. The system finally outputs the assigned reading.

Technical issues of the proposal are divided into two categories: one is image recognition, and the other is constraint solving. In the former issues, we have to think of extracting an image containing just one historical Hiragana from the original image of, for example, a page, and using the effective method to recognize an extracted historical Hiragana. In the latter issues, we have to think of the method of modeling of reprinting as a CSP, and the method of efficient constraint solving.

We focused on the constraint solving issue and experimentally implemented a backtrack-based constraint solver for Reprint-CSPs. The solver is for reading Japanese historical manuscripts using hiragana, and finds maximally better solutions according to a solution comparator called locally-predicate-better (Borning et al., 1992). The solving process is a kind of morphological analysis. Because of an insufficient branch-and-bound pruning, it can not suppress combinatorial explosion.

We then implemented a minimum-cost-method-based solver as a successor of the backtrack-based solver (Watanabe et al., 2015). Given a CSP, a dictionary and an positive integer n, it extracts better admissible solutions according to the locally-predicate-better comparator from n-best admissible solutions according to solution cost. The solver employs the A^* algorithm to avoid combinatorial explosion.

Both the backtrack-based solver and the minimum-cost-method-based solver are implemented in Ruby and are invoked as commands from character user interface.

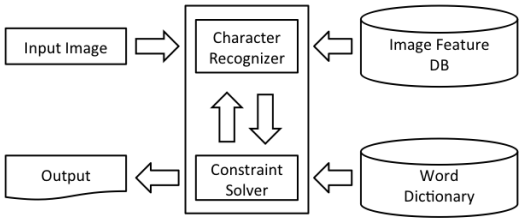

Figure 2: System Configuration.

4 REPRINT CONSTRAINT SATISFACTION PROBLEM

We redefine the CSP for reading historical kana text, which we call the Reprint-CSP, because the definition in (Watanabe et al., 2015) does not fit the CSP which the minimum-cost-method-based solver deals with.

A Reprint-CSP consists of the following seven components.

- a finite number of variables
- variables' domains
- a directed acyclic graph (DAG) over the variables
- a word dictionary
- explicitly-given constraints
- a word occurrence cost function
- a connectivity cost function

Each variable corresponds to a segmented character on a historical text image which contains one character. Because of complex shapes of historical kana characters, the character recognizer may not be able to determine if a segmented image corresponds to one or more characters. In such a case, the character recognizer enumerates possible cases. As a result, a segmented character corresponding to a variable may overlap segmented characters corresponding to different variables.

Fig.3 shows an image of a Japanese historical text consisting of five characters taken from Tale of Ise (Reizei, 1994), and a Reprint-CSP constructed from the text. The DAG of Fig.3 represents reading order among segmented characters. A variable x_1 is assigned to the first character. Similarly, variables x_2, x_5, x_6 and x_7 are assigned to the 2nd, the 3rd, the 4th, and the 5th character respectively. However, the second character can be recognized as a combination of two characters. For this reason, two variables x_3 and x_4 are assigned to the two characters.

The domain of each variable is a finite set of possible kana for the segmented character because it is difficult to determine a unique kana for a segmented character.

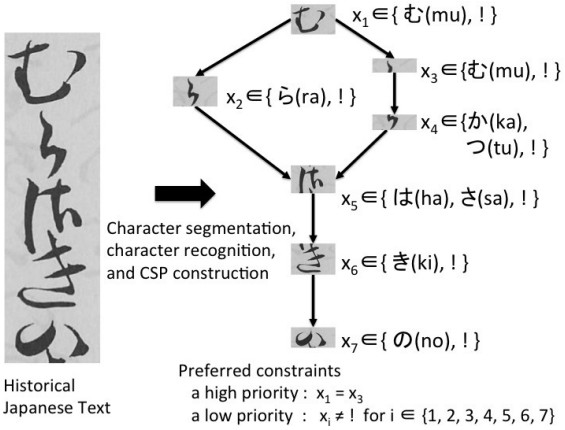

Figure 3: A constraint satisfaction problem.

For example, the domain of the variable x_4 includes two elements (ka) and (tu) in Fig.3 because the segmented character corresponding to x_4 can be read as (ka) or (tu). All variables' domains in Fig.3 include an element "!". We use the symbol as a special character meaning an unreadable character. Because the segmented character corresponding to x_3 can be read as an odori-ji, the domain of x_1, which is immediately prior to x_3 in the reading order, is added to the domain of x_3 so that the value of x_3 can be equal to that of x_1.

A constraint is a relation over the variables' domains. Constraints are labeled with priority levels for overconstrained situations. Constraints with the highest priority level are constraints which must be satisfied. They are called required constraints. Constraints with other priority levels are constraints which may not be satisfied. They are called preferred constraints.

In Fig.3, constraints $x_i \neq$ "!" for $i \in \{1,2,3,4,5,6,7\}$ are imposed not to assign the unreadable character to the variables. A constraint $x_1 = x_3$ is a constraint for the odori-ji.

In a Reprint-CSP, some constraints are implicitly given by a pair of the dictionary and the DAG over variables, and other constraints are explicitly given. Each maximal path over the DAG represents reading order of variables. The dictionary constrains local character sequences so that each of them forms a word in the dictionary. In addition, the DAG constrains combinations of the locally constrained character sequences so that each of maximal character sequences forms a sequence of words in the dictionary. These implicit constraints are required constraints. Explicitly-given constraints are required or preferred constraints. They are, for example, used to declare constraints of the odori-ji.

A solution for a Reprint-CSP is a partial function from variables to their values. An admissible solution for a Reprint-CSP is a function from variables on a maximal path of the DAG to their values where a sequence of variables' values according to the maximal path must be a sequence of words in the dictionary due to the implicit required constraints.

For example, in Fig.3, there are two maximal paths p_1 and p_2 as follows.

$$\begin{aligned} p_1 &\equiv (x_1, x_2, x_5, x_6, x_7) \\ p_2 &\equiv (x_1, x_3, x_4, x_5, x_6, x_7) \end{aligned}$$

The following function θ is an admissible solution on the maximal path p_1.

$$\begin{aligned} \theta \equiv \{&x_1 \rightarrow (mu), x_2 \rightarrow (ra), x_5 \rightarrow (sa), \\ &x_6 \rightarrow (ki), x_7 \rightarrow (no)\} \end{aligned}$$

It is because the solution θ can be obtained by word sequences (a noun (mu-ra-sa-ki), a particle (no)), (a noun (mu-ra), a noun (sa-ki), a particle (no)) and so on.

If a solution of a Reprint-CSP does not give any value to a variable, constraints relevant to the variable are regarded as constraints satisfied by the solution. For example, because the solution θ does not give any value to x_3, the constraint $x_1 = x_3$ is satisfied by θ.

Better admissible solutions can be selected from two different viewpoints: 1) solution cost and 2) priority levels of constraints.

1. An admissible solution θ is better than an admissible solution ρ from a viewpoint of cost if the cost of θ is smaller than that of ρ. The cost of an admissible solution is the minimum cost of word sequences which give the solution. The cost of a word sequence is a cumulative cost of word occurrence costs and connectivity costs in the word sequence. A word occurrence cost and a connectivity cost between two words are given by two cost functions of the Reprint-CSP.
2. Better admissible solutions, which satisfy preferred constraints as well as possible, can be selected using solution comparators which determine partial orders over admissible solutions. For example, locally-predicate-better is one of such comparators (Borning et al., 1992).

5 A MINIMUM-COST-METHOD-BASED SOLVER

We explain the constraint solving process of the minimum-cost-method-based solver mentioned in the section 3 briefly. It works as follows.

1. The solver constructs a reading assignment graph with costs from a given Reprint-CSP. It is a directed acyclic graph with a most upstream node

Figure 4: A fragment of historical Japanese text which reads (si)-(no)-(hu)-(su)-(ri)-(no)-(ka)-(ri)-(ki)-(nu)-(wo).

and a most downstream node. Each node is labeled with a sequence of variables, an assigned reading (a word in a dictionary of the Reprint-CSP), a part of speech, and two costs. The two costs are a occurrence cost of the word and the minimum cost from the most upstream node to the node. Each directed edge represents a reading order between two nodes, and is labeled with a connectivity cost.

2. The solver enumerates *n*-best solutions over the reading assignment graph with costs from a view point of costs, and extracts locally-predicate-better solutions from the *n*-best solutions.

In the following, an example of constraint solving process is shown using Fig.4, Fig.5, Fig.6, and Fig.7.

Fig.4 shows a fragment of historical Japanese text (Reizei, 1994) which reads (si)-(no)-(hu)-(su)-(ri)-(no)-(ka)-(ri)-(ki)-(nu)-(wo).

Fig.5 shows a directed acyclic graph over variables of a Reprint-CSP for the text of Fig.4. Each node is labeled with a variable in the upper part and its domain in the lower part. Each directed edge between two nodes represents a reading order between the two variables.

Fig.6 shows a complete reading assignment graph with costs constructed from the Reprint-CSP. The most upstream node and the most downstream node represents the beginning and the end of a fragment of text respectively. The graph was constructed with

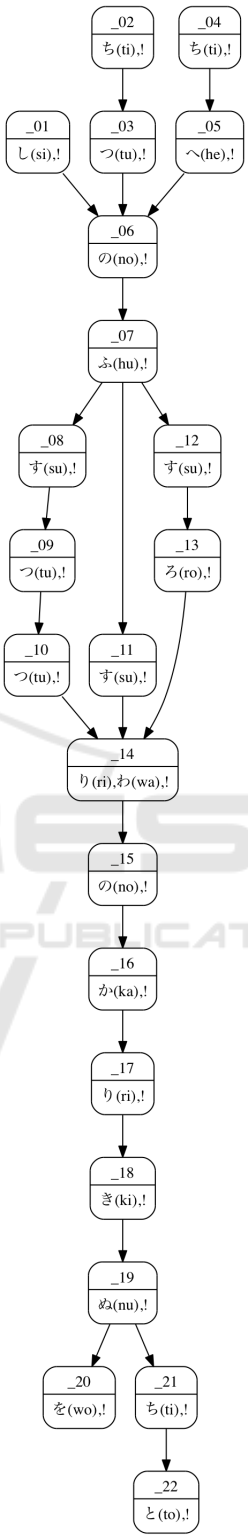

Figure 5: A directed acyclic graph over variables of a Reprint-CSP for the text in Fig.4.

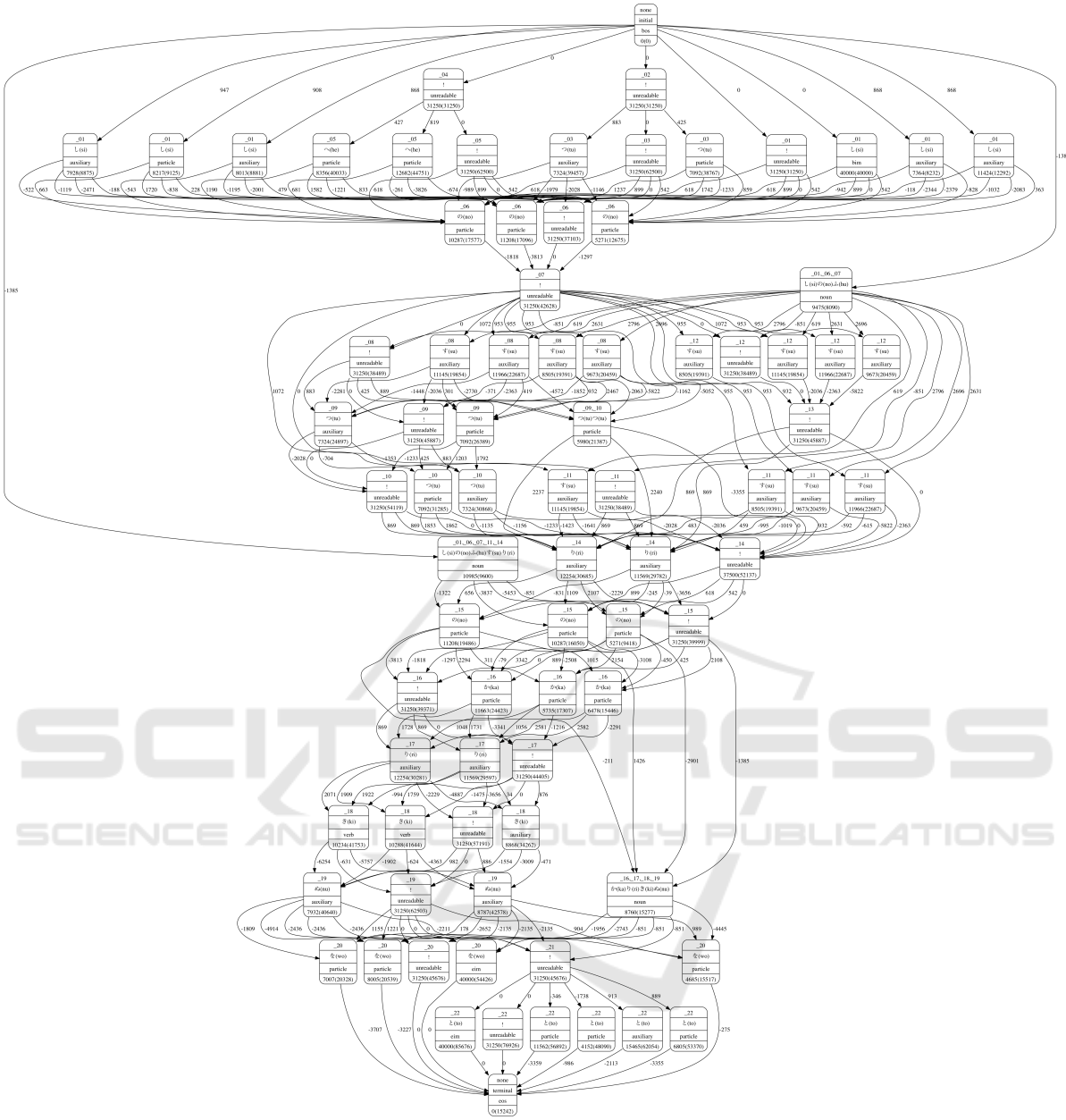

Figure 6: A complete reading assignment graph with costs.

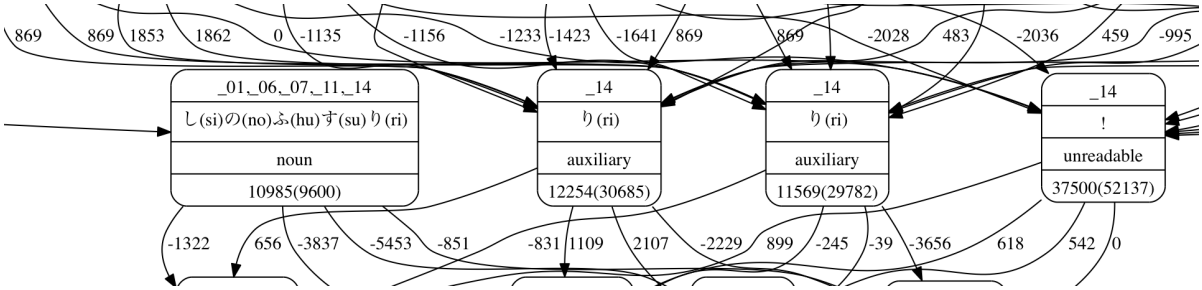

Figure 7: A part of a reading assignment graph with costs.

reference to a dictionary with 363 words, and has 81 nodes. If a dictionary with 421,896 words is used as in our experiments in section 7, the resulting graph has 558 nodes.

Fig.7 shows a part of the complete reading assignment graph. In Fig.7, a noun which reads (si)-(no)-(hu)-(su)-(ri) is assigned to a sequence of variables (_01, _06, _07, _11, _14). The occurrence cost of the node is 10,985, and the minimum cost from the most upstream node to the node is 9,600. The minimum cost to a node can be less than the occurrence cost of the node because connectivity costs on directed edges can be negative as shown in Fig.7.

6 A CONSTRAINT SOLVING WEB SERVICE

We added a Web service layer to the minimum-cost-method-based solver using a Web application framework Ruby on Rails. Thanks to the HTTP-based protocol, it realizes loose couplings between the constraint solver and parts of the whole system. As a result, any programming languages can be used to implement other subsystems in the whole system. In addition, the constraint solving service can be public through the Internet.

We designed an asynchronous Web API because constraint solving process is time-consuming as shown in experimental results in section 7. For example, it takes a few seconds for the minimum-cost-method-based solver to solve a Reprint-CSP with about 200 variables. If a Reprint-CSP and its solutions are exchanged between a server and a client in one round trip over HTTP, it causes blocking. It means the client has to wait for the solution in a few seconds.

Fig.8 shows a protocol sequence diagram of the Web API. To avoid blocking, the Web API protocol involves two round trips between a client and a server. The following is a brief explanation of the protocol.

1. A client sends a Reprint-CSP, a dictionary name, and a positive integer n in a JSON format (T. Bray, 2014) using the POST method of HTTP to a Web server as follows.

   ```
   POST /solver/csps
   ```

2. The server returns a URL with a randomly generated ID to the client, and starts constraint solving.
3. The client requests solutions for the Reprint-CSP to the Web service using the GET method of HTTP with the returned URL as follows.

   ```
   GET /solver/solutions/ID
   ```

   Because the ID in the URL is randomly generated, it is difficult for a client to steal a look at solutions for other users.
4. If solutions for the URL are available at that time, the Web service provides a set of pairs of a solution and unsatisfied constraints in the solution in a JSON format. If they are not available, the Web service provides an error message in a JSON format. Unsatisfied constraints sent from the Web service will be hints to revise the original Reprint-CSP.

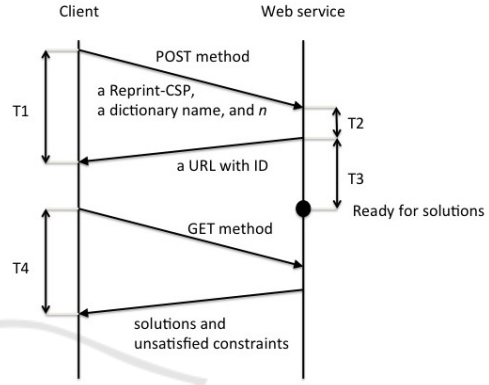

Figure 8: A protocol sequence diagram of the Web service.

7 EXPERIMENTAL RESULTS

In this section, we experimentally compare the minimum-cost-method-based solver implemented as a UNIX command and the solver implemented as a Web service. In addition, we evaluates the design of Web API based on the experimental results.

We built a Linux environment on a virtual machine on Mac OS X, and run our Web service on the environment on a Mac mini with 2.3GHz Intel Core i7 CPU and 16GB memory. We used a Mac mini with same specification in a same local area network for a Web client (a UNIX command `curl`) and the solver implemented as a UNIX command.

The two solvers use a same dictionary called UniDic for Early Middle Japanese (Ogiso et al., 2012), which includes 421,896 words. Every time we invoke the solver as a UNIX command, the command reads the dictionary, stores it as a trie tree and use the trie tree for a constraint solving. After finishing the constraint solving, the trie tree is lost. On the other hand, when we start the solver as a Web service, the Web service reads the dictionary, stores it as a trie tree, and keep the trie tree for constraint solvings as long as the Web service is running.

Table 1: Reprint-CSPs for experiments.

	# of variables	Average size of variables' domains	# of constraints
Reprint-CSP 1	211	1.22	220
Reprint-CSP 2	214	2.00	215
Reprint-CSP 3	303	1.23	318

Table 2: Average execution time.

	UNIX command (sec)	Web service				
		T1 (sec)	T2 (sec)	T3 (sec)	T4 (sec)	T1+T3+T4 (sec)
Reprint-CSP 1	5.538	1.258	0.044	2.576	0.002	3.836
Reprint-CSP 2	12.358	1.286	0.041	8.961	0.002	10.249
Reprint-CSP 3	7.146	1.260	0.047	3.913	0.002	5.176

We used three Reprint-CSPs shown in Table.1. Reprint-CSP 1 and Reprint-CSP 2 are CSPs for a page in Tale of Ise, and Reprint-CSP 3 is a CSP for two pages in Tale of Ise.

We gave each of the three Reprint-CPSs to both the solver as a UNIX command and the solver as a Web service five times, and measured their average execution time. For the solver as a Web service, we measured four durations T1, T2, T3, and T4 in Fig.8.

- T1 is a waiting time of a client between sending a Reprint-CSP and receiving a URL.
- T2 is an execution time in a Web service between receiving a Reprint-CSP and sending a URL.
- T3 is an execution time for constraint solving in a Web service.
- T4 is a waiting time of a client between requesting solutions and receiving the solutions.

Because T1 and T3 may overlap, a waiting time of a client between sending a Reprint-CSP and receiving solutions is at most T1+T3+T4.

Table.2 shows the results. In the three cases, the solver as a Web service is faster than the solver as a UNIX command by about 2 seconds. The execution time of a Web service (T1+T3+T4) ranges between 69% and 84% of that of a UNIX command. The reason is that the solver as a Web service is ready to solve when it receives a Reprint-CSP while the solver as a UNIX command has to read the dictionary before solving when it receives a Reprint-CSP.

8 CONCLUSION

We added a Web service layer to an existing constraint solver for reprinting Japanese historical text. Thanks to the Web service layer, we can use suitable programming languages to implement other parts of the whole reprint support system and will be able to publish our constraint solver on a Web site as a Web service so that other researchers concerning reprinting of Japanese historical text can use it. We conducted experiments and confirmed the solver as a Web service is faster than the original solver as a UNIX command by about two seconds if we use both a client and the Web server in a same local area network. It is because the solver as a Web service can reuse a setup done at startup of the Web service while the solver as a UNIX command has to do setup every time it is invoked.

One of our future works is to publish our Web service on a Web site. In addition, implementing other subsystems in the whole reprint support system and combining them into one system are also our future works.

ACKNOWLEDGEMENTS

This work was supported by JSPS KAKENHI Grant Number JP16K00463.

REFERENCES

Arai, Y., Suzuki, T., and Aiba, A. (2013). *Recognizing Historical KANA Texts Using Constraints*, pages 151–164. Springer Japan, Tokyo.

Borning, A., Feldman-Benson, B., and Wilson, M. (1992). Constraint hierarchies. In *Lisp and Symbolic Computation*, pages 48–60.

Frellesvig, B. (2010). *A History of the Japanese Language*. Cambridge University Press.

Hayasaka, T., Ohno, W., Kato, Y., and Yamamoto, K. (2017a). Recognition of kuzushiji (hentaigana and cursive script) by deep learning (ver.0.4.1).

Hayasaka, T., Ohno, W., Kato, Y., and Yamamoto, K. (2017b). Trial production of application software for machine transcription of hentaigana by deep learning. In *Proceedings of the 31st Annual Conference of the Japanese Society for Artificial Intelligence*.

Ogiso, T., Komachi, M., Den, Y., and Matsumoto, Y. (2012). Unidic for early middle japanese: a dictionary for morphological analysis of classical japanese. In Chair), N. C. C., Choukri, K., Declerck, T., Doan, M. U., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

Reizei, T. (1994). *Tales of Ise (photocopy)*. Kasama Shoin.

T. Bray, E. (2014). The javascript object notation (JSON) data interchange format. RFC 7159.

Terasawa, K. and Kawashima, T. (2011). Word spotting online. In *Proceedings of the Computers and the Humanities Symposium*, volume 2011, pages 329–334.

Watanabe, S., Suzuki, T., and Aiba, A. (2015). Reducing of the number of solutions using adjacency relation of words in recognizing historical kana texts. *IPSJ Journal*, 56(3):951–959.

Yamada, S. and Shibayama, M. (2003). An estimation method of unreadable historical character for manuscripts in fixed forms using n - gram and ocr. *IPSJ SIG Notes*, 2003(59):17–24.

Yamamoto, S. and Osawa, T. (2016). Labor saving for reprinting japanese rare classical books. *Journal of Information Processing and Management*, 58(11):819–827.