An Extension of Blank Element Selection Algorithm for Element Fill-in-Blank Problem in Web-Client Programming Self-Study System

Huiyu Qi [1], Nobuo Funabiki [1,*], Khaing Hsu Wai [1], Mustika Mentari [1], and Wen-Chung Kao [2]

[1] Department of Information and Communication Systems Okayama University, Japan
[2] Department of Electrical Engineering, National Taiwan Normal University, Taipei, Taiwan
Email: pbr17iqa@s.okayama-u.ac.jp (H.Q.); funabiki@okayama-u.ac.jp (N.F.);
khainghsuwai@s.okayama-u.ac.jp (K.H.W.); pqt85hm5@s.okayama-u.ac.jp (M.M.); jungkao@ntnu.edu.tw (W.-C.K.)
*Corresponding author

***Abstract*—Nowadays, web application systems have placed contemporary significance in our societies through interactive human-computer interfaces made by web-client programming using HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript. Previously, we have studied the Element Fill-in-blank Problem (EFP) for self-study of web-client programming by novice students, and presented the blank element selection algorithm with selection rules to generate new EFP instances from proper source codes. However, several important rules were missing in the previous version, such as those for handling complex CSS properties and HTML attributes. Additionally, the previous version lacked rules for dealing with advanced JavaScript objects, like the navigator object used for geolocation. In this paper, we present an extension of the algorithm by adding new selection rules for a more comprehensive study of web-client programming. We increased new selection rules by analyzing 10 source codes. For evaluations, we first compared the number of generated blanks between the previous algorithm and the proposed one, showing a significant increase. Then, we generated 10 new EFP instances with the proposed rules and confirmed that through applications to students, they are more difficult and comprehensive than the previous ones, leading to a better understanding of a wider range of concepts.**

***Keywords*—web-client programming, blank element selection algorithm, element fill-in-blank, Python, BeautifulSoup, regular expression**

I. Introduction

Nowadays, computers play essential parts of new technology productions in evolving the world. This means that programming is exceptionally important in our societies. Basically, computer systems can be created using programming. Therefore, it becomes crucial to learn common programming languages to develop efficient and useful application systems, particularly, for students in Information Technology (IT) or Computer Science (CS) departments of universities or novice engineers in IT related divisions of companies. Learning popular programming languages can give students competitive edges in this technology-driven world.

Nowadays, web application systems have become most important computer application systems that occupy many parts of our daily lives and activities. On every day, we access to information and services in web application systems from smartphones or Personal Computers (PCs) through web pages.

In today, web pages are often made by combinations of Cascading Style Sheets (CSS), HyperText Markup Language (HTML), and JavaScript programs. HTML determines the structure of a web page where all the content parts are defined by HTML. CSS takes the responsibility for the organization and layout of the web page. JavaScript describes the dynamic effects of the web page.

Previously, we have studied the Element Fill-in-blank Problem (EFP) for self-study of web-client programming by novice students, and presented the blank element selection algorithm with selection rules to generate new EFP instances from proper source codes [1, 2]. In an EFP example, a source code with JavaScript, CSS, and HTML is given with several blank elements. Besides, the corresponding screenshot of the web page is given to determine the unique correct answers. In a web page, HTML and CSS use tags in the Document Object Model (DOM) to define static components, and JavaScript uses libraries to provide dynamic changes or actions to DOM. It is important to understand how to relate these three languages together in the source code. Thus, the related elements are usually blanked in the EFP code.

In the previous studies, after manually generating 46 EFP instances for basic grammar topics [1] and applicative topics [2], we analyzed the blanks and made seven blank element selection rules. They specifically target blanks related to the following: 1) tag elements, 2) CSS syntax, 3) JavaScript identifiers, 4) JavaScript reserved words, 5) JavaScript library classes and methods, 6) id names, and 7) textual content messages.

Manuscript received May 11, 2024; revised June 5, 2024; accepted July 1, 2024; published February 10, 2025.

The primary role of EFP is to facilitate novice students in reading, understanding, and practicing web-client programming code written in JavaScript, HTML, and CSS. The necessity of EFP arises from the need for students to incrementally build their code comprehension skills before moving on to writing code themselves. Initially, writing complete programs can be challenging, so starting with fill-in-the-blank exercises helps in understanding syntax and code structure. To enhance efficiency and reduce potential human errors, we presented the blank element selection algorithm using the rules. This algorithm is implemented using two open-source tools in Python, namely BeautifulSoup [3] and regular expressions [4].

It automates the generation of new EFP instances by selecting the blank elements from an input source code. To enhance efficiency and reduce potential human errors, we presented the blank element selection algorithm using the rules. This algorithm is implemented using two open-source tools in Python, namely BeautifulSoup [3] and regular expressions [4]. It automates the generation of new EFP instances by selecting the blank elements from an input source code [5]. The novelty of our approach lies in its systematic and rule-based selection process, ensuring that each blank is chosen according to specific standards. This systematic approach guarantees that each EFP instance has only one correct answer, as the selection rules are designed to produce unique blanks. However, the current rules were found to be insufficient in comprehensively covering various web-client programming studies. Therefore, to address this limitation, we propose new selection rules that include handling complex CSS properties, additional HTML attributes, and advanced JavaScript objects such as the navigator object for geolocation. These enhancements aim to provide a more comprehensive learning tool for students to better understand and practice a wider range of web programming concepts.

In this paper, we investigate new blank element selection rules to address the limitations identified in previous studies and extend the blank element selection algorithm. We add new rules for HTML tags such as the <kbd> tag, <nav> tag, and <section> tag, for JavaScript identifiers such as the navigator object, for new CSS properties like padding, font-family, and font-size. We also add new rules for HTML attributes such as class. This extension allows the generation of more diverse EFP instances that encompass a broader scope of web-client programming studies.

The rest of this paper is organized as follows: Section II discusses related works in literature. Section III reviews our previous works. Section IV presents the new blank element selection rules. Section V evaluates the extended algorithm. Section VI concludes this paper and outlines directions for future work.

II. Related Works

In this section, we discuss related works in the literature on web programming, regular expressions, and web scraping.

A. Web Programming

Anwyl *et al*. [6] introduced a tool designed for constructing online behavioral experiments. It explores the platform's features and functionalities aimed at facilitating the creation and execution of experiments in behavioral research.

Kar *et al.* [7] showed the gap between the skill requirements of software industries and web programming courses in universities, and proposed two different courses for frontend programming and backend one.

Turp *et al*. [8] explored an automated method for analyzing virtual reference interactions. It focuses on developing algorithms or systems that can automatically process and interpret interactions between librarians and patrons in virtual environments. The goal is to extract meaningful insights, patterns, or metrics from these interactions to improve service quality, efficiency, and user satisfaction in virtual reference services.

B. Web Scraping

Grade *et al*. [9] conducted a survey on Python libraries utilized for scraping social media content. It reviews various libraries and their capabilities for extracting data from platforms like Twitter, Facebook, and Instagram. The study assesses factors such as ease of use, Application Programming Interface (API) coverage, legality, and scalability, aiming to provide insights into the best tools for researchers and developers seeking to collect and analyze social media data efficiently and ethically.

Nigam *et al.* [10] explored web scraping comprehensively, covering tools, relevant legislation, and implementation using Python. It discusses popular scraping libraries and frameworks in Python, addresses legal considerations such as data privacy and copyright laws, and provides practical guidance on implementing scraping scripts. The study aims to equip readers with a holistic understanding of web scraping practices, ensuring ethical and compliant data extraction from websites for various applications.

Dalia *et al*. [11] introduced a novel web recommendation model utilizing web usage mining techniques. It focuses on analyzing user behavior data to personalize recommendations, enhancing user experience by suggesting relevant web content. The study aims to improve recommendation accuracy and effectiveness through advanced data mining methodologies applied to web usage patterns.

III. Review of Previous Works

In this section, we review our previous works on the Element Fill-in-blank Problem (EFP) for web-client programming.

A. EFP Instance for Web-Client Programming

For an EFP instance in web-client programming, students are provided with source code that includes HTML, CSS, and JavaScript with several blanks, as well

as a set of screenshots depicting the corresponding web page.

Fig. 1 illustrates the example answer interface. This interface operates within a web browser, facilitating both online and offline use, as the answer validation is handled by executing a JavaScript program within the browser. The correct answers are stored in an encrypted format using SHA256 to prevent cheating.

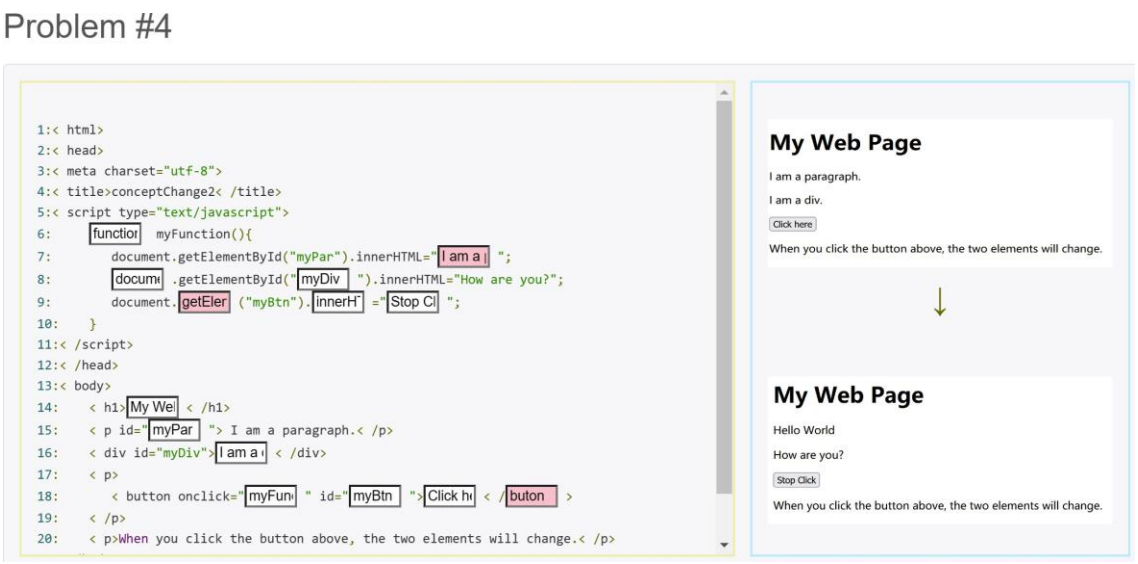

Fig. 1. ID = 4 Interface of EFP (basic version).

On the left side of the interface, the source code is displayed with forms for filling in the blanks. The right side presents screenshots of the initial web page and the web page after a button click. When the "Answer" button is pressed, the JavaScript validation function runs to verify the answers by matching them with the original elements in the source code. If a student's answer is incorrect, the background color of the form turns red. If the answer is correct, it turns white. Students can continue submitting answers until all answers are correct or they decide to stop.

B. Blank Element Candidates and Selection Rules

In an EFP instance, the following elements within the source code can be blanked:

- HTML: tags, attributes, id names, output messages,
- CSS: properties,
- JavaScript: reserved words, identifiers, id, library classes/methods, output messages.

The blank element selection algorithm is tasked with identifying the following seven components within the source code to be used as blanks:

- Text message: Elements that contain textual content within the code. This helps students practice working with text, including displaying messages, prompts, and user instructions.
- Tag element: HTML tags that structure the content. Students learn how to use these tags to organize web content, including headings, paragraphs, lists, etc.
- CSS syntax: The rules and syntax used in CSS for styling. This introduces students to CSS properties and their effects on the appearance of web pages, such as fonts, layout, and colors.
- Id name: Unique identifiers within the code. These teach students how to use id attributes to precisely target and manipulate HTML elements.
- JavaScript identifier: User-defined functions or variables. Understanding these is crucial for students as they form the building blocks of JavaScript programming.
- JavaScript reserved word: Predefined keywords in JavaScript grammar. These help students become familiar with the built-in functionalities and control structures of JavaScript, which are essential for writing effective scripts.
- JavaScript library class/method: Built-in or external JavaScript functions or methods. Exposure to these allows students to utilize powerful tools and solutions provided by various external libraries in their projects.

In the source code, JavaScript typically appears between <script> and </script> tags. This placement helps students understand how to properly embed and use JavaScript within their code. It is crucial to avoid blanking elements that are essential for understanding or cannot be inferred correctly. Blanking all elements associated with a specific id or identifier can make it extremely challenging to deduce the original content. Similarly, HTML tags typically come in pairs, and if both tags in a pair are blanked simultaneously, it becomes very difficult for students to answer them. Therefore, it is recommended to present HTML tags in pairs, blanking only one tag at a time. This approach allows them to understand how paired tags function together to define elements.

After thoroughly examining numerous source codes in web-client programming, we have determined that blanks should be drawn from the following components:

1) HTML tag elements, 2) CSS syntax, 3) JavaScript identifiers, 4) JavaScript reserved words, 5) JavaScript library classes and methods, 6) id attributes, and 7) textual content messages.

C. Blank Element Selection Algorithm

The blank element selection algorithm selects elements by following seven specific rules. It employs either regular expressions or BeautifulSoup, depending on the specific rule.

D. Methodology for EFP Instance Generation

The algorithm generates a new EFP instance for web-client programming through the following steps:

(1) Choose a web-client programming source code from a website or textbook that contains the relevant elements to be studied.
(2) Execute the source code in a web browser and capture the screenshots of the web page.
(3) Utilize the blank element selection algorithm to generate the input text file. This algorithm includes new rules for handling complex HTML attributes, ensuring that elements such as class and id are correctly identified and processed.
(4) Employ the answer interface generator with the input text file to produce the HTML, CSS, and JavaScript files for the answer interface in a web browser.
(5) Automatically substitute elements in the source code with the appropriate HTML entities, complete with their numbers and names, to ensure visibility in the browser.
(6) Integrate the captured screenshots into the HTML file to finalize the new EFP instance.

To enhance the educational value of the EFP instances, the algorithm now includes a broader range of keywords for blank elements. The selection process encompasses additional JavaScript objects, methods, and properties, as well as a more comprehensive set of CSS properties and selectors.

IV. Extension of Blank Element Selection

In this section, we present the extension of the *blank element selection algorithm* by adding the new rules.

A. New Blank Element Candidates

Here, we overview new blank elements to be selected in the new rules.

- HTML: tag, attribute, id, output message,
- CSS: selector, id, property, value,
- JavaScript: reserved word, identifier, id, value, library class/method, output message.

From these components, eight different elements should be selected for blanks in a source code, as illustrated in Fig. 2: 1) text message, 2) tag element, 3) CSS syntax, 4) id name, 5) JavaScript identifier, 6) JavaScript reserved word, 7) JavaScript library class/method, and 8) HTML attribute.

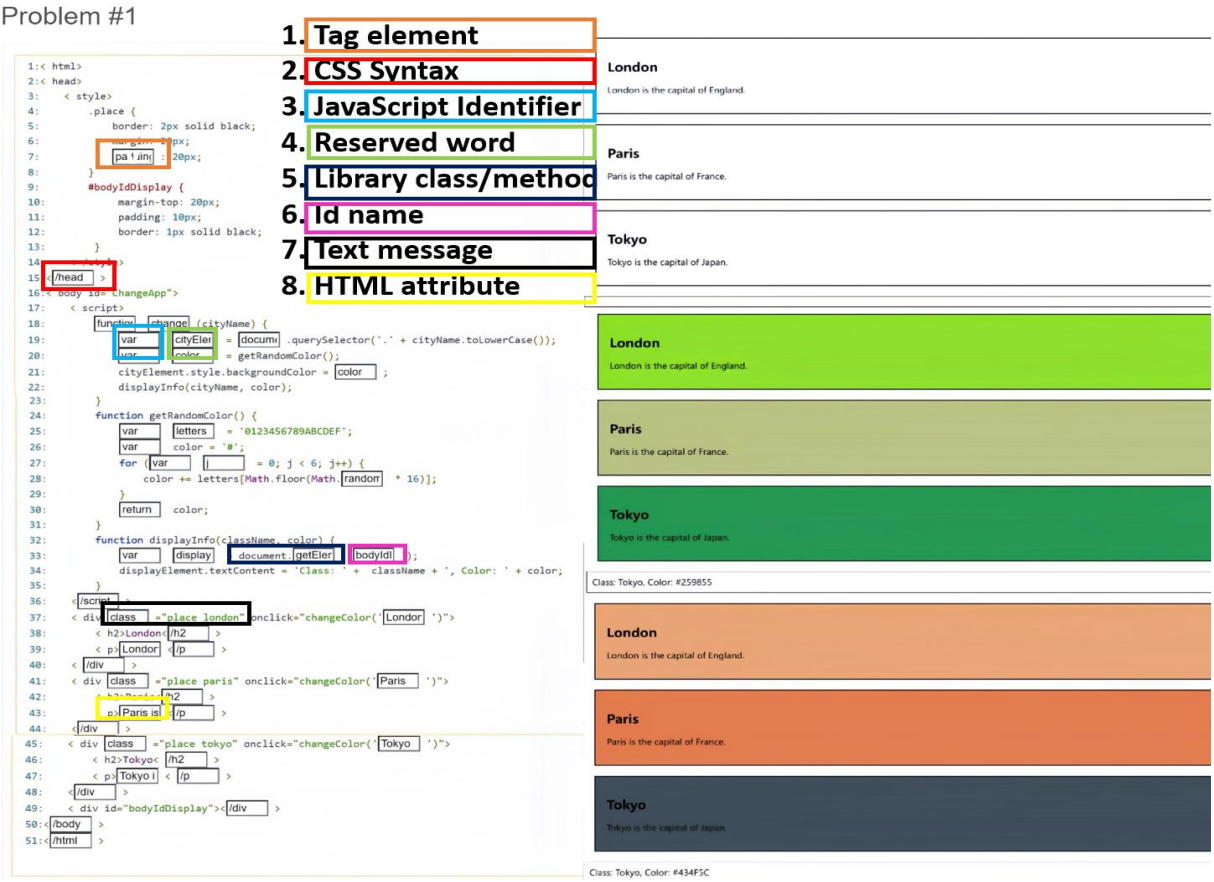

Fig. 2. Answer interface with new rules.

B. New Blank Element Selection Rules

Now, we discuss the importance, the extraction method, and the rules of selecting the blank element in the new rules.

(1) *Blank for Text Message*: A text message often appears in a web page to describe the information of the corresponding module. It is the first step of studying *web-client programming* to write codes to output text messages correctly. Since the screenshot of the web page on the right side of the interface shows any text message, it is possible to select any one for blank. Using BeautifulSoup for target data extractions [3] and re for regular expressions [4], a text message is extracted from the HTML source code.

(2) *Blank for Tag Element*: importance In HTML, a tag represents a control symbol for the following text data in the code. It is very important to understand the correct use of tags. Tags, such as <title> or <h1>, are used in pairs. At most one of each tag pair is selected for blank to avoid the confusion. Other tags, such as <input> or <img>, are used alone. This kind of tag can be selected for blank when the answer becomes clear from the following elements such as <input type="button" ...>.

(3) *Blank for CSS Syntax*: Cascading Style Sheet (CSS) is used to set the style in the web page that contains HTML elements. It can give the background color, font-size, font-family, color, and other properties in a web page. There are three types of CSS, namely, Inline CSS, Internal or Embedded CSS, and External CSS. In source codes for EFP instances, Internal CSS is used because it is intuitive and simple. The CSS syntax includes the selector and the declaration. The selector points to the corresponding HTML element style. The declaration includes a CSS property name and a value, separated by a colon. If the text size or the font-family is selected for blank, it is impossible to fill in it with the correct one. Instead, the color or the selector name is selected for blank, it is possible to answer them from the screenshot of the web page.

(4) *Blank for Id Name*: The id specifies an HTML element with the value. This value of the id must be unique within the HTML document. The id is used to point to a specific style declaration in the CSS. JavaScript can also use it to access and manipulate the element that has the specific id. Basically, the same id value appears several times in a source code, including the HTML, CSS, and JavaScript codes. At least one of them is not selected for blanks for each id value. Then, the correct id value can be filled in the blanks.

(5) *Blank for JavaScript Identifier*: In JavaScript, an identifier represents a sequence of characters in a code that identifies a variable, a function, or a property. An identifier differs from a string in that is data. An identifier is a part of the code. Like id, the same identifier value appears several times in a source code, including the HTML, CSS, and JavaScript codes. For example, with the button tag, a function is called by onclick = this function name(), to change attributes or information in the web page by clicking a button. At least one of them is not selected for blanks for each identifier value. Then, the correct identifier value can be filled in the blanks.

(6) *Blank for JavaScript Reserved Word*: In JavaScript, reserved words are defined in the grammar and need to be mastered well. Thus, they should be selected for blanks for studying JavaScript grammar. For example, a variable is defined using var, let, or const, and a function is defined using function, which are selected for blanks. Any reserved word in a source code can be selected for blank independently.

(7) *Blank for Library Class/Method*: In JavaScript, rich libraries are available for *web-client programming* to write a source code easily and simply to generate a web page that will offer various functions. Like the reserved word, any library class/method in a source code can be selected for blank independently. However, if less common methods are selected for blanks, additional descriptions to explain them should be added in the answer interface to assist solving them.

(8) *Blank for HTML Attribute*: In HTML, attributes provide additional information about elements and are essential for shaping the behaviors and appearance of a web page. Understanding and correctly using HTML attributes is fundamental to creating functional web content. HTML attributes can be associated with various elements, such as tags, and serve specific purposes based on the elements that they are applied to. The selection of blanks for HTML attributes depends on the context and the type of attribute.

V. Evaluation

A. Comparison with Previous Algorithm

In the first evaluation, we compare the number of generated blank elements between the previous algorithm and the proposal for the previous source codes in [12]. Table I shows the number of blank elements by the previous algorithm, and that by the proposal. It indicates that the number of blank elements is increased by 12.36% by the proposal, which will make EFP instances more difficult.

TABLE I. COMPARISON OF NUMBER OF BLANKS BETWEEN PREVIOUS ALGORITHM AND PROPOSAL

ID	topic	# of blanks by previous	# of blanks by proposal
1	object1	10	10
2	object2	12	12
3	changing content1	17	17
4	changing content2	21	28
5	alert() function	13	20
6	changing color	13	20
7	Date() function	20	20
8	prompt() function (subtraction)	8	8
9	prompt() function (add)	11	13
10	easy counter	21	21
11	click button	14	14
12	onmouse over and out	10	10
13	setTimeout() function	14	14
14	array	11	11
15	multiplication calculation	13	13
16	change background	17	19
17	try catch	19	19
18	try catch final (number)	22	26
19	try catch final (NaN)	21	26
20	custom timer	18	27
21	fixed timer	20	34
22	radio button (alert message)	18	19
23	radio button (show selected value)	23	26
24	radio button (text message)	18	32
25	checkbox (alert message)	18	27
26	checkbox (disable option)	19	22
27	checkbox (default option)	20	32
28	video	22	24
29	video (control function)	25	31
30	image function (modify image size)	17	17
31	image function (change image)	18	18
32	mouse pointer	11	11
33	select function (default option)	28	32
34	select function (disable option)	22	26
35	select function (output option)	26	33
36	select function (remove option)	24	24
37	image function (upload image)	22	22
38	first css web page	23	23
39	change page	23	24
40	change border image	21	24
41	table tr td	19	45
42	progress bar	20	29
43	canvas (simple rectangle)	14	14
44	canvas (gradient font)	14	14
45	canvas (gradient font)	23	23
46	canvas (repeat pattern)	19	22
47	camera	24	39
	total blanks (average)	857 (18.23)	963 (20.49)

B. *Application to New Topics*

In the second evaluation, we apply the extended algorithm to newly collected source codes that cover different topics from previous ones. Table II shows the topic, the number of code lines, the number of JavaScript lines, the number of blank elements by the proposal. Then, we assigned them to 20 students in Okayama University, Japan.

(1) *Solution Results of Individual Instances*: First, we analyze the solution results of the 10 EFP instances individually. Fig. 3 shows the average correct answer rate and the total number of answer submissions by the students for each instance. The average correct rate is 83.58% and the average submission number is 154.7 for one instance. The instances at ID = 2 resulted in the best rate of 89.71%, while the instances at ID = 10 resulted in the worst rate 78.01%. The topic of the instance at ID = 10 is navigator, which is the JavaScript object and is a more advanced one compared to simple HTML and CSS syntax. Students might not be familiar with how to use navigator.geolocation and its methods. We will investigate how to improve them.

(2) *Solution Results of Individual Students*: Next, we analyze the solution results of the 20 students individually. Fig. 4 displays each student's average correct answer rate and the total number of submissions across the 10 instances. The graph is sorted in descending order based on student performance. The average correct answer rate is 83.59%, and the average number of submissions is 77.35. Notably, the student with ID = 2 completed all 10 instances with only 22 submissions, indicating just 12 mistakes. In contrast, the student with ID = 4 made 298 submissions despite achieving a high correct answer rate of 96.95%. This suggests that the student demonstrated a strong determination to solve all instances.

TABLE II. EFP INSTANCES FOR NEW TOPICS

#ID	topic	# of code lines	# of JavaScript code lines	# of blanks
1	HTML class	51	20	43
2	HTML layout	68	7	34
3	iframe tag	46	9	28
4	computer code	46	12	27
5	form tag	55	7	30
6	input(submit)	90	23	53
7	input(color)	54	7	31
8	input(date)	54	9	31
9	input(email)	69	16	37
10	navigator object	52	14	29
total (average)		585 (58.5)	124 (12.4)	343 (34.3)

VI. CONCLUSION

This paper extends the blank element selection algorithm by incorporating new selection rules to enhance the comprehensive study of web-client programming.

For evaluations, the number of generated blanks was compared between the previous algorithm and the proposal. Then, 10 new EFP instances were generated by the proposal using source code with new topics, and were assigned to 20 students. The solution results suggest that they are more difficult than previous ones. In future works, we will explore and discover additional rules for the blank element selection to facilitate learning of web-client programming by students with enriching the functionality.

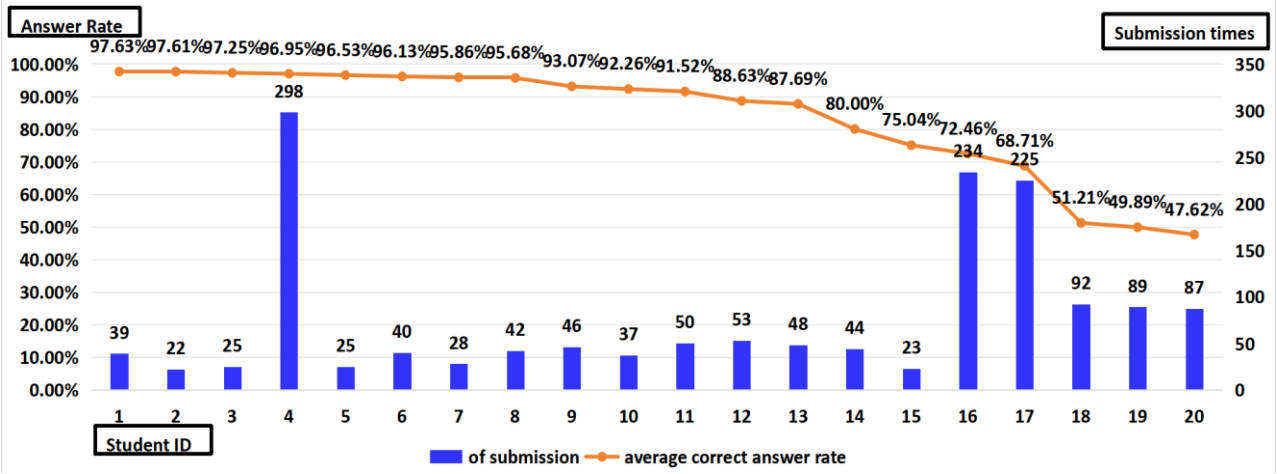

Fig. 3. Results of individual instance solutions.

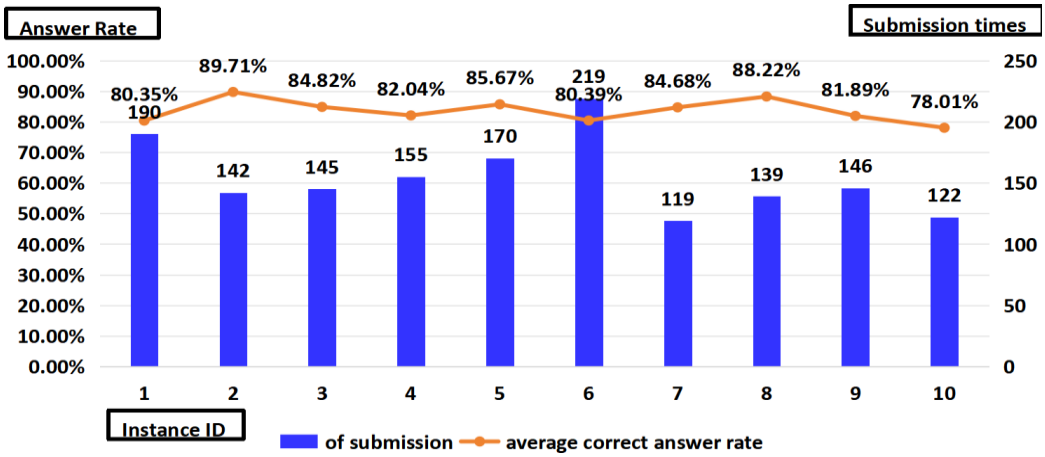

Fig. 4. Performance analysis of individual students.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Huiyu Qi conceptualized the research problem, developed the extended blank element selection algorithm, and implemented the evaluation experiments. Nobuo Funabiki provided guidance on algorithm design and supervised the research process. Khaing Hsu Wai and Mustika Mentari contributed to data analysis and results interpretation, as well as drafting the manuscript. Wen-Chung Kao reviewed the experimental methodology, contributed to the literature review, and provided critical feedback to improve the overall quality of the manuscript. All authors contributed to revising the manuscript and approved the final version for publication.

Acknowledgment

The authors wish to thank Prof. Nobuo Funabiki for his insightful guidance and support throughout the research process. They also express their gratitude to the students who actively participated in the evaluation experiments and provided valuable feedback, which contributed significantly to the study results.

References

[1] H. Qi, N. Funabiki, K. H. Wai, X. Lu, H. H. S. Kyaw, and W.-C. Kao, "An implementation of element fill-in-blank problems for code understanding study of JavaScript-based web-client programming," *International Journal of Information and Education Technology (IJIET)*, vol. 12, no. 11, pp. 1179–1184, October 2022.

[2] H. Qi, N. Funabiki, K. H. Wai, M. Z. Htun, K. T. Mon, and W.-C. Kao, "A study of element fill-in-blank problems for applicative grammar topics in JavaScript-based web-client programming," *IPSJ SIG Tech. Rep.*, vol. 2022-CE-166, no. 2, Oct. 2022.

[3] Python-BeautifulSoup. [Online]. Available: https://www.datacamp.com/tutorial/web-scraping-using-python.

[4] Python-Regular Expression. [Online]. Available: https://www.w3schools.com/python/python regex.asp.

[5] H. Qi, N. Funabiki, K. H. Wai, F. V. Hendryanna, K. T. Mon, M. Mentari, and W.-C. Kao, "A blank element selection algorithm for element fill-in-blank problems in client-side web programming," *Engineering Letters*, vol. 32, no. 3, pp. 684–700, 2024.

[6] A. L. Anwyl-Irvine, J. Massonni, A. Flitton, N. Kirkham, and J. K. Evershed, "Gorilla in our midst: An online behavioral experiment builder," *Behavior Research Methods*, vol. 52, pp. 388–407, 2020.

[7] S. Kar, M. M. Islam, and M. Rahaman, "State-of-the-art reformation of web programming course curriculum in digital

Bangladesh," *Int. J. Adv. Comp. Sci. Appl.*, vol. 11, no. 3, pp. 193–201, 2020.
[8] T. Clara and S. Hervieux, "Exploring an automated method for the analysis of virtual reference interactions," *Reference Services Review*, vol. 52, no. 2, pp. 247–256, 2024.
[9] S. Thivaharan, G. Srivatsun, and S. Sarathambekai, "A survey on python libraries used for social media content scraping," in *Proc. International Conference on Smart Electronics and Communication*, 2020, pp. 361–366.
[10] N. Harshit and P. Biswas, "Web scraping: From tools to related legislation and implementation using python," *Innovative Data Communication Technologies and Application*, vol. 59, pp. 353–370, 2021.
[11] D. L. Elsheweikh, "A Novel web recommendation model based on the web usage mining technique," *Journal of Advances in Information Technology*, vol. 14, no. 5, pp. 1019–1028, 2023.
[12] H. Qi, N. Funabiki, K. H. Wai, M. Z. Htun, V. Flasma, and Y. W. Syaifudin, "A study of blank element selection rules for element fill-in-blank problem in JavaScript-based web-client programming," in *Proc. International Conference on Engineering Management and Sustainable Innovative Technologies (ICEMSIT 2022)*, Nov. 2022, pp. 57–62.