applied system innovation

MDPI

Article

WOJR: A Recommendation System for Providing Similar Problems to Programming Assignments

Ryoya Yoshimura [1,*,†,‡], Kazunori Sakamoto [1,2,*,†,‡], Hironori Washizaki [1,‡] and Yoshiaki Fukazawa [1,‡]

[1] School of Fundamental Science and Engineering, Waseda University, 1-104 Totsukamachi, Shinjuku-ku, Tokyo 169-8050, Japan; washizaki@waseda.jp (H.W.); fukazawa@waseda.jp (Y.F.)
[2] WillBooster Inc., 1-36-2 Shinjuku, Shinjuku-ku, Tokyo 160-0022, Japan
* Correspondence: ryoya9071@toki.waseda.jp (R.Y.); exkazuu@gmail.com (K.S.)
† These authors contributed equally to this work.
‡ Submission is extension of conference paper: Yoshimura, R.; Sakamoto, K.; Washizaki, H.; Fukazawa, Y. Recommendation System Providing Similar Problems Instead of Model Answers to Programming Assignments. In Proceedings of the 5th IEEE Eurasian Conference on Educational Innovation 2022 (IEEE ECEI 2022), Taipei, Taiwan, 10–12 February 2022; pp. 1–4.

Abstract: Programming education for beginners often employs online judges. Although this helps improve coding skills, students may not obtain sufficient educational effects if the assignment is too difficult. Instead of presenting a model answer to an assignment, this paper proposes an approach to provide students with problems that have content and answer source code similar to the assignment. The effectiveness of our approach is evaluated via an intervention experiment in a university lecture course. The improvement in the number of correct answers is statistically significant compared to the same course offered in a different year without the proposed system. Therefore, the proposed approach should aid in the understanding of an assignment and enhance the educational effect.

Keywords: online judge; programming assignment; similar problem; recommendation system; programming education; model answer; algorithm learning

Citation: Yoshimura, R.; Sakamoto, K.; Washizaki, H.; Fukazawa, Y. WOJR: A Recommendation System for Providing Similar Problems to Programming Assignments. *Appl. Syst. Innov.* **2022**, *5*, 53. https://doi.org/10.3390/asi5030053

Academic Editors: Teen-Hang Meen and Chun-Yen Chang

Received: 26 February 2022
Accepted: 25 May 2022
Published: 31 May 2022

1. Introduction

In programming education, students learn to understand and implement algorithms. Online judges often support algorithm learning [1]. Online judges present the code specifications to be implemented to the students, receive students' code submissions, and determine whether the submitted code meets the specifications. Online judges can help students improve their ability to read problem sentences, write code in consideration of time and spatial complexities, and debug code.

Assignments and tasks must be selected appropriately to maximize the education effect. Professors and teaching assistants can assist students with challenging exercise tasks assigned during a lecture. However, individualized instruction is not an option when students work on assignments or are working asynchronously. Consequently, if the assignment being judged online is too difficult, students may quit without deep thinking and not gain the desired learning effect.

Traditional online judges only evaluate whether the submitted code meets the specifications. They lack a mechanism to support students who cannot solve problems without assistance. Although model answers may be helpful, assignments are often given for a grade. In this situation, model answers to assignments are not published until after the deadline. On the other hand, model answers are available anytime when an assignment is not used to evaluate students' academic performance. In this case, students tend to read the model answers when facing challenging problems without deep thinking. Hence, students have difficulty understanding the model answers deeply because there are no opportunities to apply the model answers to other problems.

Instead of providing a model answer for a difficult assignment, this paper proposes an approach that shares the content of a problem similar to the assignment (hereafter, similar problem) and an example of the answer source code of the similar problem (hereafter, reference code). By reading and understanding the similar problem content and reference code, students can learn how to solve challenging assignments. Because the assignment answer code and reference code are similar but not identical, students must solve challenging assignments independently, even if they look at similar problems or reference code. Thus, they are likely to understand assignments with the support of similar problems and reference code.

To verify the effectiveness of our approach, we developed a system named Waseda online judge recommender (WOJR). WOJR recommends similar problems, and it works with an existing online judge named Waseda online judge (WOJ), which our university employs. WOJR uses model answers for an assignment to search for similar problems and their reference code, which are available in other online judges (hereafter, problem repositories). These search results are presented as candidates to the professor. The professor selects the appropriate similar problems and reference code. Students can then read similar problems and reference code to better understand the implementation of an algorithm concept. Therefore, students continue to learn without giving up on an assignment.

This paper aims to answer the following two research questions (RQs):

RQ1. How many students view similar problems and reference code provided by WOJR?
RQ2. Does WOJR improve the number of correct answers in students' assignments?

To answer these research questions, we conducted an intervention experiment, which provided WOJR in a course on algorithm exercises using an online judge. We employed Aizu Online Judge (AOJ) [2] as the problem repository to find similar problems and reference code. We compared the course results using WOJR (intervention year) to the same course without intervention (non-intervention year). The number of correct answers in the intervention year is statistically improved compared to that in the non-intervention year, suggesting that our approach can improve students' understanding of an assignment and enhance the educational effect.

The contributions of this paper are as follows:

- We proposed an approach that finds similar problems based on the similarity between model answers instead of the similarity of problem sentences to assist in students' understanding of difficult assignments.
- We developed WOJR to implement the above approach.
- We conducted an intervention experiment and confirmed that students provided statistically significantly more correct answers to assignments in the intervention year.

2. Recommendation System

2.1. Overview

In conventional courses where an assignment is solved after a lecture, the difference in comprehension required to answer correctly can be very large. One option to overcome this gap is to reduce the assignment difficulty. However, this changes the content and may even reduce what can be learned. The proposed approach aims to reduce the gap without changing the content by suggesting a problem and its correct answer similar to the assignment.

WOJR helps professors provide appropriate similar problems and reference code to students. Figure 1 shows an overview of the procedure by which WOJR helps professors.

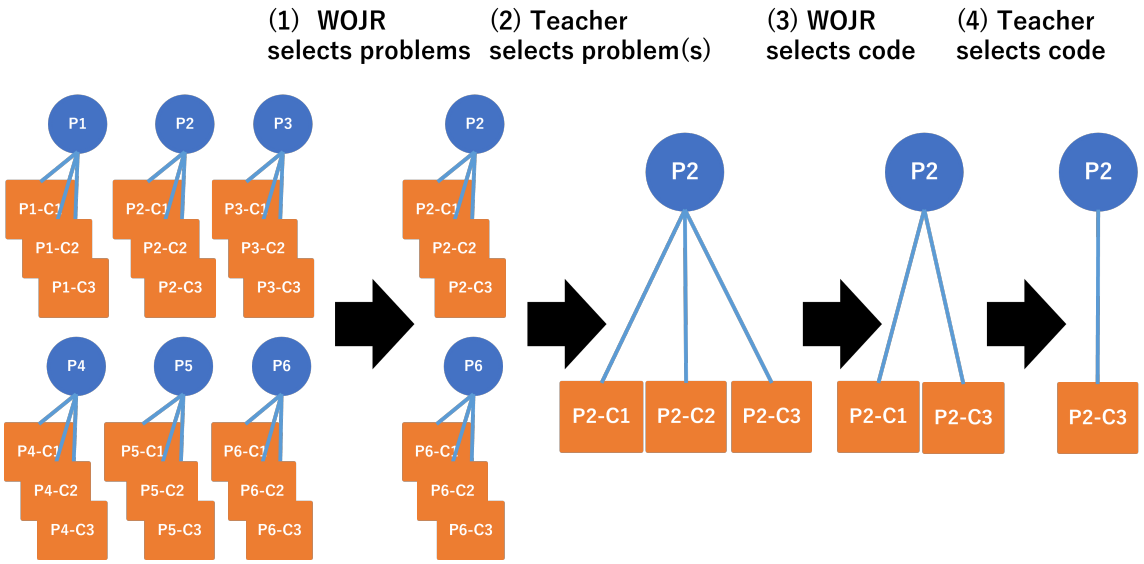

Figure 1. Procedure that selects similar problems and reference code with WOJR.

The scenario where a professor provides similar problems and reference code with WOJR is as follows:

1. The professor creates an assignment and registers the answer code for the assignment in WOJR.
2. WOJR presents candidates of similar problems (Figure 1(1)).
3. The professor selects some similar problems in WOJR (Figure 1(2)).
4. WOJR presents candidates of the reference code for each similar problem (Figure 1(3)).
5. The professor selects reference code (Figure 1(4)).
6. Students try to solve the assignment. When students fail to solve it, they read the similar problems and the reference code.
7. Students solve the assignment using ideas from the reference code for the similar problems.

Figure 2 shows the architecture of WOJR and related systems. WOJ is an assignment system at Waseda University, consisting of a web application and a database. WOJR retrieves assignments through the WOJ database. We employ AOJ, which is an existing online judge, as a problem repository. WOJR calls AOJ APIs to retrieve problems and answer code. WOJR is also a web application with a database and a term frequency–inverse document frequency (TF–IDF) module. The TF–IDF module calculates the similarity between the assignment answer code and reference code.

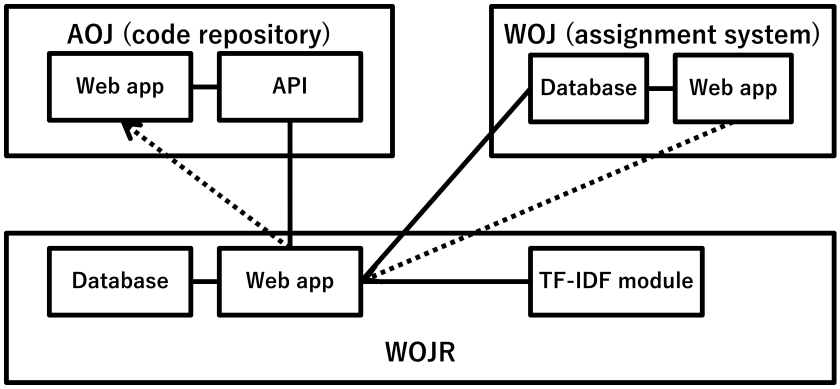

Figure 2. Architecture of WOJR and the related systems. Solid line indicates the two components communicate. Dotted line indicates a component links to webpages on another component.

Figure 3 shows a screenshot of the page where students read a similar problem and reference code. Note that the pages are written in Japanese and that we translated them into English in this paper. Students can read problem sentences and open the source page of the problem via a link. Students can also read reference code written in C, C++, and Java by clicking the menu on the left side. Students can push the "Helpful" and "Not helpful" buttons to give the professors feedback on whether the similar problem and reference code are helpful or not.

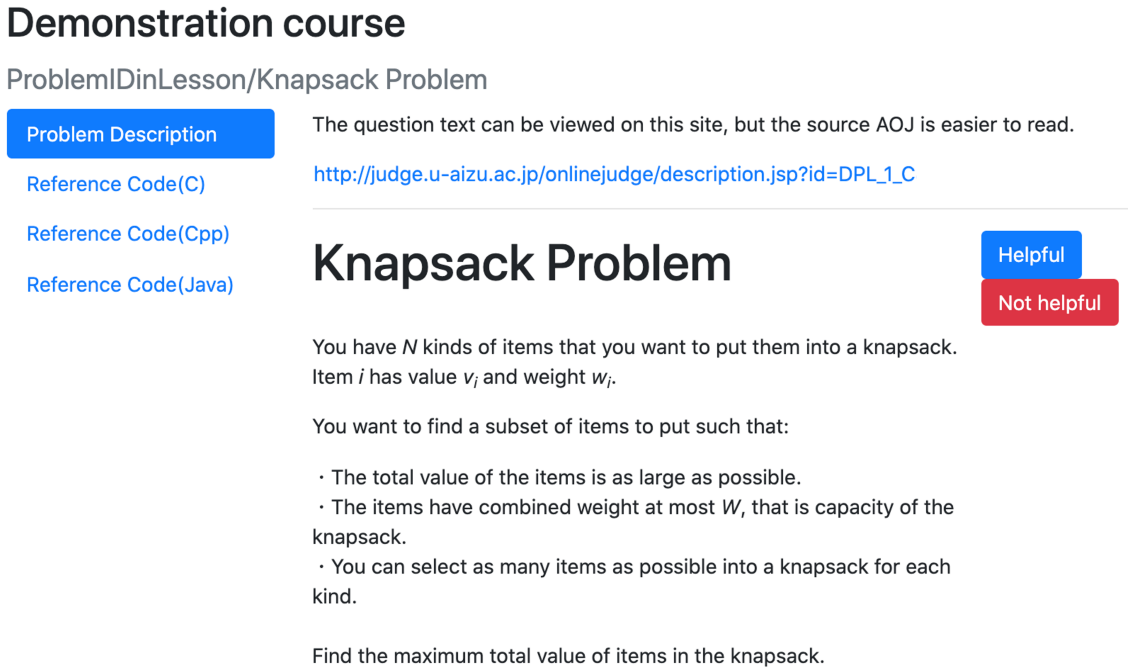

Figure 3. Screenshot of the page where students read a similar problem and reference code. The url in the figure is https://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=DPL_1_C (accessed on 25 February 2022).

The TF–IDF module calculates similarity scores between the assignment problem and the candidates. WOJR presents similar problems and their reference code from the problem repository for each assignment given by WOJ. Although WOJ is used in this study, WOJR can also support other existing online judges or similar systems. WOJR currently supports three programming languages: C, C++, and Java. WOJR provides students links to AOJ webpages as the source of similar problems and reference code.

We assume that problems requiring similar solutions help students understand a difficult problem and assume that the similarity of answer code is more important than the similarity of problem texts from the perspective of similarity of solutions. For example, Problem B helps students understand how to solve Problem A if the answer code contents for Problems A and B are similar. Thus, WOJR calculates the similarity score between two source code contents based on the TF–IDF method and cosine similarity. To determine the similarity, we also consider the model answer code, the correct answer code of previous learners, and the code fragment written by a professor as the answer code.

Although the TF–IDF method is traditionally used to calculate the similarity of document genres, it has recently been used to detect code clones [3], to cluster submitted code in competitive programming [4], and to extract characteristic parts in the source code [5]. The TF value indicates the occurrence frequency of the word of interest in the target document. The IDF value is the reciprocal of the occurrence frequency of the word of interest in the document, which is set as a population. Thus, the TF–IDF value is calculated as the product of the TF value and the IDF value [5]. WOJR calculates the TF–IDF value of each source code using `TfidfVectorizer` of `scikit-learn` (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html, accessed on 25 February

2022). Although `scikit-learn` does not support distributed processing, the calculation algorithm of the TF–IDF value can be implemented using distributed processing. If performance issues occur in the future, we can employ a framework for distributed processing (e.g., Spark).

2.2. Selection of Similar Problems

WOJR evaluates the candidates of similar problems (hereafter, similar problem candidates) from two perspectives: (1) maximum and (2) average similarity between the answer code of an assignment and a set of answer code of a similar problem candidate.

Suppose there is model answer code of an assignment (Figure 4) and answer code of a similar problem candidate (Figure 5). WOJR implements the following procedure to calculate similarity:

```
int add(int a, int b) {
    return a + b;
}
```

Figure 4. Fragment of model answer code of an example assignment (adding two integers).

```
int subtract(int a, int b) {
    return a − b;
}
```

Figure 5. Fragment of answer code of an example similar problem (subtracting two integers).

1. Divide each content of the answer code into tokens. A token consists of a set of unigrams, bigrams, and trigrams, excluding blanks and separator symbols (e.g., $[(int), (add), (int), (int, add), (add, int), (int, add, int)]$ for Figure 4).
2. Count the frequency of each token (e.g., $[(a) : 2, (a, int) : 1, (a, int, b) : 1, (add) : 1, (add, int) : 1, \dots]$ for Figure 4).
3. Calculate a TF–IDF feature vector for each content of the answer code. Note that the given answer code of the assignment and all the answer code in code repositories are defined as a whole set of documents in the calculation of IDF values (e.g., $[(a) : 0.14, (a, int) : 0.09, (a, int, b) : 0.17, (add) : 0.26, (add, int) : 0.31 \dots]$ for Figure 4 when the whole set consists of Figures 4 and 5).
4. Calculate the cosine similarity value between the content of the answer code of the similar problem candidate and the answer code of the assignment.
5. Calculate the maximum and average similarity scores between each similar problem candidate and the assignment based on the cosine similarity values.
6. Display two lists (one for the maximum and one for the average values) of the similar problem candidates in descending order of similarity scores. Then, the professor selects similar code.

Figure 6 explains steps 4–6 in detail using the assignment problem P1 and similar problem code PX as an example. The professor can feed multiple contents of the answer code of an assignment into WOJR. Consider the case where the professor feeds three answer contents (e.g., P1-C1, P1-C2, and P1-C3) into WOJR and where a similar problem candidate has three answer code contents (e.g., PX-C1, PX-C2, and PX-C3). There are nine possible pairs of answer code contents of P1 and PX. The two similarity scores for each pair are calculated as described above. Suppose each similarity ranges between 0.1 and 0.9. The maximum and average values of the nine similarities are calculated as 0.9 and 0.5. The two values are the similarity scores between P1 and PX.

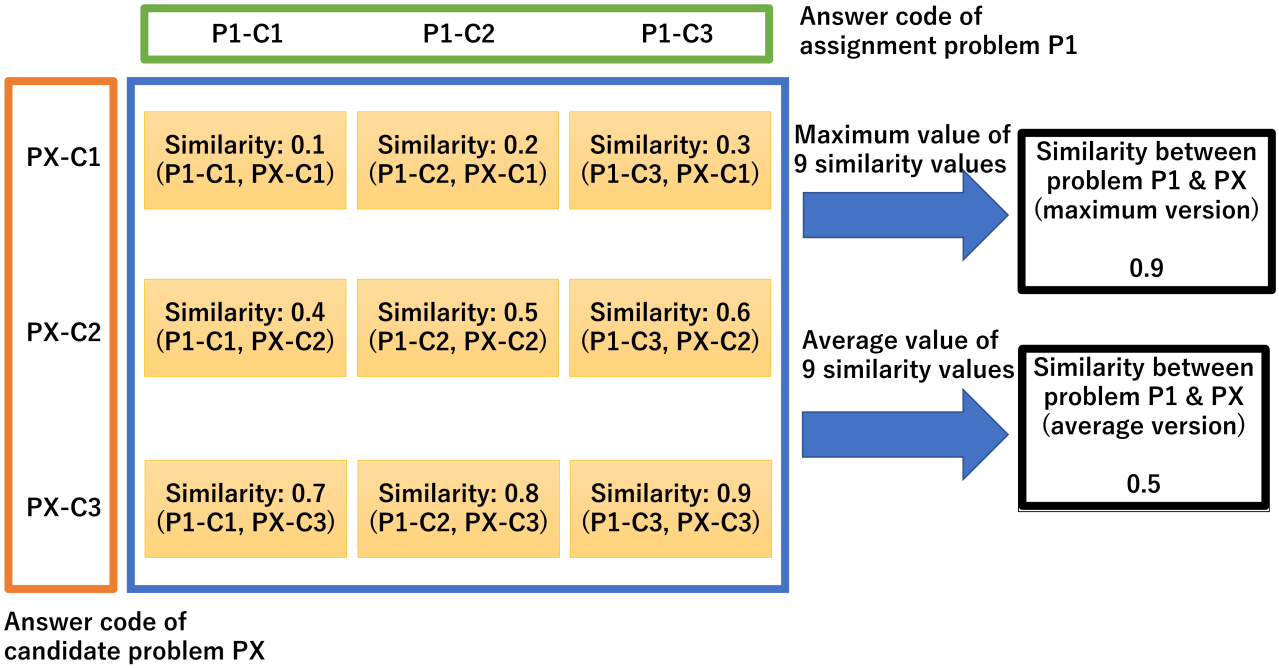

Figure 6. How to calculate problem similarity scores from a set of similarity scores between model answer code and reference code.

The TF–IDF value depends on whether a pair of source code contains the same words or not. Thus, TF–IDF is effective in finding a pair of source code that employs a specific data structure (e.g., stack) and a specific algorithm (e.g., sort). However, it is difficult for TF–IDF to find a source code pair with non-common words. In other words, TF–IDF cannot consider structural similarity. For example, Figure 7 solves the same problem as Figure 4, so the similarity score should be the highest. However, "addTwoValues", "x", and "y" are different from "add", "a", and "b", so the similarity score between Figures 4 and 7 is less than the similarity score between Figures 4 and 5.

```
int addTwoValues(int x, int y) {
    return x + y;
}
```

Figure 7. Fragment of answer code of a similar example problem whose similarity score is low.

There are also techniques that judge whether a pair of source code is structurally similar or not. For example, plagiarism detection and code clone detection can consider structural similarity. Since students who plagiarize other students' source code rename identifiers (e.g., variable names) to hide their plagiarism, plagiarism detection tools are robust to wording [6]. Code clones are similar or identical fragments of code and are classified into four types [7]. Since the types 2, 3, and 4 of code clones do not depend on the names of identifiers, tools for detecting code clones are also robust to wording [6]. Although plagiarism detection tools provide similarity scores, which can be alternatives to TF–IDF scores, code clone detection tools do not provide similarity scores. Thus, we evaluate how effectively plagiarism detection tools can find similar problems and reference code.

2.3. Selection of Reference Code

To recommend reference code candidates, WOJR calculates a score of the source code metric, which indicates the simplicity of the source code. We assume that the semantics of a simple code are easier to read and understand. We employ $max(1000 - 10 * [cyclomatic\_complexity] - [the\_number\_of\_tokens], 0)$ as a metric based on both cyclomatic complexity (CC) and the number of tokens. Although this metric can be replaced, the defi-

nition of the metric is not the essence of this paper. There are two reasons for selecting this metric. First, existing studies have shown that CC and software quality are correlated [8]. Second, students seem more likely to quit if the browsed code is long. Note that WOJR excludes source code less than five lines to eliminate extreme code that aims to reduce the source code by up to 1 byte.

The procedure to select reference code is as follows. First, the system registers the source code on AOJ in the database in advance. WOJR uses an existing tool called Lizard (https://github.com/terryyin/lizard, accessed on 25 February 2022) to calculate the metric scores based on the CC and the number of tokens in each source code. WOJR stores the calculation results in the database. If the professor requests reference code for a professor-selected similar problem, the source code is presented as a list of reference code candidates on the professor's side in descending order of the metric score.

3. Evaluation

3.1. Experiment Settings

We conducted an experiment in a course centered on algorithm exercises at Waseda University in Japan to evaluate WOJR. The control group took the course one year before the intervention. The same professor taught both the intervention course (2019) and the non-intervention course (2018) before COVID-19 became widespread. A total of 80 students and 89 students took the intervention and non-intervention courses, respectively. A prerequisite for all students was completing an introductory course on algorithms. In the intervention year, interventions occurred during the first five lectures. The questionnaire was prepared by referencing the literature [9,10] and administered anonymously during the fifth lecture.

3.2. Results

We compared the number of correct answers between the non-intervention and intervention courses. Of the 80 students enrolled in the intervention course, 52 (65%) students completed the questionnaire, and 19 (23.8%) indicated that they used WOJR.

Table 1 shows the number of students with all correct answers and the average number of correct answers by year. Although the difficulty of the assignments was the same in the intervention year and the non-intervention year, some of the problems were slightly different. The intervention involved nine problems, which were similar in both the non-intervention and intervention years. Problems that were drastically different, too easy, or whose reference code was too similar to the answer code of the assignments were excluded. The average number of correct answers increased from 8.37 to 8.77, and the average percentage of correct answers increased from 93.0% to 97.4%. This is about a 4.4% improvement. The percentage of students who answered all the problems correctly increased from 79.8% to 91.3%. This is about an 11.5% improvement.

Table 1. Changes in the number of correct answers to all problems and the average number of correct answers. Each year has nine problems. Problem contents are almost identical between the intervention and non-intervention years.

	Non-Intervention Year	Intervention Year
Number of Students	89	80
Number of Students Who Answered All Problems Correctly	71	73
Average Number of Correct Answers	8.37	8.77
Percentage of Students Who Answered All Problems Correctly	79.8%	91.3%

A two-sided test was used in the Mann–Whitney U test [11] to verify whether the difference was statistically significant. We used this test because it is unbiased in assuming a normal distribution for the number of correct answers. This non-parametric test examines significant differences between two unpaired groups without assuming a population distribution. For the test, we used the number of correct answers. Since the two groups took the course in different years, there is no correspondence. A significance level of 0.05,

which is generally used in education, was employed. The obtained p-value was 0.033. Thus, the number of correct answers improved significantly in the intervention year compared to that in the non-intervention year.

In the questionnaire, we asked only the students who answered that they used WOJR what they liked, and what they would improve. We received responses from six students, which are summarized below:

- There is a similar problem whose statement is difficult to understand.
- It was helpful to refer to similar problems and algorithms with similar ideas.

We also asked all the students what kind of functionality they wanted. We received responses from 6 students who used WOJR and 14 students who did not use WOJR, which are summarized below:

- I want the degree of similarity with the assignment to increase.
- Please explain the algorithm using figures.

3.3. Examples of Assignments, Similar Problems, and Reference Code

Here, we show examples of assignments, similar problems, and reference code. The first example is calculating the number of routes in a rectangular area by dynamic programming. Figure 8 shows the model answer written in C. WOJR offered similar candidates, and the knapsack problem was adopted in the experiment. Figure 9 shows the reference code provided by WOJR. Since the model answer and reference code are similar, students can use these as references to solve the assignment.

```
for (int i = 0; i < H; i++) {
    for (int j = 0; j < W; j++) {
        if (ok[i][j] == 'X') continue;

        if (i != 0) dp[i][j] += dp[i−1][j];
        if (j != 0) dp[i][j] += dp[i][j−1];
    }
}
```

Figure 8. Fragment of model answer code of example assignment 1 (calculating the number of routes in a rectangular area by dynamic programming) written in C.

```
for (i = 0; i < N; i++) {
    for (j = 0; j <= W; j++) {
        if (w[i] > j) dp[i+1][j] = dp[i][j];
        else dp[i+1][j] = max (dp[i][j], dp[i+1][j−w[i]]+v[i]);
    }
}
```

Figure 9. Fragment of reference code of similar problem 1 (solving the knapsack problem by dynamic programming) written in C.

The second example is as follows: "Given a sequence, select the pair with the smallest difference and output the absolute value of their difference." Figure 10 shows the code for part of the answer example for this assignment. WOJR was given a similar problem: "Given a sequence, output minimum, maximum, and their total values." Figure 11 shows the code for part of the answer. There is enough similarity that the reference code may be helpful for the assignment.

```
int result = abs(a[0]−a[1]);
for(int i=0; i<n; i++) {
    for (int j=i+1; j<n; j++) {
      result = min(result, abs(a[i] − a[j]));
    }
}
cout << result << endl;
```

Figure 10. Fragment of model answer code of example assignment 2 (selecting the pair with the smallest difference from the given array) written in C.

```
int result = a[0];
for(int i=1; i<n; i++){
    result = min(result, a[i]);
}
cout << result;
```

Figure 11. Fragment of reference code of similar problem 2 (selecting the minimum value from the given array) written in C.

3.4. Similarity Metrics and Problem Repositories

TF–IDF considers the similarity of token sequences, but it does not consider structural similarity. The structural similarity may also be effective in finding similar problems and reference code. JPlag is the most famous plagiarism detection tool that considers the structural similarity, and it can show similarity scores [6,12]. Thus, we compared TF–IDF with JPlag in the second experiment.

The size of a set of problem repositories may affect WOJR performance because WOJR cannot recommend similar problems and reference code that do not exist in the problem repositories. We conducted the first WOJR experiment with AOJ, but AtCoder is larger than AOJ, and the submitted code is also available in AtCoder. Thus, we also compared AOJ with AOJ and AtCoder.

To evaluate how the similarity metrics and the sizes of problem repositories affect WOJR performance, we implemented the two similarity metrics (TF–IDF and JPlag) for Python language. We also made WOJR support the two sets of problem repositories (AOJ and AOJ + AtCoder). WOJR supports the two aggregation operators (maximum and average) to calculate a similarity score of a similar problem from similarity scores between answer code and reference code candidates. We selected the top five similar problems with reference code with respect to each combination of the two similarity metrics, the two sets of problem repositories, and the two aggregation operators. Note that we excluded too similar (almost same) problems from the top five similar problems because students can change the reference code of the problem to the model answer code easily without thinking.

We selected all the nine problems that we used in the intervention experiment and added the six problems that we did not use in the intervention experiment but the professor provided after the interventions in his lecture. Since we expected that TF–IDF would not be suitable to find similar code that had few common words but a similar structure, we selected the six problems that asked about tree and graph algorithms. Contrary to TF–IDF, we expected that JPlag could find structurally similar code for tree and graph algorithms. Table 2 shows the algorithms (or data structures) that selected assignment problems require students to implement.

We recruited a programming lecturer who is an expert in algorithms and data structures. The authors and the lecturer judged whether or not each similar problem with reference code recommended by each combination was valid for students with the following standard. For example, General 1 requires students to implement a stack. If problems require other data structures, such as a pure array or a queue, the problems are not similar to General 1.

- Both an assignment problem and a similar problem require the same algorithm or data structure.
- A similar problem is simple and easy to understand. It is not too difficult in comparison to an assignment problem.

Table 2. Algorithms that selected assignment problems require students to implement.

Assignment	Algorithm or Data Structure
General 1	Stack
General 2	Doubly linked list
General 3	Binary search
General 4	Set
General 5	Divide and conquer
General 6	Enumerating 2-combinations
General 7	Depth-first search
General 8	Subset sum problem (dynamic programming)
General 9	Recursion
Tree 1	Tree traversal
Tree 2	Binary search tree
Tree 3	Max heap
Graph 1	Minimum spanning tree
Graph 2	Dijkstra's algorithm
Graph 3	Union find

Table 3 shows the numbers of valid similar problems with reference code. The authors and the lecturer agreed with the results. The maximum value of a cell is five because we selected the top five similar problems. The total number of similar problems with reference code including duplication is 600 ($15 \times 5 \times 2 \times 2 \times 2$) since there are combinations of the 15 problems, the top 5 similar problems, the 2 similarity metrics (TF–IDF and JPlag), the 2 sets of problem repositories (AOJ and AOJ + AtCoder), and the 2 aggregation operators (maximum and average). Overall, both TF–IDF with only AOJ using the average operator and TF–IDF with AOJ and AtCode using the maximum operator are the most capable of providing similar problems.

Table 3. Numbers of valid similar problems with reference code with respect to each combination of the two similarity metrics, the two sets of problem repositories, and the two aggregation operators. The highlighted cells indicate that the combination is the best performing.

	Average				Max			
Assignment	**TF–IDF (AOJ)**	**TF–IDF (All)**	**JPlag (AOJ)**	**JPlag (All)**	**TF–IDF (AOJ)**	**TF–IDF (All)**	**JPlag (AOJ)**	**JPlag (All)**
General 1	4	5	0	0	3	5	0	0
General 2	2	1	0	1	0	0	1	1
General 3	1	1	0	0	1	1	1	1
General 4	2	2	3	3	1	1	2	1
General 5	1	1	0	0	1	1	0	0
General 6	2	1	0	0	1	1	1	1
General 7	2	1	4	4	2	3	3	3
General 8	0	0	0	0	0	2	0	1
General 9	0	0	0	0	1	0	0	0
Tree 1	0	0	0	0	1	0	2	0
Tree 2	2	2	1	0	0	0	1	1
Tree 3	0	0	0	0	1	1	0	0
Graph 1	0	0	0	0	0	0	0	0
Graph 2	2	2	2	1	1	0	1	3
Graph 3	2	2	1	2	3	5	2	5
Total	20	18	11	11	16	20	14	17

We also recruited a student who was studying computer science at a university, and asked him whether each similar problem with reference code was helpful for him to solve the assignment problems or not. Table 4 shows the results. Overall, TF–IDF with only AOJ using the maximum operator is the best, but the difference in the number of Yes answers is only one in comparison to the others using the maximum operator.

Table 4. A summary indicating whether each similar problem with reference code is helpful to solve the assignment problem or not. Yes and No indicate helpful and not helpful, respectively. Nothing indicates that no valid similar problem exists. The highlighted cells indicate that only one of TF–IDF or JPlag found helpful similar problems among the results of the average and maximum operators.

	Average				Max			
Assignment	**TF–IDF (AOJ)**	**TF–IDF (All)**	**JPlag (AOJ)**	**JPlag (All)**	**TF–IDF (AOJ)**	**TF–IDF (All)**	**JPlag (AOJ)**	**JPlag (All)**
General 1	No	No	Nothing	Nothing	Yes	Yes	Nothing	Nothing
General 2	Yes	Yes	Nothing	No	Nothing	Nothing	Yes	No
General 3	Yes	Yes	Nothing	Nothing	Yes	Yes	Yes	Yes
General 4	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
General 5	Yes	Yes	Nothing	Nothing	Yes	Yes	Nothing	Nothing
General 6	Yes	Yes	Nothing	Nothing	Yes	No	No	No
General 7	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
General 8	Nothing	Nothing	Nothing	Nothing	Nothing	No	Nothing	Yes
General 9	Nothing	Nothing	Nothing	Nothing	No	Nothing	Nothing	Nothing
Tree 1	Nothing	Nothing	Nothing	Nothing	No	Nothing	No	Nothing
Tree 2	No	No	No	Nothing	Nothing	Nothing	No	No
Tree 3	Nothing	Nothing	Nothing	Nothing	Yes	Yes	Nothing	Nothing
Graph 1	Nothing	Nothing	Nothing	Nothing	Nothing	Nothing	Nothing	Nothing
Graph 2	Yes	Yes	Yes	Yes	Yes	Nothing	Yes	Yes
Graph 3	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Number of Yes	8	8	4	4	9	7	6	6

TF–IDF found helpful similar problems for General 1, General 5, General 6, and Tree 3, but JPlag did not find similar problems for them. JPlag wrongly recommended non-similar problems for General 1, General 5, General 6, and Tree 3, which requires students to implement array manipulation (not stack), aggregation operations such as average or maximum (not divide and conquer), finding an integer pair whose difference is a given integer (difficult version of enumerating 2-combinations), and comparison of array items (not max heap), respectively. In contrast, JPlag provided a helpful similar problem for General 8, but TF–IDF did not find helpful similar problems for it. TF–IDF wrongly recommended non-similar problems for General 8, which requires students to implement a calculator of number of cases using dynamic programming (not subset sum problem).

4. Discussion

4.1. RQ1: How Many Students View Similar Problems and Reference Code Provided by WOJR?

Of the 80 students in the intervention year, 19 used WOJR, which is about 25% (RQ1). Although 25% seems like a small percentage, not all students were expected to use WOJR because it is a support mechanism for students who cannot solve the assignments. Assuming that both groups had a constant percentage of students who answered all the assignments correctly (Table 1), 79.8% of the students in the intervention year should not require intervention to answer all the assignments correctly. Hence, we estimated that about 20% of students would be unable to solve all the problems by themselves. Thus, the fact that about 25% of the students used WOJR indicates that it was accessed appropriately.

4.2. RQ2: Does WOJR Improve the Number of Correct Answers in Students' Assignments?

Our analysis included all the students enrolled in the course, even those who did not use WOJR, to minimize the effect of two biases. The first bias is that WOJR is more valuable

for students who cannot solve the problems by themselves. We speculated that students who did not use WOJR would tend to have low skills and low academic performance with respect to the first bias. The second bias is that enthusiastic students are more likely to try using new systems and methods, including WOJR. We hypothesized that such students would have high skills and academic performance due to their high learning motivation. Consequently, we included all enrolled students to eliminate these two biases.

Compared to the non-intervention year, the average number of correct answers increased by about 0.4 for the nine assignments in the intervention year, which is about a 4.4% improvement in the correct answer rate. There is a statistically significant difference between the non-intervention year and intervention year, indicating that WOJR improved the number of correct answers (RQ2). The percentage of students who answered all the problems correctly increased by about 11.5%. The value of 11.5% is relatively large.

In general, improving the number of correct answers by providing similar problems and answers does not necessarily mean an educational effect because similar problems may work just as hints. However, we consider this improvement an educational effect for the following three reasons.

(1) The selected reference code differed from the answer code of the assignment. Therefore, students must solve assignments by understanding similar problems and reference code, and then applying the knowledge to the assignment. (2) Difficult problems still make students think deeply even if similar problems and reference code are provided. For example, famous programming contests like the ACM International Collegiate Programming Contest (ICPC) allow contestants to read any materials, including similar problems and reference code. The number of correct answers to ICPC problems can precisely reflect the contestant's programming skills, that is, the better contestants can solve more ICPC problems. (3) Worked examples, including similar problems and reference code, are well studied. Many researchers have revealed the educational effects of worked examples [13–16]. For example, Hattie found that the effect size of educational effects of worked examples was 0.57 (The latest analysis of Hattie (https://visible-learning.org/wp-content/uploads/2018/03/VLPLUS-252-Influences-Hattie-ranking-DEC-2017.pdf, accessed on 25 February 2022) shows the effect size is 0.37) [17].

On the other hand, a ceiling effect is likely to exist because about 80% of the students solved all the assignments correctly before the intervention. Targeting more challenging assignments should increase the improvement rate of WOJR.

4.3. Similarity Metrics and Problem Repositories

We considered three factors in WOJR performance: the two aggregation operators, the two similarity metrics, and the two sets of problem repositories.

In the second experiment, the maximum operator is better overall than the average operator since the average operator is not robust to noisy reference code. For example, if a similar problem contains several submissions of very similar reference code and many submissions of not similar reference code, the average operator concludes that the problem is not similar. In JPlag, this tendency is remarkable. The number of helpful similar problems of JPlag with average and maximum operators are four and six, respectively.

We also find that the set of AOJ and AtCoder is overall better than only AOJ with respect to the number of valid similar problems since the size of the set is larger than the size of only AOJ when we employ the maximum operator. In our dataset, the numbers of problems in AOJ and AtCoder are 1222 and 2656, respectively. Since the average operator is not robust to noise, the differences in the repository sets are small when we employ the average operator. However, we find no clear difference in the repository sets with respect to the number of helpful similar problems. Thus, we conclude that AOJ alone is enough to find helpful similar problems.

Although the number of valid similar problems found by TD–IDF is higher than JPlag, there is one assignment problem for which only JPlag can find a helpful similar problem. The goal of the assignment problems in the lecture is to help students learn

representative data structures and algorithms in computer science. These data structures and algorithms have well-known names such as stack, heap, queue, and sort. TF–IDF tends to outperform JPlag when finding similar problems for such data structures and algorithms. However, some algorithms such as dynamic programming (e.g., General 8) do not have a specific keyword in source code. It is difficult for TF–IDF to find similar problems for such algorithms. Therefore, we conclude that we should employ multiple similar metrics and merge multiple sets of found similar problems to deal with various algorithms.

4.4. Questionnaire Results

Finally, we considered the free responses to the questionnaire. Some students found referring to problems with similar ideas helpful, suggesting that WOJR worked as designed. However, other students commented that the problem statement for a similar problem was difficult to understand. The reason may be because AOJ was adopted as the search destination for similar problems, and AOJ contains many past problems such as the ACM ICPC, which have problem statements written in a complicated manner. Therefore, expanding the problem collection during the search for similar problems may overcome this issue.

There are two possible factors influencing the low similarity to the assignments. The first factor is the problem set. As mentioned above, AOJ contains many problems from past programming contests, some of which are complex. Since this course deals with more straightforward problems than those in programming contests, no similar appropriate problems may exist. Therefore, extending problem repositories containing less complex problems may improve the results. The second factor is the similarity calculation method. WOJR calculates the similarity of the problem using the similarity of the source code via TF–IDF. The experiment revealed very few similar problems related to tree structures and graphs because their code tends to have diverse tokens, and TF–IDF does not work well in such a situation. Because TF–IDF mainly reflects the token similarity, calculating the similarity utilizing existing code clone detection techniques [18–20], which consider the structural viewpoint, may improve the performance. In the second experiment, there are helpful similar problems that only TF–IDF can find, and ones that only JPlag can find. Therefore, we think we need to combine multiple similarity metrics to recommend similar problems.

A two-step narrowing may be necessary since the calculation cost of similarity considering the structural viewpoint is generally high. The similar problem candidates should be initially narrowed using the similarity, which has a small calculation cost. Then, the final candidates should be determined from the narrowed candidates, which have a higher calculation cost.

Finally, some students indicated that they would like an explanation of the algorithm. Existing online judges only contain problems and answer code. Consequently, such an explanation is beyond the scope of WOJR.

5. Limitations

This study has three limitations: (1) requirements for students, (2) the dependencies of the diversity in the problem repositories, (3) the professors' skills, and (4) threats to validity.

WOJR provides similar problems and reference code. Students must have the skills to understand the given problems and code to utilize them to solve assignments. Otherwise, the educational effects of WOJR are very limited.

WOJR recommends similar problems and reference code from the given problem repositories. The educational effects of WOJR depend on the quality of similar problems and reference code. In other words, the educational effects depend on the content diversity in the problem repositories. If problem repositories do not contain a similar problem, then WOJR is ineffective. Adding more problem repositories should overcome this issue.

Professors select similar problems and reference code from the system-selected problems and code in WOJR. In addition, the recommended system-selected problems correspond to the query code fed by the professor into WOJR. The educational effect depends on

the quality of the professor-selected problems and the code. Consequently, the professors' skills impact the educational effect.

Threats to Validity

We compared the results of a course taught by the same professor in two different years (2018 and 2019). The students in each course did not represent the same population because the courses were offered in different years. Additionally, the contents and assignments varied slightly by year. These differences are threats to internal validity. To alleviate this, the experiment should be repeated by splitting students enrolled in the same course for a given year into intervention and non-intervention groups.

This experiment was conducted in a university setting. It is unclear if similar results will be obtained for other populations (e.g., different ages, regions, and educational institutions). This is a threat to external validity. To alleviate this threat, the experiment should be repeated while targeting different populations.

We experimented with a set of the same assignments in a course. It is also unclear if similar results will be obtained for other assignments. This is another threat to external validity. To alleviate this threat, the experiment should be repeated using different target assignments.

6. Related Work

Many studies have helped students learn algorithms. In particular, various visualization approaches have been proposed [21–28]. The existing studies provide students with visualization tools to help them understand how algorithms work. WOJR provides similar problems and reference code to help students learn how to solve problems (mainly algorithm problems). The existing studies can be combined with WOJR to understand how reference code solves similar problems.

We focus on online judges, and there are various studies related to online judges. Wasik et al. surveyed existing studies related to online judges and classified online judges according to their principal objectives [29]. We can classify our approach as enhancing education in their taxonomy. Fonte et al. proposed an evaluation method for online judges [30]. The method can grade partially correct answers by calculating semantic similarity between expected outputs and outputs generated by the submission code. Combéfis et al. proposed a taxonomy of educational programming games and classified several online programming games and platforms, including online judges [31]. Although those studies and our study have the same goal of enhancing programming education, the approaches are different and can be integrated.

There are also studies that proposed methods for presenting existing correct submission code and repairing wrong submission code automatically based on the analysis of existing correct submission code or model answer code. Fujiwara created a source code presentation system called TAMBA [4]. TAMBA presents the correct answer code submitted by another learner when the online judge determines that the submitted answer does not meet the specifications. Wang et al. proposed a data-driven program repair framework to automate the repair of wrong submission code by leveraging a large amount of available student submission code [32]. Parihar et al. proposed a tool named Clara, which generates repair patches as small as possible by clustering correct submissions [33]. Perry et al. proposed a new clustering algorithm for student submissions of programming assignments, and their approach outperforms Clara [34]. Hu et al. proposed a tool named Refactory, which generates repair patches by refactoring model answer code with a smaller amount of correct submission code [35]. Although those approaches help students solve challenging programming problems, we consider them to be an extension of providing model answer code, and they may hinder deep thinking. Since we avoid providing model answer code, those approaches' concept is different from ours.

Recommendation systems have emerged to extract helpful information from too much information. In the field of education, recommendation systems are proposed

to recommend educational resources to students. Urdaneta-Ponte et al. conducted a systematic review of studies in recommendation systems for education [36]. The review included 98 articles from a total of 2937 found in primary databases. The survey paper referred to the study of Prisco et al., which recommends programming problems [37]. The comprehensive study of [37] employed ELO rating to recommend programming problems whose difficulties are appropriate [38]. Yera et al. proposed a recommendation system for programming problems using collaborative filtering [39]. Although recommendation systems for programming education exist, to the best of our knowledge, WOJR is the first system for recommending similar problems based on the similarity between model answer code and reference code.

As described in Section 2, there are alternative metrics to calculate similarity scores. Novak et al. conducted a systematic review of studies in plagiarism detection [6]. Plagiarism detection tools can find similar code from other students' code, and they can also show similarity scores. The authors identified the top five most known tools: JPlag [12], MOSS [40], Plaggie [41], SIM-Grune [42], and Sherlock-Warwick [43]. JPlag was compared the most in the reviewed papers [6], and it can consider structural similarity. Ragkhitwetsagul et al. evaluated 30 code similarity detection techniques and tools, including JPlag [44]. They found that JPlag was good overall and that JPlag offered the highest performance on boilerplate code. Therefore, we employed JPlag as an alternative metric to calculate similarity scores in the second experiment.

In another direction, there are studies that encourage students to think deeply. Cutts et al. reported a supplementary course with voluntary participation using paper and pencil called Thinkathon [45]. The authors found that some students submit answer code repeatedly without thinking deeply. The authors suggested that students who heavily relied on computers would have more profound learning by solving problems on paper. Yuan et al. offered a hybrid pair programming course [46]. Due to the differing abilities between pairs in pair programming, often, one student solved the tasks while the other did not think deeply. To overcome the issue, Yuan et al. proposed hybrid pair programming where students solved a problem individually, and then they solved an advanced version of the problem via pair programming.

7. Conclusions and Future Work

A lasting education effect may not be realized if the assignment is too difficult when using an online judge. Although model answers can help overcome this issue, they may lead to two subsequent problems. First, model answers cannot be used prior to the assignment deadline if the assignment is for an academic grade. Second, students using model answers tend to feel they understand without thinking deeply. To address these issues, we propose an approach that recommends problems and reference code with content similar to the assignment. To evaluate our approach, we implemented a system, which incorporates our approach, and experimented in a university course.

The number of correct answers during the intervention year showed a statistically significant improvement compared to the non-intervention year. The experiment suggested that our approach can reduce assignment difficulty and improve the educational effect. However, the student evaluation questionnaire noted two areas of improvement for our system. First, the degree of similarity between the assignment and the displayed problem should increase. Second, the comprehensibility of the problem sentence of the displayed problem should be enhanced.

We intend to refine our experiment design because the number of correct answers before the intervention was too high in the target course. Thus, the ceiling effect may have influenced the experiment results. Conducting a similar experiment in a different course where the number of correct answers prior to the intervention is low may reveal a more substantial educational effect.

Author Contributions: Project administration, R.Y., K.S., H.W. and Y.F.; software, R.Y.; writing—original draft, R.Y.; writing—review and editing, K.S., H.W. and Y.F. All authors have read and agreed to the published version of the manuscript.

Funding: This study received no external funding.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: The questionnaire data presented in this study are available on request from the corresponding author. The data are not publicly available because the respondents agree that only academic researchers access to the data. The other data presented in this study are unavailable because the subject agreed that only the statistically processed information is published.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

WOJR	Waseda online judge recommender
WOJ	Waseda online judge
AOJ	Aizu Online Judge
TF–IDF	Term frequency–inverse document frequency
CC	Cyclomatic complexity
ICPC	ACM International Collegiate Programming Contest

References

1. Kurnia, A.; Lim, A.; Cheang, B. Online Judge. *Comput. Educ.* **2001**, *36*, 299–315. [CrossRef]
2. Watanabe, Y. Online judge Development and Operation-Aizu Online Judge-. *Inf. Process.* **2015**, *56*, 998–1005.
3. Yamanaka, H. High-Speed Function-Based Code Clone Detection Method Using the TF–IDF Method and LSH Algorithm. Master's Thesis, Osaka University, Osaka, Japan, 2014.
4. Fujiwara, S. TAMBA: Gradual Source Code Recommendation System for Programming Learners. Master's Thesis, Nara Institute of Science and Technology, Nara, Japan, 2016.
5. Kobayashi, Y.; Mizuno, O. Characteristic partial extraction method in source code using N-gram IDF. In Proceedings of the Software Symposium 2017 in Miyazaki, Miyazaki, Japan, 6–8 June 2017; pp. 46–55.
6. Novak, M.; Joy, M.; Kermek, D. Source-Code Similarity Detection and Detection Tools Used in Academia: A Systematic Review. *ACM Trans. Comput. Educ.* **2019**, *19*, 1–37. [CrossRef]
7. Rattan, D.; Bhatia, R.; Singh, M. Software clone detection: A systematic review. *Inf. Softw. Technol.* **2013**, *55*, 1165–1199. [CrossRef]
8. Chen, E.T. Program Complexity and Programmer Productivity. *IEEE Trans. Softw. Eng.* **1978**, *SE-4*, 187–194. [CrossRef]
9. Kan, T. *Questionnaire Survey and Statistical Analysis That Can Be Understood Well with Actual Examples*; Natsume Co., Ltd.: Tokyo, Japan, 2011.
10. Iwasa, H.; Yadohisa, H. *"Questionnaire" for Class Evaluation and Market Research a Book That Can Be Used for Research and Analysis*; Shuwa System Co., Ltd.: Tokyo, Japan, 2009.
11. Ikeda, I. For those who use statistical tests without understanding II. *Chem. Biol.* **2013**, *51*, 408–417.
12. Prechelt, L.; Malpohl, G.; Philippsen, M. Finding plagiarisms among a set of programs with JPlag. *J. Univers. Comput. Sci.* **2002**, *8*, 1016–1038. [CrossRef]
13. Crissman, J.K. *The Design and Utilization of Effective Worked Examples: A Meta-Analysis*; The University of Nebraska-Lincoln: Lincoln, NE, USA, 2006.
14. Lui, A.K.; Cheung, Y.H.Y.; Li, S.C. Leveraging Students' Programming Laboratory Work as Worked Examples. *SIGCSE Bull.* **2008**, *40*, 69–73. [CrossRef]
15. Hosseini, R.; Akhuseyinoglu, K.; Petersen, A.; Schunn, C.D.; Brusilovsky, P. PCEX: Interactive Program Construction Examples for Learning Programming. In Proceedings of the 18th Koli Calling International Conference on Computing Education Research (Koli Calling'18), Koli, Finland, 22–25 November 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 1–9. [CrossRef]
16. Nainan, M.; Balakrishnan, B.; Ali, A.Z.M. Design of Worked Examples for Learning Programming: Literature Review. *Int. J. Instr. Technol. Soc. Sci.* **2021**, *1*, 8–16.
17. Hattie, J. *Visible Learning: A Synthesis of over 800 Meta-Analyses Relating to Achievement*; Routledge: Oxfordshire, UK, 2008.
18. Roy, C.K. Detection and Analysis of Near-Miss Software Clones. Ph.D. Thesis, Queen's University, Kingston, ON, Canada, 2009.
19. Khatoon, S.; Li, G.; Mahmood, A. Comparison and evaluation of source code mining tools and techniques: A qualitative approach. *Intell. Data Anal.* **2013**, *17*, 459–484. [CrossRef]

20. Zhao, G.; Huang, J. DeepSim: Deep Learning Code Functional Similarity. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018), Lake Buena Vista, FL, USA, 4–9 November 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 141–151. [CrossRef]
21. Deng, X.; Wang, D.; Jin, Q.; Sun, F. ARCat: A Tangible Programming Tool for DFS Algorithm Teaching. In Proceedings of the 18th ACM International Conference on Interaction Design and Children (IDC 2019), Boise, ID, USA, 12–15 June 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 533–537. [CrossRef]
22. Végh, L.; Stoffová, V. Algorithm animations for teaching and learning the main ideas of basic sortings. *Inform. Educ.* **2017**, *16*, 121–140. [CrossRef]
23. Saltan, F. The Impact of Online Algorithm Visualization on ICT Students' Achievements in Introduction to Programming Course. *J. Educ. Learn.* **2017**, *6*, 184–192. [CrossRef]
24. Velázquez-Iturbide, J.Á.; Hernán-Losada, I.; Paredes-Velasco, M. Evaluating the Effect of Program Visualization on Student Motivation. *IEEE Trans. Educ.* **2017**, *60*, 238–245. [CrossRef]
25. Ishizue, R.; Sakamoto, K.; Washizaki, H.; Fukazawa, Y. PVC.js: Visualizing C programs on web browsers for novices. *Heliyon* **2020**, *6*, e03806. [CrossRef] [PubMed]
26. Crow, T.; Luxton-Reilly, A.; Wuensche, B. Intelligent Tutoring Systems for Programming Education: A Systematic Review. In Proceedings of the 20th Australasian Computing Education Conference, Brisbane, QLD, Australia, 30 January–2 February 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 53–62.
27. Vrachnos, E.; Jimoyiannis, A. Design and Evaluation of a Web-based Dynamic Algorithm Visualization Environment for Novices. *Procedia Comput. Sci.* **2014**, *27*, 229–239. [CrossRef]
28. Sorva, J.; Karavirta, V.; Malmi, L. A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Trans. Comput. Educ.* **2013**, *13*, 1–64. [CrossRef]
29. Wasik, S.; Antczak, M.; Badura, J.; Laskowski, A.; Sternal, T. A survey on online judge systems and their applications. *ACM Comput. Surv.* **2018**, *51*, 1–34. [CrossRef]
30. Fonte, D.; da Cruz, D.; Gançarski, A.L.; Henriques, P.R. A Flexible Dynamic System for Automatic Grading of Programming Exercises. In Proceedings of the 2nd Symposium on Languages, Applications and Technologies, Porto, Portugal, 20–21 June 2013; Volume 29, pp. 129–144.
31. Combéfis, S.; Beresnevičius, G.; Dagiene, V. Learning Programming through Games and Contests: Overview, Characterisation and Discussion. In Proceedings of the International Scientific Conference eLearning and Software for Education, Bucharest, Romania, 21–22 April 2016; Volume 1, pp. 492–497.
32. Wang, K.; Singh, R.; Su, Z. Search, Align, and Repair: Data-Driven Feedback Generation for Introductory Programming Exercises. *SIGPLAN Not.* **2018**, *53*, 481–495. [CrossRef]
33. Parihar, S.; Dadachanji, Z.; Singh, P.K.; Das, R.; Karkare, A.; Bhattacharya, A. Automatic Grading and Feedback Using Program Repair for Introductory Programming Courses. In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, Bologna, Italy, 3–5 July 2017; pp. 92–97.
34. Perry, D.M.; Kim, D.; Samanta, R.; Zhang, X. SemCluster: Clustering of Imperative Programming Assignments Based on Quantitative Semantic Features. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, Phoenix, AZ, USA, 22–26 June 2019; pp. 860–873.
35. Hu, Y.; Ahmed, U.Z.; Mechtaev, S.; Leong, B.; Roychoudhury, A. Re-Factoring Based Program Repair Applied to Programming Assignments. In Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering, San Diego, CA, USA, 11–15 November 2019; pp. 388–398.
36. Urdaneta-Ponte, M.C.; Mendez-Zorrilla, A.; Oleagordia-Ruiz, I. Recommendation Systems for Education: Systematic Review. *Electronics* **2021**, *10*, 1611. [CrossRef]
37. Prisco, A.; Santos, R.D.; Bez, J.L.; Tonin, N.; Neves, M.; Teixeira, D.; Botelho, S. A Facebook chat bot as recommendation system for programming problems. In Proceedings of the 2019 IEEE Frontiers in Education Conference (FIE), Covington, KY, USA, 16–19 October 2019; pp. 1–5.
38. Prisco, A.; dos Santos, R.; Nolibos, A.; Botelho, S.; Tonin, N.; Bez, J. Evaluating a programming problem recommendation model—A classroom personalization experiment. In Proceedings of the 2020 IEEE Frontiers in Education Conference (FIE), Uppsala, Sweden, 21–24 October 2020; pp. 1–6.
39. Yera, R.; Martínez, L. A Recommendation Approach for Programming Online Judges Supported by Data Preprocessing Techniques. *Appl. Intell.* **2017**, *47*, 277–290. [CrossRef]
40. Bowyer, K.W.; Hall, L.O. Experience using "MOSS" to detect cheating on programming assignments. In Proceedings of the 29th Annual Frontiers in Education Conference: Designing the Future of Science and Engineering Education, San Juan, PR, USA, 10–13 November 1999; Volume 3, pp. 13B3/18–13B3/22. [CrossRef]
41. Ahtiainen, A.; Surakka, S.; Rahikainen, M. Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises. In Proceedings of the 6th Baltic Sea Conference on Computing Education Research Koli Calling, Uppsala, Sweden, 1 February 2006; Volume 276. [CrossRef]

42. Chudá, D.; Kováčová, B. Checking plagiarism in e-learning. In Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies, Sofia, Bulgaria, 17–18 June 2010; Volume 471. [CrossRef]
43. Joy, M.; Luck, M. Plagiarism in programming assignments. *IEEE Trans. Educ.* **1999**, *42*, 129–133. [CrossRef]
44. Ragkhitwetsagul, C.; Krinke, J.; Clark, D. A comparison of code similarity analysers. *Empir. Softw. Eng.* **2018**, *23*, 2464–2519. [CrossRef]
45. Cutts, Q.; Barr, M.; Bikanga Ada, M.; Donaldson, P.; Draper, S.; Parkinson, J.; Singer, J.; Sundin, L. Experience Report: Thinkathon-Countering an "I Got It Working" Mentality with Pencil-And-Paper Exercises. In Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education-ITiCSE'19, Aberdeen, UK, 15–17 July 2019; pp. 203–209. [CrossRef]
46. Yuan, H.; Cao, Y. Hybrid pair programming—A promising alternative to standard pair programming. In Proceedings of the SIGCSE 2019—Proceedings of the 50th ACM Technical Symposium on Computer Science Education, Minneapolis, MN, USA, 27 February–2 March 2019; pp. 1046–1052. [CrossRef]