AlgoSolve: Supporting Subgoal Learning in Algorithmic Problem-Solving with Learnersourced Microtasks

Kabdo Choi
School of Computing,
KAIST
Daejeon, Republic of Korea
cyron1259@gmail.com

Hyungyu Shin
School of Computing,
KAIST
Daejeon, Republic of Korea
hyungyu.sh@kaist.ac.kr

Meng Xia
School of Computing,
KAIST
Daejeon, Republic of Korea
iris.xia@connect.ust.hk

Juho Kim
School of Computing,
KAIST
Daejeon, Republic of Korea
juhokim@kaist.ac.kr

ABSTRACT

Designing solution plans before writing code is critical for successful algorithmic problem-solving. Novices, however, often plan on-the-fly during implementation, resulting in unsuccessful problem-solving due to lack of mental organization of the solution. Research shows that subgoal learning helps learners develop more complete solution plans by enhancing their understanding of the high-level solution structure. However, expert-created materials such as subgoal labels are necessary to provide learning benefits from subgoal learning, which are a scarce resource in self-learning due to limited availability and high cost. We propose a learnersourcing workflow that collects high-quality subgoal labels from learners by helping them improve their label quality. We implemented the workflow into AlgoSolve, a prototype interface that supports subgoal learning for algorithmic problems. A between-subjects study with 63 problem-solving novices revealed that AlgoSolve helped learners create higher-quality labels and more complete solution plans, compared to a baseline method known to be effective in subgoal learning.

CCS CONCEPTS

• **Applied computing** → *Computer-assisted instruction*; *E-learning*; • **Human-centered computing** → Computer supported cooperative work.

KEYWORDS

Algorithmic problem-solving, Learnersourcing, Subgoal learning

ACM Reference Format:
Kabdo Choi, Hyungyu Shin, Meng Xia, and Juho Kim. 2022. AlgoSolve: Supporting Subgoal Learning in Algorithmic Problem-Solving with Learnersourced Microtasks. In *CHI Conference on Human Factors in Computing Systems (CHI '22), April 29-May 5, 2022, New Orleans, LA, USA*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3491102.3501917

CHI '22, April 29-May 5, 2022, New Orleans, LA, USA

ACM ISBN 978-1-4503-9157-3/22/04…$15.00
https://doi.org/10.1145/3491102.3501917

1 INTRODUCTION

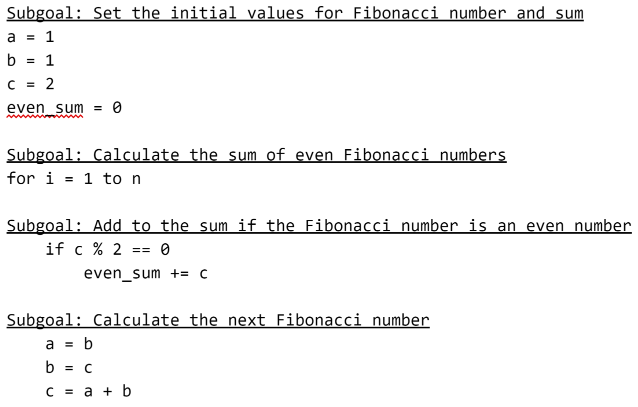
Algorithmic problem-solving [3]—solving programming problems by formulating efficient algorithms that satisfy the time and memory constraints of the problem—teaches learners programming and program design [1]. A number of online websites provide a learning environment for algorithmic problem-solving, most notably Online Judge systems [26, 47] such as TopCoder Competitive Programming [1], Peking University Judge Online [2], and Google Code Jam [3]. Online Judge systems enable self-learning of algorithmic problem-solving skills with their various features, such as automated grading of the solution code where learners can immediately receive results on the correctness of their solution or learner communities where learners can ask questions and discuss solutions with each other.

One characteristic of algorithmic problem-solving is that planning out a solution before coding is crucial for successfully solving the problem [6]. This practice is important since the learner can separate the two key challenges in problem-solving—decomposing the problem into subproblems (decomposition problem) and composing subsolutions into a complete solution (composition problem)—and tackle each stage in isolation [18, 20]. However, novices often develop plans on-the-fly as they write code instead of planning in advance [6, 32, 40], meaning that they are dealing with decomposition and composition—already challenging problems for novices [27]—at once, and end up producing incorrect or inferior solutions [6]. Also, since the solution is not mentally well-constructed in the beginning, their plans get easily altered as they discover new problems in the middle of implementation [42], causing additional frustration.

Prior work has shown that subgoal learning [8]—a method where students learn solutions for problems by decomposing the solution into functional pieces (i.e., subgoals)—helps learners develop more complete solution plans [12]. The key benefit of subgoal learning is that learners can better understand the generalizable, high-level solution structure using subgoal labels—short textual descriptions that explain the purpose of a given subgoal (example shown in Figure 1). The use of subgoal labels can aid learners when creating schemas by letting them focus on the structure of the solution [28] and help them when later using the schemas as a basis for building solution plans [41]. Subgoal learning has been shown to help learners in applying the learnt solution more easily when solving other problems [7, 30]. Margulieux and Catrambone [28] compared various subgoal learning methods and discovered that making learners create their own subgoal labels and guiding the knowledge construction with hints during label creation or expert-created labels

[1] https://www.topcoder.com/thrive/tracks?track=Competitive Programming
[2] http://poj.org/
[3] https://codingcompetitions.withgoogle.com/codejam

as feedback yields the best problem-solving performance improvement. However, such high-quality materials are a scarce resource in self-learning due to the high cost of producing them, and the lack of expert materials deteriorates the learning experience, which greatly limits the applicability of subgoal learning.



```
Subgoal: Set the initial values for Fibonacci number and sum
a = 1
b = 1
c = 2
even_sum = 0

Subgoal: Calculate the sum of even Fibonacci numbers
for i = 1 to n

Subgoal: Add to the sum if the Fibonacci number is an even number
    if c % 2 == 0
        even_sum += c

Subgoal: Calculate the next Fibonacci number
    a = b
    b = c
    c = a + b
```

Figure 1: An example of a subgoal-labeled worked example.

To overcome the limited availability of expert-created labels in self-learning environments, we employ learnersourcing [48], a crowdsourcing approach for collecting various materials while workers contribute as learners and gain learning benefits; in the algorithmic problem-solving context, our approach collects high-quality subgoal labels from learners while helping them improve their problem-solving ability in return. Controlling the quality from the potentially noisy data is an important challenge in learnersourcing. A common approach for obtaining high-quality materials is by asking learners themselves to rate the quality of the collected data in another learning activity [16, 48, 50]. While such activities provide learning benefits to learners as well as make the workflow scalable, the learners' benefit may vary depending on the quality of the presented material [15]. For example, the learner might have an ineffective learning experience if they are given only low-quality materials during the activity.

In this work, we propose a two-stage learnersourcing workflow (Figure 2) for collecting high-quality subgoal labels in the algorithmic problem-solving context. Our workflow design focuses on providing microtasks that guide learners to improve the quality of the labels they produce, thereby collecting high-quality labels as well as helping them improve their problem-solving skills. The first stage is composed of a set of Subgoal Voting tasks, which are multiple-choice questions where learners select the best subgoal label from multiple subgoal label examples. These tasks act as a 'warm-up' stage that introduces and exposes learners to high-quality subgoal labels, thereby nudging them to create high-quality labels [15]. The second stage consists of Subgoal Labeling tasks where learners create subgoal labels. Learners are first prompted to create their own labels, and then presented with several high-quality subgoal label examples where they can fix any errors or mistakes and improve the quality of their initial labels by making comparisons against the provided examples. The final, improved labels are submitted to the system. The collected labels are later shown as high-quality subgoal label examples to other learners. The system needs to provide high-quality labels for learners to recognize good examples and produce high-quality ones themselves [15]. To provide high-quality examples in the microtasks, the system uses a multi-arm bandits algorithm, which was shown to be effective in previous work in learnersourcing [50].

We implemented our learnersourcing workflow into Algosolve, a prototype interface that supports subgoal learning for algorithmic problems, and conducted a between-subjects study with 63 participants who are novices in algorithmic problem-solving and compared AlgoSolve against a baseline interface where participants receive expert-created labels after creating subgoal labels, which is known to be most effective for subgoal learning [28]. Participants who learned through AlgoSolve created significantly higher quality subgoal labels that reflect a more correct and deeper level of understanding of the subgoal compared to the baseline. The high-quality subgoal label examples selected by the system were also shown to be comparable in terms of quality to those created by experts. AlgoSolve also helped learners improve their problem-solving skills; 31% of the AlgoSolve group were able to provide a complete solution plan compared to 6% in the baseline group, showing a deeper level of understanding of the solution technique taught in the activity.

The contributions of this work are as follows:

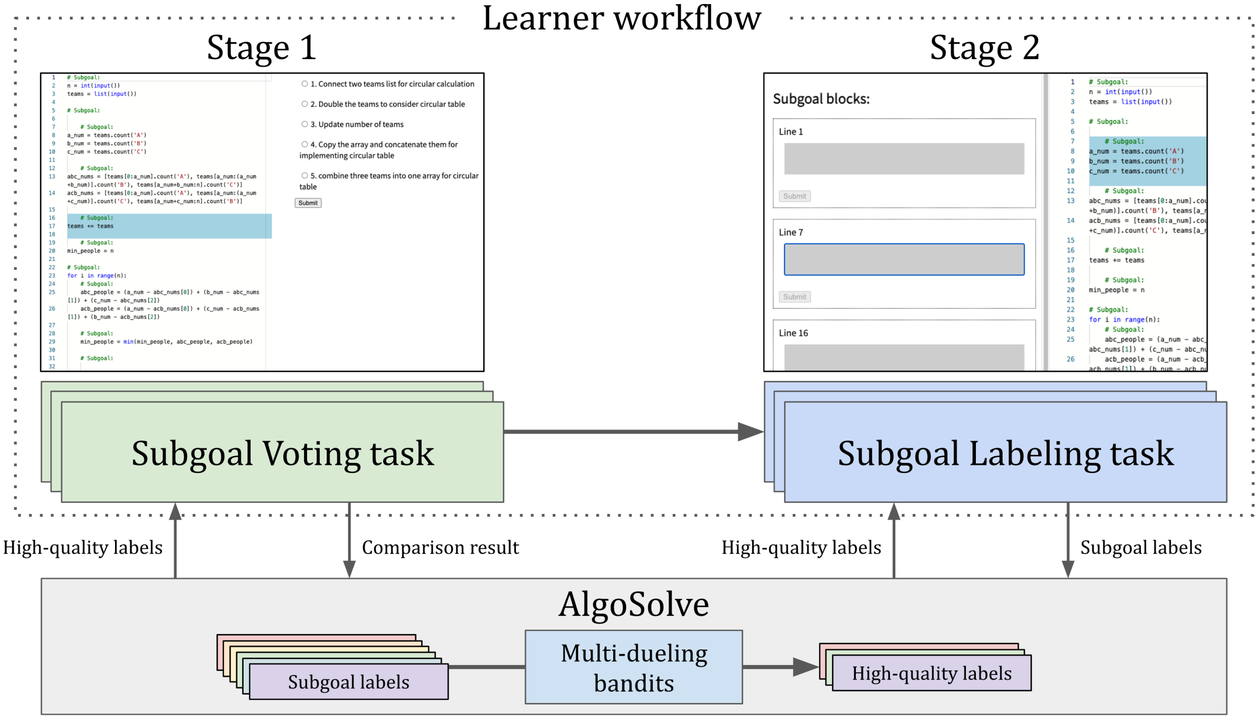
- A learnersourcing workflow that collects high-quality subgoal labels for algorithmic problem-solving by guiding learners to improve the quality of the labels they produce while providing them with learning benefits in solving algorithmic problems.
- Results from a between-subjects study, indicating that AlgoSolve successfully guides learners to provide higher quality subgoal labels while helping them improve their solution planning ability, compared to the best performing method in subgoal learning.

2 RELATED WORK

We review research on subgoal learning and existing learnersourcing approaches.

2.1 Subgoal Learning

The subgoal learning method [8] helps learners in understanding a solution by representing the solution as a hierarchical structure rather than as a flat list of steps. Subgoals—a group of solution steps that form a functional piece of the solution—highlight the purpose of the steps and how they are organized. Subgoal labels encourage learners to focus on the high-level solution structure that is generalizable to other problems instead of memorizing individual solution steps. As learners recognize the generalizable chunks of the problem (or schemas), they can later retrieve the constructed schema when planning out solutions for other problems, in which subgoal labels help schema creation by letting learners focus on the important solution structure [28]. Subgoal learning has been shown to be effective at improving the learners' ability to apply a solution when solving other problems that require transfer [7, 30] or build more complete solution plans that contain all necessary parts for solving a problem [12]. Not only restricted to reading



```
# Subgoal:
n = int(input())
teams = list(input())

# Subgoal:

    # Subgoal:
a_num = teams.count('A')
b_num = teams.count('B')
c_num = teams.count('C')

    # Subgoal:
abc_nums = [teams[0:a_num].count('A'), teams[a_num:(a_num+b_num)].count('B'), teams[a_num+b_num:n].count('C')]
acb_nums = [teams[0:a_num].count('A'), teams[a_num:(a_num+c_num)].count('C'), teams[a_num+c_num:n].count('B')]

    # Subgoal:
teams += teams

    # Subgoal:
min_people = n

# Subgoal:
for i in range(n):
    # Subgoal:
    abc_people = (a_num - abc_nums[0]) + (b_num - abc_nums[1]) + (c_num - abc_nums[2])
    acb_people = (a_num - acb_nums[0]) + (c_num - acb_nums[1]) + (b_num - acb_nums[2])

    # Subgoal:
    min_people = min(min_people, abc_people, acb_people)

    # Subgoal:
```

○ 1. Connect two teams list for circular calculation

○ 2. Double the teams to consider circular table

○ 3. Update number of teams

○ 4. Copy the array and concatenate them for implementing circular table

○ 5. combine three teams into one array for circular table

```
# Subgoal:
n = int(input())
teams = list(input())

# Subgoal:

    # Subgoal:
a_num = teams.count('A')
b_num = teams.count('B')
c_num = teams.count('C')

    # Subgoal:
abc_nums = [teams[0:a_num].cou
+b_num)].count('B'), teams[a_n
acb_nums = [teams[0:a_num].cou
+c_num)].count('C'), teams[a_n

    # Subgoal:
teams += teams

    # Subgoal:
min_people = n

# Subgoal:
for i in range(n):
    # Subgoal:
    abc_people = (a_num - abc_
abc_nums[1]) + (c_num - abc_nu
    acb_people = (a_num - acb_
acb_nums[1]) + (b_num - acb_nu
```

Figure 2: An overview of the proposed learnersourcing workflow. Learners first gain an understanding of good examples of subgoal labels by going through a set of multiple-choice questions comparing various subgoal labels (Subgoal Voting task). Learners then create their own labels and iterate on their initial labels by making comparisons against peer examples (Subgoal Labeling task). A zoomed-in screenshot of the tasks is provided in Figures 4 and 6.

worked examples, but subgoals can also be used during the problem-solving process, such as putting subgoal labels as comments before implementation, acting as building blocks [31]. The effectiveness of subgoal learning at improving the problem-solving performance has been shown across various domains, such as mathematics [2, 7], chemistry [29], and programming [28, 30, 36, 37]. A number of studies in the educational domain have also highlighted the importance of learning how to decompose the problem into smaller subgoals [11, 18, 19, 25], in which subgoal learning can be effectively applied to support such skills.

Initially, subgoal learning was mostly provided as a passive activity where learners read a worked example with pre-written subgoal labels and understand the solution structure [28]. However, passive learning is known to be less effective than other types of learning activities such as active, constructive, and interactive methods [10]. A recent study [28] discovered that letting learners construct their own subgoal labels led to the best problem-solving performance compared to passive or active methods, but only when they received guidance—either hints during creation or expert-created labels as the correct response. However, these types of guidance require the presence of an instructor who can generate such hints or labels, which is not always available in self-learning.

2.2 Learnersourcing Approaches for Collecting Learning Resources

Learnersourcing [24] is a crowdsourcing method where learners engage in learning activities and contribute to building the required materials while going through a meaningful learning experience themselves. Learnersourcing has been successfully employed in collecting various learning materials that are comparable in terms of quality to that of experts. AXIS [50] collects explanations for math problems by asking learners to generate, revise, and evaluate explanations, where machine learning is applied to provide high-quality peer explanations to future learners. PeerWise [14] is a learnersourcing system where students create, answer, and discuss multiple-choice questions. Students were able to write high-quality questions, and even poorly written questions became useful learning resources with other students' comments on the question [13]. UpGrade [46] presents a learnersourcing approach for generating multiple-choice questions using prior student solutions. Compared to traditional open-ended assignments, students who answered UpGrade-generated questions achieved comparable learning outcomes in substantially less time. Other learnersourcing approaches aim to generate hints for improving student solutions [16], subgoal labels for instructional videos [48] or mathematical problems [22], tutorial videos [49], explanations for programming misconceptions [17], or even complex design problems [33].

Researchers have explored learnersourcing workflows that can collect high-quality subgoal labels. Crowdy [48] implements a three-stage workflow for generating high-quality subgoal labels on how-to videos, where the labels are first generated, evaluated, and then proofread. The quality of the final labels was found to be comparable to expert-created labels. However, there were also cases where learners generated subpar quality labels due to a lack of training. In the domain of mathematical problems, SolveDeep [22] uses a learnersourcing workflow that collects subgoal labels and builds a hierarchical representation of subgoals, where learners first generate and then revise their labels based on the feedback given by the system. SolveDeep, however, primarily focuses on building the hierarchy of subgoals, and learners have less freedom in reflecting on their own labels in that they have to either fully accept or reject the given feedback. Also, both approaches ask learners to create subgoal labels without any training on the labeling activity. In contrast to previous work, we design the workflow that guides learners to become familiar with the task and produce better subgoal labels by adding a 'warm-up' stage prior to the creation of the labels.

2.3 Improving the Quality of Learner Contributions in Learnersourcing

The learnersourced materials are often provided to future learners during their learning activity [16, 48, 50]. However, the quality of the learnersourced materials may not be entirely trusted, as they can be of low quality and may even be inaccurate [17], and providing low-quality materials to learners may lead to a worse learning experience. Only a few dedicated learners may exhibit high effort and provide large amounts of high-quality contributions, and when contributions are required (e.g., assessments in a course), the overall quality drops significantly as many learners may be disengaged or aim to achieve the minimum requirements [23].

Although much work has been done on selecting high-quality contributions among the learnersourced data, either through learners themselves [16, 48], automated methods [38], or both [50], aiming to design tasks that improve the learners' ability to generate high-quality materials has gained interest only recently. Khosravi et al. [23] proposed several approaches that could lead learners to provide higher-quality contributions, which include open learner models for aiding self-regulation, tying the tasks with mandatory assessments, and gamification mechanisms. Doroudi et al. [15] suggested that providing high-quality examples to crowd workers can significantly improve the quality of the produced materials. Singh et al. [43] showed that learners who had the option not to create multiple-choice questions but instead only answer existing multiple-choice questions had a better learning experience and created better quality questions, where one of the main factors for not choosing to create questions was that learners lacked confidence in both topic and language. These findings indicate a need for helping learners perceive themselves as being more competent in producing materials, which leads to the design of the warm-up stage in our workflow. Results from an experiment by Nguyen et al. [39] imply that the quality of the crowdsourced output can depend on the task design and cognitive load of the learners. Another study done by Moore et al. [34] revealed that students who showed a better understanding of the material being covered created higher-quality multiple-choice questions [34]. Building upon the findings from previous work, we propose a novel learnersourcing workflow that helps learners familiarize themselves with the task being covered and enhance their understanding of the learning material.

3 LEARNERSOURCING WORKFLOW DESIGN

Learners need to improve their solution planning skills and thereby improve their algorithmic problem-solving performance. We provide learners with a subgoal learning activity as they go through the learnersourcing workflow so that they can have a conceptual understanding of the solution [7] and improve their ability to design solution plans [12].

The design of the activity follows the approach commonly used in subgoal learning research; learners study a worked example that solves a given algorithmic problem, where the code is grouped into subgoals. Learners need to be given high-quality subgoal labels as they create their own labels [28]. We now introduce the design of our learnersourcing workflow that helps learners to achieve these goals.

3.1 Learnersourcing Workflow Design

We designed a two-stage learnersourcing workflow that guides learners to improve the quality of the subgoal labels they produce, and implemented it in a prototype system that supports subgoal learning for algorithmic problems, named AlgoSolve. The workflow consists of two types of microtasks: 1) **Subgoal Voting tasks** where learners read through several subgoal label examples, compare the quality between them, and vote for the best example through system-generated multiple-choice questions, and 2) **Subgoal Labeling tasks** where learners create their own subgoal labels and iterate on their labels through comparisons against peer examples. The Subgoal Voting task is designed as a 'warm-up' stage before creating subgoal labels, helping learners to gain an understanding of good subgoal labels by asking them to compare different examples of subgoal labels. The Subgoal Labeling task helps learners make better quality labels and thereby effectively build the conceptual organization of the solution.

The microtasks are generated from the same worked example that solves a given problem. The worked example is decomposed into subgoals, and each subgoal is randomly assigned to either of the microtasks (Figure 3). The worked example and the subgoals in the worked example are generated prior to the activity, done by domain experts. Determining an appropriate subgoal scope could have also been developed as a learnersourcing task in order to build a fully autonomous system with only learners; however, we did not include in this work to focus on gathering quality subgoal labels.

In order to support learners in learning subgoals using high-quality examples, the system needs to secure enough label examples in the beginning. AlgoSolve gathers initial examples by providing learners with the Subgoal Labeling task alone, without the support from peer examples. After three different examples are accumulated for each subgoal, the system transitions into the full workflow and provides both tasks with peer example support.

We now explain each stage in detail, followed by the policy for selecting peer examples used in AlgoSolve.

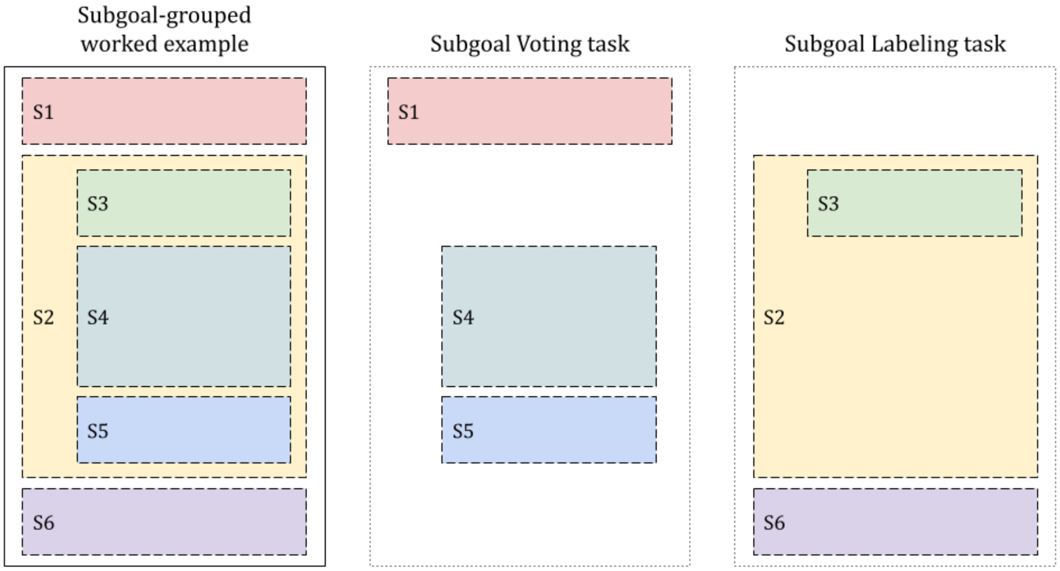
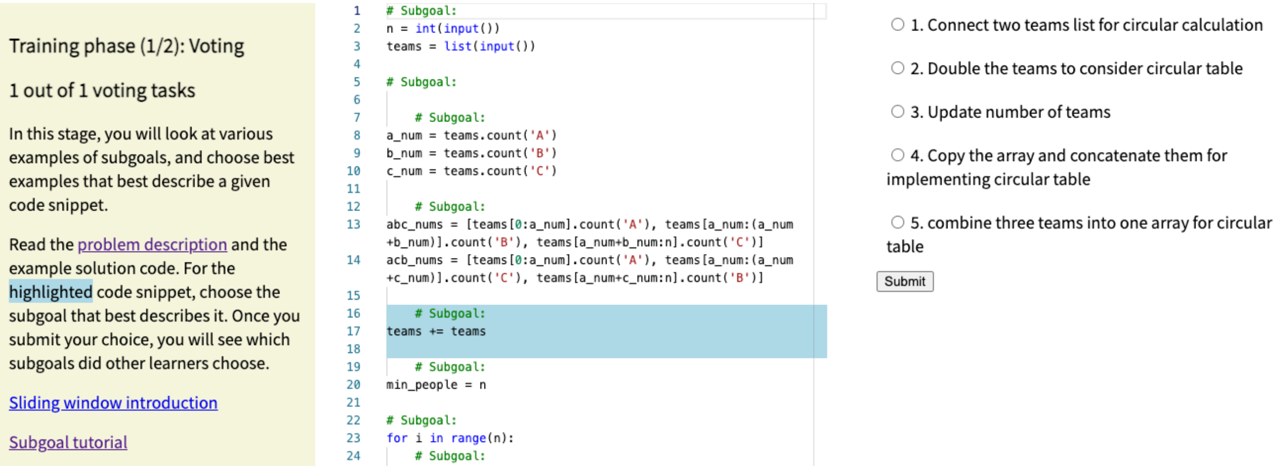


Figure 3: The microtasks are generated from a single worked example, which is grouped in terms of subgoals (S1 - S6) prior to the process. Each subgoal, shown as boxes, is randomly assigned to either the Subgoal Voting task or the Subgoal Labeling task, forming a single microtask.



Training phase (1/2): Voting

1 out of 1 voting tasks

In this stage, you will look at various examples of subgoals, and choose best examples that best describe a given code snippet.

Read the problem description and the example solution code. For the highlighted code snippet, choose the subgoal that best describes it. Once you submit your choice, you will see which subgoals did other learners choose.

Sliding window introduction

Subgoal tutorial

```
# Subgoal:
n = int(input())
teams = list(input())

# Subgoal:

    # Subgoal:
a_num = teams.count('A')
b_num = teams.count('B')
c_num = teams.count('C')

    # Subgoal:
abc_nums = [teams[0:a_num].count('A'), teams[a_num:(a_num
+b_num)].count('B'), teams[a_num+b_num:n].count('C')]
acb_nums = [teams[0:a_num].count('A'), teams[a_num:(a_num
+c_num)].count('C'), teams[a_num+c_num:n].count('B')]

    # Subgoal:
teams += teams

    # Subgoal:
min_people = n

# Subgoal:
for i in range(n):
    # Subgoal:
```

○ 1. Connect two teams list for circular calculation

○ 2. Double the teams to consider circular table

○ 3. Update number of teams

○ 4. Copy the array and concatenate them for implementing circular table

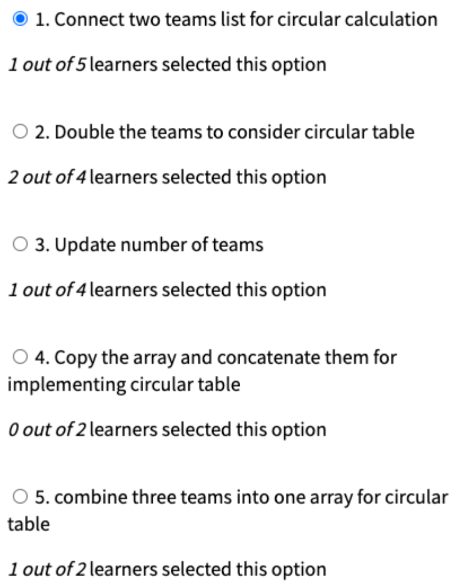
○ 5. combine three teams into one array for circular table

Figure 4: Overview of the Subgoal Voting task. Learners are given up to five subgoal labels and are asked to select a single option that best describes the given subgoal.

3.1.1 *Task 1: Subgoal Voting.* The Subgoal Voting task (Figure 4) is designed as multiple-choice questions that ask learners to select a subgoal label example that best describes a given subgoal with the corresponding code segment. Learners are given up to five examples and select a subgoal label example that best describes a given code snippet. The system uses learners' selections to determine the quality differences between subgoal labels and which labels to show to future learners as examples. By showing subgoal label candidates that explain the given subgoal, we expect learners to quickly form a good understanding of the solution presented in the worked example [25].

The Subgoal Voting stage acts as a 'warm-up' learning activity where learners familiarize themselves with subgoal learning and also good examples of subgoal labels. In contrast to prior work where learners are simply given such examples [15], the Subgoal Voting tasks ask learners to choose the best example among the given options, which is likely to increase learner engagement and gain a better understanding of subgoals [10]. The learners' selections also influence which labels the system determines as high-quality (further explained in Section 3.1.3).

Learners might lack knowledge in the topic being covered or subgoal labels and have difficulty in understanding and determining high-quality subgoal labels. In order to support learners in their learning process, the system provides both system-selected, high-quality subgoal labels and randomly-selected labels that are likely to be of lower quality compared to system-selected labels. Learners receive three system-selected labels and up to two randomly selected ones. We decided to include both types of examples so that learners can make easier contrasts quality-wise by differentiating labels of varying quality. At the same time, the number of randomly selected examples is limited to two in order to avoid taking up too many of the available options since these low-quality options themselves have little benefit to learners [45]. We also decided to make learners choose a single option so that learners have to deeply think over selecting the best option rather than simply checking all plausible-looking ones and avoid the system having unclear distinctions among multiple plausible options.



● 1. Connect two teams list for circular calculation

1 out of 5 learners selected this option

○ 2. Double the teams to consider circular table

2 out of 4 learners selected this option

○ 3. Update number of teams

1 out of 4 learners selected this option

○ 4. Copy the array and concatenate them for implementing circular table

0 out of 2 learners selected this option

○ 5. combine three teams into one array for circular table

1 out of 2 learners selected this option

Figure 5: Peer voting result information after the learner submits their vote in the Subgoal Voting task.

Multiple-choice questions are typically accompanied by feedback on the learners' choice, such as showing the correct answer and additional explanations for the options. However, since the system does not have a clear knowledge of the correctness of the options, it instead provides the peer voting result on each of the examples (Figure 5). The peer voting result is shown as the number of times the example was chosen by previous learners.

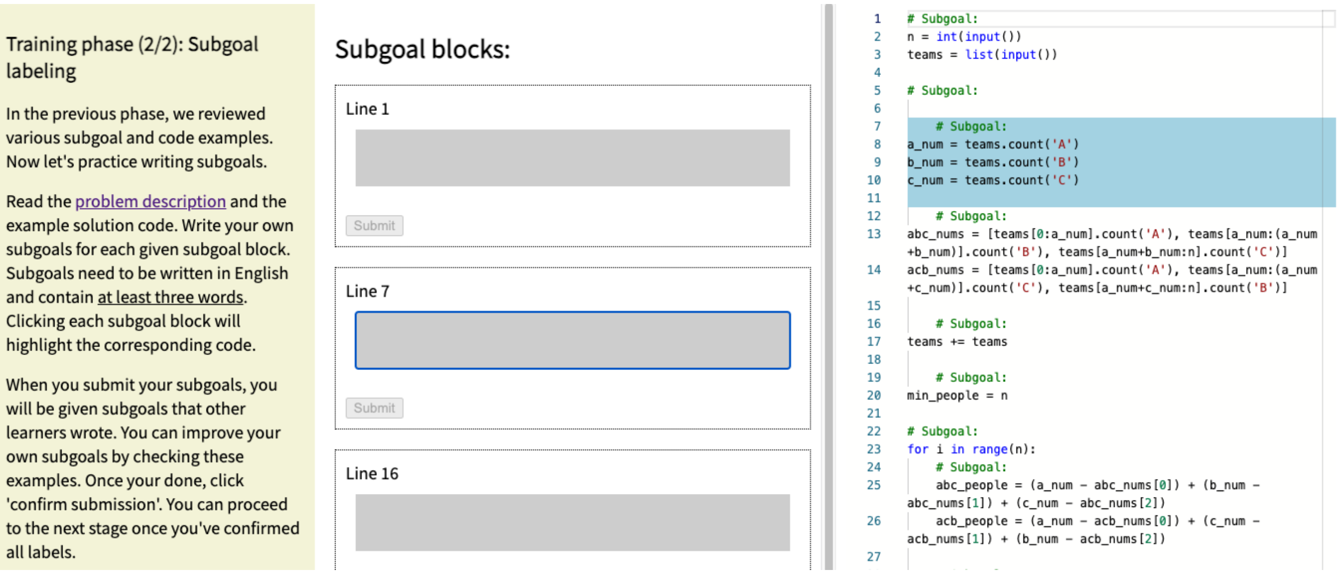
3.1.2 Task 2: Subgoal Labeling. Now that learners gained an understanding of high-quality subgoal labels, the system then asks learners to provide their labels in the Subgoal Labeling task (Figure 6). Learners first submit their initial work (i.e., initial subgoal labels) and then resubmit their final descriptions (i.e., final subgoal labels) after viewing feedback given by the system (Figure 7). The system provides three system-selected subgoal labels as feedback, where learners can make comparisons with their initial labels, revise errors or misconceptions in the initial labels, and make iterations. The final, improved labels are collected by the system and provided to future learners.

The Subgoal Labeling task is grounded on prior work on subgoal learning [28] and resembles the most effective method for guiding learners to successfully build mental organizations of the solution. Margulieux and Catrambone [28] argue that guidance should be carefully thought over; poorly designed guidance, such as providing both hints and expert-created labels as feedback, could hinder learning by dismantling the learner's prior understanding of the solution and making them blindly apply the expert labels. We address this issue by intentionally not presenting the examples as being feedback nor of high quality so that learners become more likely to keep their own explanations reflecting their own understanding of the subgoal, rather than perceiving the given examples as 'correct' answers and simply copying them.

3.1.3 Policy for Selecting Subgoal Label Examples. As subgoal labels get accumulated, AlgoSolve needs to determine high-quality subgoal label examples to show in the microtasks. Many of the existing learnersourcing workflows use majority voting [16, 22, 48] to choose the best examples. Another approach is to employ machine learning to dynamically determine examples shown to learners, such as multi-armed bandits [50, 51], a machine learning approach for extracting the maximum reward from several alternative choices (i.e., arms) where the reward of each arm is previously unknown. We decided to use the latter approach, specifically multi-armed bandits. We chose multi-armed bandits as it is capable of searching for newly added examples (exploration) while also selecting examples known to be of the best quality (exploitation), which gives fair exposure to more recently added examples.

AlgoSolve observes the decisions learners make in choosing the best subgoal label among multiple examples through multiple-choice questions. To fit the multi-armed bandit problem into our context of voting, we formulate this problem as a multi-dueling bandit problem [5]—a variant of the multi-armed bandit problem where the algorithm selects multiple arms, which observes the result of pairwise duels between the selected arms. In particular, we use IndSelfSparring [44], a multi-dueling bandits algorithm that uses Thompson sampling. In order to incentivize newly added subgoal labels in the selection process, we gave the labels a high prior distribution of Beta(4, 1), meaning that the labels are likely to be chosen four out of five times.

Since learners submit their labels after making comparisons against existing labels, there is a possibility that the submitted label is a duplicate of a previously submitted label. Also, for simple labels (e.g., subgoal on printing the result value), there could be identical labels in the pool of label examples. In order to avoid label examples with identical content, we prevent all labels that are identical to a previously submitted label after removing non-alphabetical characters from being selected by the system.



Training phase (2/2): Subgoal labeling

In the previous phase, we reviewed various subgoal and code examples. Now let's practice writing subgoals.

Read the problem description and the example solution code. Write your own subgoals for each given subgoal block. Subgoals need to be written in English and contain at least three words. Clicking each subgoal block will highlight the corresponding code.

When you submit your subgoals, you will be given subgoals that other learners wrote. You can improve your own subgoals by checking these examples. Once your done, click 'confirm submission'. You can proceed to the next stage once you've confirmed all labels.

Subgoal blocks:

Line 1

Submit

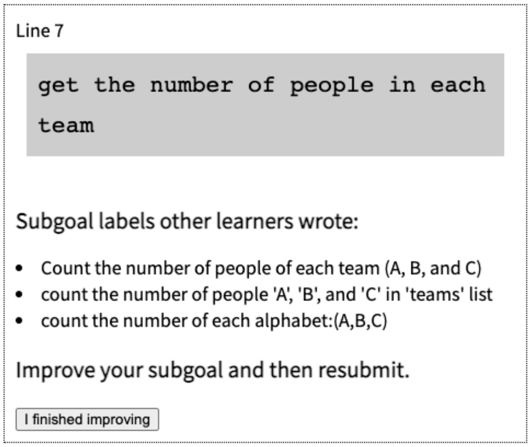
Line 7

Submit

Line 16

```
1   # Subgoal:
2   n = int(input())
3   teams = list(input())
4
5   # Subgoal:
6
7       # Subgoal:
8   a_num = teams.count('A')
9   b_num = teams.count('B')
10  c_num = teams.count('C')
11
12      # Subgoal:
13  abc_nums = [teams[0:a_num].count('A'), teams[a_num:(a_num
    +b_num)].count('B'), teams[a_num+b_num:n].count('C')]
14  acb_nums = [teams[0:a_num].count('A'), teams[a_num:(a_num
    +c_num)].count('C'), teams[a_num+c_num:n].count('B')]
15
16      # Subgoal:
17  teams += teams
18
19      # Subgoal:
20  min_people = n
21
22  # Subgoal:
23  for i in range(n):
24      # Subgoal:
25      abc_people = (a_num - abc_nums[0]) + (b_num -
    abc_nums[1]) + (c_num - abc_nums[2])
26      acb_people = (a_num - acb_nums[0]) + (c_num -
    acb_nums[1]) + (b_num - acb_nums[2])
27
```

Figure 6: Overview of the Subgoal Labeling task. Learners provide their own subgoal labels that describe the given subgoal.



Line 7

get the number of people in each team

Subgoal labels other learners wrote:

- Count the number of people of each team (A, B, and C)
- count the number of people 'A', 'B', and 'C' in 'teams' list
- count the number of each alphabet:(A,B,C)

Improve your subgoal and then resubmit.

I finished improving

Figure 7: After the learner submits their initial labels in the Subgoal Labeling task, system-selected peer examples are provided.

4 EVALUATION

In the current study, we evaluate whether AlgoSolve helps learners improve the quality of the subgoal labels they produce while improving their solution planning ability. We address the following three research questions:

RQ1. (System label quality) Is the designed learnersourcing workflow capable of providing high-quality subgoal label examples to learners?

RQ2. (Learner contribution quality) Does the presence of microtasks help learners create better quality labels, compared to producing labels without the support of microtasks?

RQ3. (Learner experience) How do the learning activities in the microtasks affect the learners’ learning experience?

To answer these questions, we conducted a between-subjects study that compares AlgoSolve (**microtask** condition) to a baseline interface (**baseline** condition). The baseline interface replicates the best performing method in subgoal learning where learners create labels on their own, accompanied by expert-created subgoal labels as feedback [28]. The baseline condition can also be considered as a basic learnersourcing approach where learners create materials without supporting features that help them improve the quality of their contribution. For each participant in the microtask condition, we randomly assigned half of the subgoals to the Subgoal Voting task and the remaining half to the Subgoal Labeling task. Participants in the baseline condition were asked to write subgoal labels for all subgoals.

4.1 Participants

We recruited 68 participants who are novices in algorithmic problem-solving from two universities and a popular online judge system, all located in South Korea. Among the participants, the first five were assigned to create the initial seed subgoal labels for AlgoSolve by using the baseline interface and were excluded from the analysis. The remaining participants were randomly assigned to one condition. In the end, 63 participants (male: 46, female: 17, mean age: 23.4, SD age: 4.16) completed the study session, where 31 used the baseline interface and 32 used AlgoSolve. Participants were compensated with 20,000 KRW (approximately 20 USD) after completing the 1.5-hour study.

We asked participants to report their proficiency in algorithmic problem-solving (baseline: $\mu = 3.22$, microtask: $\mu = 3.31$), their familiarity with the topic covered in the study (baseline: $\mu = 1.51$, microtask: $\mu = 1.75$), and confidence in writing explanations in English on a 7-point Likert scale (baseline: $\mu = 3.62$, microtask: $\mu =$

3.92). We found no significant differences between the conditions for any of the questions. All except six participants had less than a year of experience in algorithmic problem-solving. More than two-thirds of the participants (44 out of 63) were completely unfamiliar with the solution technique used in the current study (i.e., Sliding Window).

4.2 Study Materials

The Sliding Window technique [9] was selected as the topic for the current study. Sliding Window is a solution technique commonly used for reducing the time complexity when solving array problems that deal with subarrays (e.g., find the subarray that has the largest sum), which is done by sliding a subarray (i.e., window) throughout the entire array and updating the necessary values efficiently as the window gets slid. We chose Sliding Window since it is a relatively straightforward technique that requires little background knowledge in algorithms, and therefore learner performance would be less affected by their previous experience in algorithmic problem-solving. We selected two algorithmic problems that are solvable using the technique. We sampled the problems in terms of difficulty so that they do not have too trivial solutions, thereby presenting enough challenge to novices and having similar levels of difficulty. The level of difficulty was extracted from an expert-sourced repository that collects the difficulty rating of algorithmic problems [4].

The subgoal-grouped worked example and expert subgoal labels were created through discussion between the experimenters and an expert who had significant experience in problem-solving competitions. The created worked example consisted of 14 subgoals. The worked example with expert subgoal labels is included in the supplementary material.

4.3 Procedure

An overview of the study procedure is shown in Table 1. Participants first completed the pre-questionnaire where they reported their proficiency and expertise in algorithmic problem-solving. Before starting the main session, participants were given instructional materials on the Sliding Window technique and subgoal labels. In the main study session, learners received either the baseline labeling task or microtasks depending on the condition they were assigned. We asked participants to complete a post-questionnaire where they reported the cognitive load of the activity and their experience in using the system. The study session concluded with an assessment task where participants build and submit a solution plan for a similar problem which also uses the Sliding Window technique.

Time (mins)	Baseline	Microtask
Pre-session (20)		
5	Introduction & Consent	
5	Pre-questionnaire	
5	Sliding Window technique introduction	
5	Subgoal label tutorial	
Main study session (30)		
30	Baseline labeling task	Subgoal Voting task
		Subgoal Labeling task
Post-session (25)		
5	Post-questionnaire	
20	Assessment task	

Table 1: Overview of the study procedure.

4.4 Measurements

We asked learners about their experience in using the system. In the post-questionnaire, learners were asked to rate their cognitive load, total time spent for the main session, helpfulness of the provided subgoal labels, and helpfulness of the learning activity given in the main session. For measuring cognitive load, we used the CS Cognitive Load Component Survey (CS CLCS) [35], a cognitive load measurement specialized for the context of computing education research. CS CLCS is composed of 10 questions that measure three aspects of cognitive load: intrinsic, extraneous, and germane load. The degree of helpfulness was rated using a 7-point Likert scale (1: not helpful at all, 7: very helpful). While germane load is now considered as being redundant [21], we use CS CLCS as it is the only measure in computing education research and has been shown to be reliable in measuring cognitive load [52].

One of the benefits of using subgoal labels is that it reflects the purpose of the given subgoal instead of surface-level features. To investigate how microtasks affect learners in creating quality subgoal labels that successfully explain their purpose, we developed a subgoal label quality score that consists of four categories (Table 2). The labels were first categorized based on whether the label contained any incorrect explanations or did not summarize the code in plain language (L0). Among the correct labels, we categorized them in terms of explanation depth (L1, L2, L3). We chose explanation depth since it reflects how well the learner conceptually understands the solution, which is the primary goal of subgoal learning. Representative examples for each level are shown in Table 4.

Category (Level)	Description
Improper (L0)	Labels that convey wrong information about the subgoal, do not summarize the function but instead explain line-by-line, or do not explain in plain language (e.g., code)
Surface-level (L1)	Labels that provide no explanation other than the surface- or code-level interpretation of the code (e.g., directly mentioning a variable with its name)
Intermediate-level (L2)	Labels that contain more than mere surface-level explanations but part of the explanation lacks depth.
Deep-level (L3)	Labels that successfully explain the purpose of the given code.

Table 2: Categories of subgoal label quality that were used in the current study.

We recruited four experts as raters to measure the quality of learner-created subgoal label examples selected by AlgoSolve. The

[4] https://solved.ac/

SOLO category	Description
1 - Prestructural	Nonsensical answer or an answer that had no more information than the question provided.
2 - Unistructural	Described 1-2 concepts that applied to the problem, but the description was incomplete or did not demonstrate based on the Sliding Window technique.
3 - Multistructural	Described all concepts needed to solve the problem but provided no explanation beyond the question at hand.
4 - Relational	Described how to solve the problem and explained how the different pieces of the choices were made to solve this particular problem.
5 - Extended Abstract	Explained how to solve a problem like this in abstract terms.

Table 3: SOLO scoring criteria that were used in the current study.

raters had significant experience in solving algorithmic problems and/or teaching algorithmic problem-solving. We asked raters to solve the training problem using the Sliding Window technique prior to the evaluation to ensure that they are knowledgeable about the solution technique. They were also shown the tutorial material on subgoal labels identical to participants so that they have a common understanding of what constitutes good subgoal labels. The evaluation was made by comparing a system-selected subgoal label against the expert-created subgoal label for each of the 14 subgoals, rating the labels as either *system better*, *expert better*, or *matching* in terms of quality. The interrater reliability, measured with Fleiss' kappa, showed a slight agreement of 0.24.

To assess whether AlgoSolve provides learning benefits in algorithmic problem-solving to learners, we measured their solution planning ability using the solution plans submitted in the assessment task. The submitted solution plans are graded using the SOLO taxonomy [4], following previous work [12]. The SOLO taxonomy measures how well learners understand the topic and whether they produce higher quality, structurally more complex learning output; in our study, how well learners describe their solution plan using the conceptual understanding gained from the subgoal learning activity. The SOLO taxonomy consists of five categories: Prestructural, Unistructural, Multistructural, Relational, and Extended Abstract. The criteria we used in the current study, which are a variant of prior research on assessing the effect of subgoal learning in solution planning [12], is provided in Table 3. An example solution plan for each category is provided in the supplementary material.

For coding the submitted subgoal labels and solution plans based on the categories in Tables 2 and 3, we used the following method, which is similar to prior work [12]. Three raters graded the first five sets of submissions (for each subgoal for labels) together to form a consensus on expectations for each category. The raters then individually graded another five sets of submissions and then discussed and resolved any differences. After repeating the process for 20% of the submissions, we measured the inter-rater reliability of the individually graded submissions. The intraclass correlation coefficient with absolute agreement (ICC(A)) was used for measuring inter-rater reliability, following prior work [12, 28]. The raters achieved a high level of agreement for both types of submissions; 0.90 for subgoal labels and 0.97 for solution plans. The remaining 80% of the submissions were graded by a single rater.

5 RESULTS

A total of 658 labels were collected after the study (baseline: 434, microtask: 224). Note that the microtask participants were given the Subgoal Labeling task for only half of the total subgoals and therefore resulted in around half the number of labels submitted by the baseline participants. We summarize the results in terms of the three research questions.

5.1 RQ1. Quality of the System-selected Labels

The comparative evaluation between system-selected and expert-created labels (Table 5) revealed that raters equally preferred the system-selected labels (S6, S7, S8) and the expert-created labels (S2, S4, S10). Three labels (S1, S5, S14) were considered as being of comparable quality. The remaining five labels had no majority opinions (i.e., the same number of raters diverged into different opinions), two of them being a tie between 'matching' and system label, implying a weak preference to the system-selected label. This result indicates that, even from a small population of roughly 30 learners, our system is *able to determine and provide subgoal labels of decent quality that are comparable with expert-created labels*.

In general, raters preferred expert-created labels for better describing the high-level and abstract purpose of the subgoal in question. For example in S10, the phrase "slide the window" in the expert-created label is a key term for explaining its high-level purpose. Meanwhile, raters noted that the system-selected labels would be better as these provide more details for understanding the given code, such as "in the worst case as n" (S6) or "for two patterns of ordering" (S8). This indicates that learners, especially novices, may have different expectations on subgoal labels from instructors or experts; subgoal labels are believed to be of higher quality if the labels are written in high-level and abstract terms [28], while novices show a higher demand for more in-depth explanations. Lastly, raters indicated that the meaning of some system-selected labels (e.g., S13) could be misinterpreted by learners, which implies the need for validation or proofreading tasks of the collected labels in the learnersourcing workflow.

5.2 RQ2. Quality of the Learner Submissions

Overall, the microtasks in AlgoSolve were able to help learners create better quality labels compared to the baseline task (Figure 8). Among the initial labels created in the microtask condition, 22% of them were classified as being improper (i.e., L0), which was reduced to 15% in their final submissions, while 30% of the labels baseline participants produced were labeled as L0. Meanwhile, microtask participants were able to create L3 labels (i.e., showed a deep understanding of the purpose of the given subgoal) for 27% of the total labels in their first attempt, later improved to 43%, while only 20% of the submitted labels in the baseline condition was identified as L3. The degree of difference varied between subgoals though; for

S#	L0	L1	L2	L3
S2	settings for the overall team	Declare and initialize variables.	Preprocess for calculating output	Initial setting for Sliding Window
S4	Find the wrong input	Save operation result in abc_nums, acb_nums	Calculate number of people sitting in acb or abc	Count number of player who don't have to move.
S5	Update number of teams	Add the list to the back one more time.	duplicate team list	Connect two teams list for circular calculation
S7	step for reorganization	compare n times and get real min_people	Get minimal number of people that must switch seats.	Get the minimum number of people to move by sliding window method
S10	check abc people that is lowest	update abc_nums and acb_nums for next loop	update num list whether A, B, C or properly located	After moving sliding window, update the number of people who seat in place

Table 4: Representative examples of subgoal labels for each label score. Each row represents a subgoal (e.g., S1: Subgoal 1).

S#	System-selected label	Expert-created label	Majority
S1	Get the input	Get the input values	Matching
S2	Make the setup for sliding window	Set up the initial values for sliding window	Expert
S3	Count the number of people of each team (A, B, and C)	Calculate the number of people in each team	No maj. (M, S)
S4	Count how many A, B, C are in ABC or ACB order	For each possible team formation, calculate the number of people who are correctly seated	Expert
S5	Double the list of team to find possible cases for circular table	Since both ends of the seat are connected, duplicate the sitting status list	Matching
S6	Set the minimum number of people need to move in the worst case as 'n'	Set up the initial minimum value holder	System
S7	Find minimum people to be in ABC or ACB group order using siding window method.	Slide the window and update the minimum value holder	System
S8	Get the number of people who should change their seats for two patterns of ordering	Calculate the number of people who have to move seats to fit the team formation	System
S9	Update the number of minimum changes	Update the minimum value holder	No maj. (M, S)
S10	Update the number of people in ABC or ACB order	Slide the window to the next index and update the number of correctly seated people	Expert
S11	Update the "correct" sitting number of people based on the starting position (we want it to be A)	Handle the person who is moved out from the first team	No maj. (E, S)
S12	Update abc_nums and acb_nums for the change of starting point of the second region.	Handle the person who is moved out from the second team	No maj. (E, S)
S13	Update the number of people who should change their seats in the third group	Handle the person who is moved out from the third team	No maj. (E, S)
S14	Print the minimal number of people who need to move	Print the minimal number of people who need to move seats	Matching

Table 5: System-selected labels with the highest mean score among labels that received three votes or better (except for S6). We also provide the comparison results between system-selected labels and expert-created labels. For subgoals without a majority, we also denote the top choices (E: expert, M: match, S: system).

S5, there was a substantial increase in the number of L3 labels in the final labels submitted by microtask participants, while there was no difference in S10.

We performed a Cochran-Armitage trend test to test the difference in the label quality score distribution between baseline labels (B) and initial labels in the microtask condition (Mi), and between initial and final labels in the microtask condition (Mf), using Cliff's delta (d) for calculating effect size. The results showed statistical significance for the difference in the distributions with small effect size—B vs. Mi: $p < 0.004$ ($d = 0.13$), Mi vs. Mf: $p < 0.002$ ($d = 0.17$). In the microtask condition, 163 out of 224 labels were initially considered incomplete (i.e., L0 - L2). Among them, 34% were improved in the final submission (Table 6) after the comparison in the Subgoal Labeling task. Only seven labels, and none of the labels initially counted as L3 decreased in label quality. Overall, the results demonstrate that *microtasks encouraged participants to create more correct, more complete, and better quality labels.*

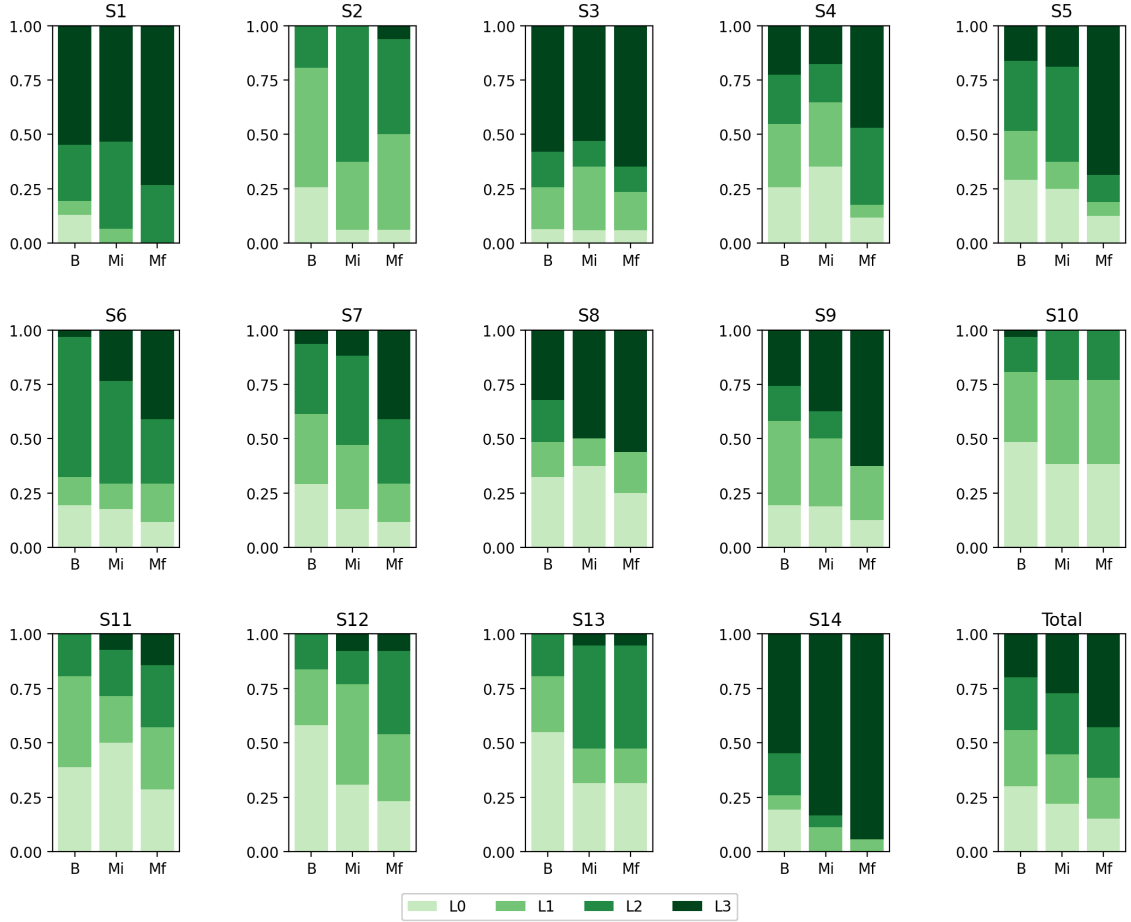


Figure 8: Distributions of label quality scores for each subgoal (S1 - S14) and the aggregated result (B: baseline, Mi: microtask - initial submission, Mf: microtask - final submission).

		Final labels			
		L0	L1	L2	L3
Initial labels	L0	29	6	6	8
	L1	2	33	8	7
	L2	3	2	38	20
	L3	0	0	0	61

Table 6: Changes made in the labels during the Subgoal Labeling task in terms of label quality score. Rows represent scores of initial labels in the Subgoal Labeling task (Mi), while columns represent scores of final labels (Mf).

In addition to the comparison based on label quality score, we provide a brief analysis of the changes made in the final labels.

5.2.1 Notable Pattern of Changes After Comparison. For more than half of the total initial labels (126 out of 224), participants made changes after comparing against peer examples. The degree of changes that were made varied greatly, from minor adjustments (e.g., changing a single word) to modifying the entire content into a different meaning. In particular, participants paid attention to the purpose of the subgoal (e.g., 'considering the circular formation' in S5) or the role of a variable (e.g., 'correctly seated people' in S4), which are critical aspects of subgoal labels but was not included in the initial submission. Even for minor adjustments, participants were able to include key terms that are critical in explaining the purpose of the given subgoal (e.g., 'initialization' in S1, 'update' in S9).

5.2.2 Labels that Decreased in Terms of Label Quality Score. Although participants were able to improve their labels after comparing with peer examples, there were a few cases where the label became worse in terms of label quality score. Among the seven labels that were worsened after comparison, four labels were identified as misinterpreting the meaning of a particular variable, e.g., abc_nums as 'incorrectly' (instead of correctly) seated people. We suspect that this misinterpretation is due to the peer examples containing ambiguous or incorrect information, which is in line with the raters' analysis of system-selected labels compared to expert-created labels.

5.2.3 Copying peer examples. A possible downside of providing peer examples is that learners can blindly follow the given examples by directly copying them. We inspected cases where participants created a label that was identical to a provided label. Even though we did not put any proactive methods or guidelines that prevent plagiarising, only 13 labels (6%) were found to be the same as a previously submitted label, four of which were submitted by a single participant.

5.3 RQ3. Learning Experiences Provided by the Microtasks

5.3.1 Effects on Solution Planning Ability. Participants who learned using the microtask interface were able to come up with better solution plans; 31% of the participants received a score of three or higher, implying that they were able to correctly apply the technique to novel problems, in contrast to only 6% for the baseline condition. The distribution of SOLO scores on participants' solution plans is shown in Table 7. The Cochran-Armitage trend test revealed statistically significant differences between the conditions ($p < 0.05$, $d = 0.26$). This difference implies that microtasks can *help learners apply the learned solution technique when planning out solutions for similar problems.*

	1	2	3	4	5
Baseline	25 (81%)	4 (13%)	1 (3%)	1 (3%)	0 (0%)
Microtask	18 (56%)	4 (13%)	9 (28%)	1 (3%)	0 (0%)

Table 7: SOLO score frequency of the solution plans submitted in the assessment task.

In general, learners in both conditions showed low performance in generating solution plans. The majority of the participants in both conditions received the lowest score (i.e., Prestructural), while none received the highest score possible (i.e., Extended Abstract). The current study was done through a single session rather than a semester-long instruction as in prior work [12], and learners were novices who were also new to the Sliding Window technique, so the overall low performance is not unexpected. It should also be noted that not every solution plan was completely nonsensical but also included inefficient solutions that did not make use of the solution technique they have learned, i.e., they were able to derive a plausible solution plan but were unable to apply the given technique.

5.3.2 Cognitive Load and Helpfulness of the Microtasks. A Kruskal-Wallis H test revealed no significant difference in each cognitive load measure (Table 8) between the conditions—Intrinsic: $p = 0.26$ ($d = 0.17$), Extraneous: $p = 0.94$ ($d = 0.01$), Germane: $p = 0.54$ ($d = 0.05$). Time spent on the training session was also similar between the two conditions (baseline: 28.87 minutes, microtask: 29.63 minutes, $p = 0.81$, $d = 0.03$), indicating that the use of microtasks did not impact the amount of training needed during the session.

	Intrinsic	Extraneous	Germane
Baseline	5.52 (2.64)	2.84 (2.28)	6.50 (2.11)
Microtask	4.97 (2.12)	2.64 (1.78)	6.57 (1.65)

Table 8: Mean (standard deviation) score of cognitive load.

The peer-created labels were shown to be comparable in terms of helpfulness against expert-created ones (Figure 9), with no significant difference between the two groups ($p = 0.67$, $d = 0.11$). Participants also found the learning activities given as microtasks helpful (Figure 10). Overall, the provided activities *helped them gain a better understanding of subgoals and the given solution.* We describe the participants' experiences for each of the microtasks in detail below. However, we also discovered a sharp contrast between Voting and Labeling tasks in terms of helpfulness; the Cochran-Armitage trend test reveals significant differences against the baseline Labeling task ($p < 0.02$, $d = 0.39$) and the Subgoal Labeling task ($p < 0.02$, $d = 0.39$).

Learner Experience on the Subgoal Voting Task. Although the majority of participants (19 out of 32) thought the Subgoal Voting task as being helpful in learning subgoals, participants rated lower in terms of helpfulness compared to the other tasks. Participants reported that the subgoal label examples acted as hints for grasping the meaning of code segments they did not understand well, showing that the Subgoal Voting task acted as a scaffold for understanding the solution when novice learners struggled to comprehend the given worked example. The examples were also helpful for participants in learning good examples of subgoal labels.

Participants found the task unhelpful when the provided options did not enhance but instead distracted from their understanding of the solution. When multiple options had the same meaning with only subtle or subjective differences, participants were confused about which option they should choose. One participant also reported that choosing among the options that have ambiguous meaning was difficult and wished to see more descriptive options. Also, the peer voting result was generally perceived as unhelpful, mainly when the example was recently added and there was not enough peer information. Since there is little peer information about recently added labels (thereby shown as '0 out of 0'), learners were confused by its meaning and did not find any use of the information. Nevertheless, participants still found the task useful in that they had to think hard to choose the best option, and showing various options aided them in understanding the subgoals.

Learner Experience on the Subgoal Labeling Task. Contrary to the Subgoal Voting task, participants were mostly positive about the Subgoal Labeling task. All except five participants reported that they found the task helpful for learning subgoals. Participants found the peer examples useful as a guide for making good labels or improving on their inferior initial labels. One participant noted that he was able to improve code-level terms (e.g., list, for-loop) into more abstract terms (e.g., seating sequence, repetitively) by looking at the peer examples. Peer examples also acted as feedback, enabling learners to identify and fix their errors or misunderstandings about the code segment.

Participants also expressed desires for controlling how and what examples they receive. One participant commented that she would like to get more subgoals that highlight the drawbacks of her subgoal label. Another participant wished that these are presented more like a hint so that she could further develop her thinking skills, where a single example is presented at a time, starting from

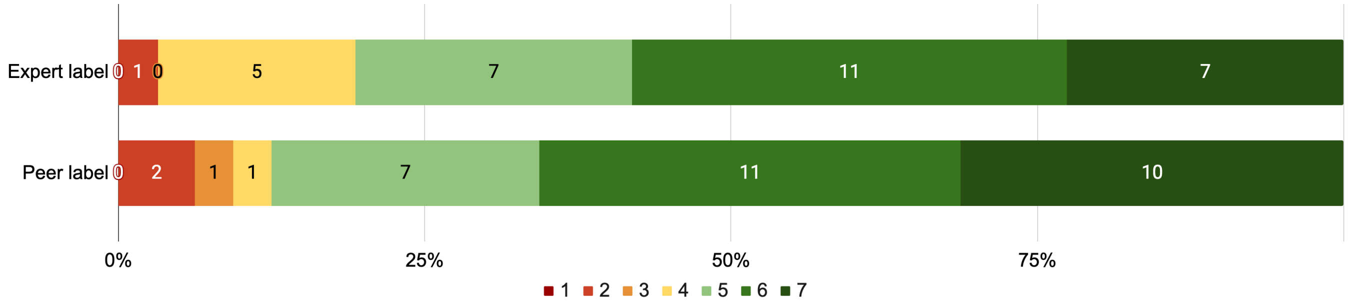


Figure 9: Helpfulness of the given labels; expert labels in baseline, peer examples in microtask (1: not helpful at all, 7: very helpful).

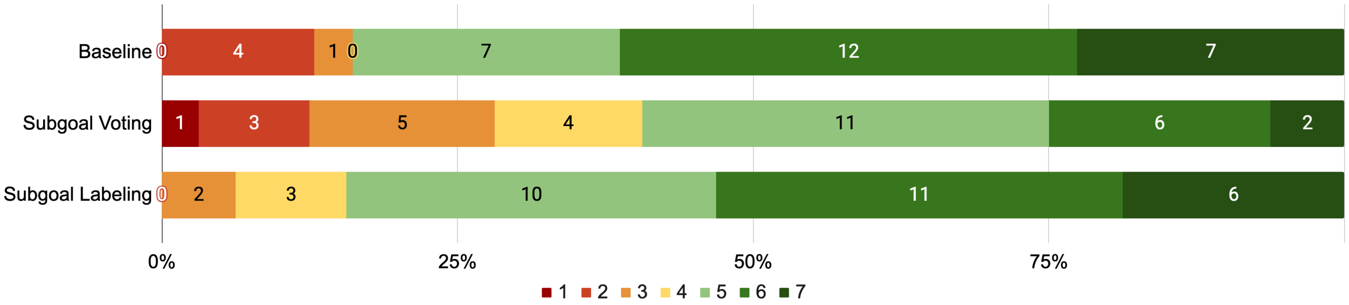


Figure 10: Helpfulness of the learning activities (1: not helpful at all, 7: very helpful).

short labels and later showing more detailed, lengthier explanations.

6 DISCUSSION, LIMITATIONS, AND FUTURE WORK

Our learnersourcing workflow was able to collect high-quality subgoal labels while helping learners gain a better understanding of the solution technique, thereby helping them improve their ability to plan the solution using the learned solution technique. In this section, we discuss the implications, limitations, and future directions of our work.

6.1 Improving the Workflow Design

6.1.1 Improving Quality of Subgoal Labels. Overall, our learnersourcing workflow helped learners produce better quality labels compared to the baseline which did not provide any support when creating the labels. However, the quality of labels that participants created was different across subgoals (Figure 8). We briefly discuss two notable patterns.

Three subgoals (S2, S7, S10) were identified as compound subgoals — high-level subgoals consisting of lower-level subgoals. Compound subgoals were expected to be most challenging to novices as these require an understanding of larger blocks of code. Overall, participants struggled to produce high-quality subgoal labels, but in S7, they were able to improve their labels after comparing against peer examples in the Subgoal Labeling task. We discovered that the voting pattern in the Subgoal Voting task differed between S7 and the other subgoals; the majority of the participants were able to select L3 labels that reflect a deep understanding of the subgoal. We suspect that participants being able to successfully identify high-quality labels in the Subgoal Voting task led to the system successfully showing such labels in the microtasks, resulting in making higher quality labels after comparison. It suggests the importance of successful voting of good explanations in propagating its quality in future learner contributions.

In S4 and S5, we observed a significant improvement after comparison in the Subgoal Labeling task, while participants initially produced labels that were similar to that of baseline participants in terms of quality. The labels received low scores because participants failed to catch the intent in the code. For example, the concatenation operation in S5 is to consider the circular formation, but learners did not explain the purpose of the operation and simply mentioned the operation itself. However, we believe that learners were able to realize their surface-level descriptions after they were given peer examples. This outcome implies that peer comparison took a key role in creating higher-quality subgoal labels.

Even though most of the participants were able to improve their labels after comparing against peer examples, there were cases where peer examples can be misinterpreted and spread wrong information to future labels. These results suggest that a sufficient validation process of the collected labels (e.g., proofreading) should precede before being shown in the Subgoal Labeling task.

6.1.2 Improving Learning Experience. Participants noted that the microtasks helped them in learning subgoals and the given solution technique. However, we observed that the Subgoal Voting task was perceived to be less helpful than the Subgoal Labeling task.

In the Subgoal Voting task, participants were asked to select only a single option. This decision was intended to make the system quickly identify high-quality subgoal label examples, while learners need to make careful decisions on their choices. However, this also induced unwanted confusion and difficulty for learners. When multiple options were virtually identical, learners were confused about which to choose. This shows an important tension in designing learnersourcing systems: The system wants a clear distinction of high-quality labels, while learners might be hesitant to select the best option. The Subgoal Voting task could be improved by providing semantically diverse examples to learners. For example, the system could cluster similarly phrased labels and select one that will be shown to learners from each cluster.

The peer voting result, given as a method of feedback for the multiple-choice questions in the Subgoal Voting tasks, was generally not useful. Learners felt the information was difficult to interpret, which was especially confusing in the early stage of the labeling, where there weren't enough peer votes. Although a longer-term study with more accumulated peer data might provide better insight to learners, the findings from the current study indicate that presentation of such information should be carefully considered so that learners can easily interpret its meaning.

Through open-ended responses, we were able to discover directions that could further improve the learning experience. Currently, the system does not take the learner's initial label into consideration when choosing subgoal label examples. However, as one participant noted, the system could provide examples that are more relevant and better in terms of quality so that learners can make better comparisons. A similar implication is suggested in previous work [28]; learners could have a negative learning experience even though being provided with high-quality subgoal labels due to the sheer difference between their own labels and the expert-created ones, which makes comparison highly difficult. Future work could explore how providing relevant examples for comparison would have an effect on the learning outcome.

Currently, subgoals are randomly assigned to either of the microtasks. However, the amount of benefit from the microtask might vary among learners, based on their prior experience; a competent learner might already have a good understanding of a certain subgoal and therefore feel the Subgoal Voting task as trivial and even unnecessary. Prior work [31] indicates that subgoal learning can quickly become tedious as learners gain expertise. Tasks that match and make use of their high level of understanding could be provided to mitigate this situation, such as asking them to critique the labels provided by other learners and improve their quality.

6.2 Limitations

We evaluated our system with 63 participants. However, this could be considered relatively small-scale compared to previous work on learnersourcing [48, 50] and computer science education [12, 36]. Also, participants did not use English as their primary language, which might have affected their ability to create subgoal labels and solution plans in English. Although our evaluation results strongly suggest the effectiveness of our workflow design, further investigation on a larger and more diverse population would strengthen the usefulness of the approach, most notably the peer voting result shown in the Subgoal Voting task.

While we compared the system-selected subgoal labels against expert-created labels through expert evaluation, we did not collect learners' preferences among the provided peer examples in the current study. Investigating which type of subgoal labels are helpful from the learner's perspective would reveal useful insights in improving the label example selection process, which we leave as future work.

We observed that the majority of the participants were unable to develop a complete solution plan using the solution technique they learned from the activity. Participants might have suffered from the limitation of being a single-session study with only one worked example. Longer-term instructional support with multiple algorithmic problems for a solution technique would further reveal how the microtasks take effect in improving the learners' problem-solving skills. Also, we chose a simple solution technique due to the limitations of a single-session, short-term study. Whether AlgoSolve will be effective for other topics in algorithms that require expertise remains an open question. Since many of the subgoals can be shared across different topics, e.g., input/initialization or output [42], we expect our approach to be applicable in other problems.

6.3 Extending the System into an Autonomous Learnersourcing Platform

Our two-staged workflow was able to successfully distill high-quality subgoal label examples from the collected labels through the Subgoal Voting task without expert intervention. Still, the scope and hierarchy of subgoals were determined through expert discussion prior to the study. We predetermined the subgoal scope and hierarchy in the current study in order to focus on whether the proposed workflow can successfully collect high-quality subgoal labels from learners. The scalability of the system could be improved by lessening the amount of manual work needed from experts. Future work could explore the design of the task that can effectively build such a structure of subgoals.

7 CONCLUSION

Subgoal learning can help learners improve their solution planning ability when solving algorithmic problems. Subgoal learning benefits from expert-created subgoal labels, which are scarce resources in self-learning. We introduced a learnersourcing approach for collecting high-quality subgoal labels by guiding learners to produce higher quality subgoal labels through high-quality peer-created examples while helping them improve their problem-solving skills. We designed a two-stage workflow where learners are first introduced to high-quality subgoal label examples through multiple-choice questions and then create their own labels with an opportunity of iterating on their subgoal labels through comparison against other subgoal labels. Our evaluation results demonstrated the usefulness of the proposed workflow design in helping learners improve the quality of their labels while improving their problem-solving skills.

ACKNOWLEDGMENTS

We thank members of KIXLAB (KAIST Interaction Lab) for their support and feedback. This work was supported by Algorithm LABS.

We thank our study participants for their valuable and insightful comments.

REFERENCES

[1] Owen L Astrachan. 2004. Non-competitive programming contest problems as the basis for just-in-time teaching. In *34th Annual Frontiers in Education, 2004. FIE 2004.* IEEE, T3H–20.

[2] Robert K Atkinson, Richard Catrambone, and Mary Margaret Merrill. 2003. Aiding Transfer in Statistics: Examining the Use of Conceptually Oriented Equations and Elaborations During Subgoal Learning. *Journal of Educational Psychology* 95, 4 (2003), 762.

[3] Roland Backhouse. 2011. *Algorithmic problem solving.* John Wiley & Sons.

[4] John B Biggs and Kevin F Collis. 2014. *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome).* Academic Press.

[5] Brian Brost, Yevgeny Seldin, Ingemar J Cox, and Christina Lioma. 2016. Multi-dueling bandits and their application to online ranker evaluation. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management.* 2161–2166.

[6] Francisco Enrique Vicente Castro and Kathi Fisler. 2020. Qualitative Analyses of Movements Between Task-level and Code-level Thinking of Novice Programmers. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education.* 487–493.

[7] Richard Catrambone. 1998. The subgoal learning model: Creating better examples so that students can solve novel problems. *Journal of experimental psychology: General* 127, 4 (1998), 355.

[8] Richard Catrambone and Keith J Holyoak. 1990. Learning subgoals and methods for solving probability problems. *Memory & Cognition* 18, 6 (1990), 593–603.

[9] Christian Charras and Thierry Lecroq. 2004. *Handbook of exact string matching algorithms.* King's College Publications.

[10] Michelene TH Chi. 2009. Active-constructive-interactive: A conceptual framework for differentiating learning activities. *Topics in cognitive science* 1, 1 (2009), 73–105.

[11] Michael De Raadt, Richard Watson, and Mark Toleman. 2009. Teaching and assessing programming strategies explicitly. In *Proceedings of the 11th Australasian Computing Education Conference (ACE 2009)*, Vol. 95. Australian Computer Society Inc., 45–54.

[12] Adrienne Decker, Lauren E Margulieux, and Briana B Morrison. 2019. Using the SOLO taxonomy to understand subgoal labels effect in CS1. In *Proceedings of the 2019 ACM Conference on International Computing Education Research.* 209–217.

[13] Paul Denny, John Hamer, Andrew Luxton-Reilly, and Helen Purchase. 2008. Peer-Wise. In *Proceedings of the 8th International Conference on Computing Education Research.* 109–112.

[14] Paul Denny, John Hamer, Andrew Luxton-Reilly, and Helen Purchase. 2008. PeerWise: students sharing their multiple choice questions. In *Proceedings of the fourth international workshop on computing education research.* 51–58.

[15] Shayan Doroudi, Ece Kamar, and Emma Brunskill. 2019. Not everyone writes good examples but good examples can come from anywhere. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, Vol. 7. 12–21.

[16] Elena L Glassman, Aaron Lin, Carrie J Cai, and Robert C Miller. 2016. Learnersourcing personalized hints. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing.* 1626–1636.

[17] Philip J Guo, Julia M Markel, and Xiong Zhang. 2020. Learnersourcing at Scale to Overcome Expert Blind Spots for Introductory Programming: A Three-Year Deployment Study on the Python Tutor Website. In *Proceedings of the Seventh ACM Conference on Learning@ Scale.* 301–304.

[18] Mark Guzdial, Luke Hohmann, Michael Konneman, Christopher Walton, and Elliot Soloway. 1998. Supporting programming and learning-to-program with an integrated CAD and scaffolding workbench. *Interactive Learning Environments* 6, 1-2 (1998), 143–179.

[19] Minjie Hu, Michael Winikoff, and Stephen Cranefield. 2013. A process for novice programming using goals and plans. In *Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136.* 3–12.

[20] Robin Jeffries, Althea A Turner, Peter G Polson, and Michael E Atwood. 1981. The processes involved in designing software. *Cognitive skills and their acquisition* 255 (1981), 283.

[21] Dayu Jiang and Slava Kalyuga. 2020. Confirmatory factor analysis of cognitive load ratings supports a two-factor model. *Tutorials in Quantitative Methods for Psychology* 16 (2020), 216–225.

[22] Hyoungwook Jin, Minsuk Chang, and Juho Kim. 2019. SolveDeep: A System for Supporting Subgoal Learning in Online Math Problem Solving. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems.* 1–6.

[23] Hassan Khosravi, Gianluca Demartini, Shazia Sadiq, and Dragan Gasevic. 2021. Charting the design and analytics agenda of learnersourcing systems. In *LAK21: 11th International Learning Analytics and Knowledge Conference.* 32–42.

[24] Juho Kim. 2015. *Learnersourcing: improving learning with collective learner activity.* Ph.D. Dissertation. Massachusetts Institute of Technology.

[25] Kenneth R Koedinger, John C Stamper, Elizabeth A McLaughlin, and Tristan Nixon. 2013. Using data-driven discovery of better student models to improve student learning. In *International conference on artificial intelligence in education.* Springer, 421–430.

[26] Andy Kurnia, Andrew Lim, and Brenda Cheang. 2001. Online judge. *Computers & Education* 36, 4 (2001), 299–315.

[27] H Chad Lane and Kurt VanLehn. 2005. Teaching the tacit knowledge of programming to novices with natural language tutoring. *Computer Science Education* 15, 3 (2005), 183–201.

[28] Lauren E Margulieux and Richard Catrambone. 2019. Finding the best types of guidance for constructing self-explanations of subgoals in programming. *Journal of the Learning Sciences* 28, 1 (2019), 108–151.

[29] Lauren E Margulieux, Richard Catrambone, and Laura M Schaeffer. 2018. Varying effects of subgoal labeled expository text in programming, chemistry, and statistics. *Instructional Science* 46, 5 (2018), 707–722.

[30] Lauren E Margulieux, Mark Guzdial, and Richard Catrambone. 2012. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In *Proceedings of the ninth annual international conference on International computing education research.* 71–78.

[31] Lauren E Margulieux, Briana B Morrison, Baker Franke, and Harivololona Ramilison. 2020. Effect of Implementing Subgoals in Code. org's Intro to Programming Unit in Computer Science Principles. *ACM Transactions on Computing Education (TOCE)* 20, 4 (2020), 1–24.

[32] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In *Working group reports from ITiCSE on Innovation and technology in computer science education.* 125–180.

[33] Piotr Mitros. 2015. Learnersourcing of complex assessments. In *Proceedings of the Second (2015) ACM Conference on Learning@ Scale.* 317–320.

[34] Steven Moore, Huy Anh Nguyen, and John Stamper. 2021. Examining the Effects of Student Participation and Performance on the Quality of Learnersourcing Multiple-Choice Questions. In *Proceedings of the Eighth ACM Conference on Learning@ Scale.* 209–220.

[35] Briana B Morrison, Brian Dorn, and Mark Guzdial. 2014. Measuring cognitive load in introductory CS: adaptation of an instrument. In *Proceedings of the tenth annual conference on International computing education research.* 131–138.

[36] Briana B Morrison, Lauren E Margulieux, and Adrienne Decker. 2020. The curious case of loops. *Computer Science Education* 30, 2 (2020), 127–154.

[37] Briana B Morrison, Lauren E Margulieux, and Mark Guzdial. 2015. Subgoals, context, and worked examples in learning computing problem solving. In *Proceedings of the eleventh annual international conference on international computing education research.* 21–29.

[38] Eni Mustafaraj, Khonzoda Umarova, Franklyn Turbak, and Sohie Lee. 2018. Task-specific language modeling for selecting peer-written explanations. In *The Thirty-First International Flairs Conference.*

[39] Ha Nguyen, June Ahn, William Young, and Fabio Campos. 2020. Where's the Learning in Education Crowdsourcing?. In *Proceedings of the Seventh ACM Conference on Learning@ Scale.* 305–308.

[40] David N Perkins, Chris Hancock, Renee Hobbs, Fay Martin, and Rebecca Simmons. 1986. Conditions of learning in novice programmers. *Journal of Educational Computing Research* 2, 1 (1986), 37–55.

[41] Robert S Rist. 1989. Schema creation in programming. *Cognitive Science* 13, 3 (1989), 389–414.

[42] Robert S Rist. 1991. Knowledge creation and retrieval in program design: A comparison of novice and intermediate student programmers. *Human-Computer Interaction* 6, 1 (1991), 1–46.

[43] Anjali Singh, Christopher Brooks, Yiwen Lin, and Warren Li. 2021. What's In It for the Learners? Evidence from a Randomized Field Experiment on Learnersourcing Questions in a MOOC. In *Proceedings of the Eighth ACM Conference on Learning@ Scale.* 221–233.

[44] Yanan Sui, Vincent Zhuang, Joel W Burdick, and Yisong Yue. 2017. Multi-dueling Bandits with Dependent Arms. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence.*

[45] Marie Tarrant and James Ware. 2010. A comparison of the psychometric properties of three-and four-option multiple-choice questions in nursing assessments. *Nurse education today* 30, 6 (2010), 539–543.

[46] Xu Wang, Srinivasa Teja Talluri, Carolyn Rose, and Kenneth Koedinger. 2019. UpGrade: Sourcing student open-ended solutions to create scalable learning opportunities. In *Proceedings of the Sixth (2019) ACM Conference on Learning@ Scale.* 1–10.

[47] Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. 2018. A survey on online judge systems and their applications. *ACM Computing Surveys (CSUR)* 51, 1 (2018), 1–34.

[48] Sarah Weir, Juho Kim, Krzysztof Z Gajos, and Robert C Miller. 2015. Learnersourcing subgoal labels for how-to videos. In *Proceedings of the 18th ACM Conference*

on Computer Supported Cooperative Work & Social Computing. 405–416.

[49] Jacob Whitehill and Margo Seltzer. 2017. A Crowdsourcing Approach to Collecting Tutorial Videos–Toward Personalized Learning-at-Scale. In *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale*. 157–160.

[50] Joseph Jay Williams, Juho Kim, Anna Rafferty, Samuel Maldonado, Krzysztof Z Gajos, Walter S Lasecki, and Neil Heffernan. 2016. Axis: Generating explanations at scale with learnersourcing and machine learning. In *Proceedings of the Third (2016) ACM Conference on Learning@ Scale*. 379–388.

[51] Joseph Jay Williams, Anna N Rafferty, Dustin Tingley, Andrew Ang, Walter S Lasecki, and Juho Kim. 2018. Enhancing online problems through instructor-centered tools for randomized experiments. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[52] Albina Zavgorodniaia, Rodrigo Duran, Arto Hellas, Otto Seppala, and Juha Sorva. 2020. Measuring the cognitive load of learning to program: A replication study. In *United Kingdom & Ireland Computing Education Research conference*. 3–9.