석 사 학 위 논 문
Master's Thesis

AlgoSolve: 학습자 크라우드소싱 기반 마이크로태스크를 통한 하위목표 학습 시스템

AlgoSolve: Supporting Subgoal Learning through Learnersourced Microtasks

2021

최 갑 도 (崔 甲 到 Choi, Kabdo)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

석 사 학 위 논 문

AlgoSolve: 학습자 크라우드소싱 기반 마이크로태스크를 통한 하위목표 학습 시스템

2021

최 갑 도

한 국 과 학 기 술 원

전산학부

AlgoSolve: 학습자 크라우드소싱 기반 마이크로태스크를 통한 하위목표 학습 시스템

최 갑 도

위 논문은 한국과학기술원 석사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2021년 6월 17일

심사위원장 김 주 호 (인)

심 사 위 원 오 혜 연 (인)

심 사 위 원 유 신 (인)

AlgoSolve: Supporting Subgoal Learning through Learnersourced Microtasks

Kabdo Choi

Advisor: Juho Kim

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Daejeon, Korea
June 17, 2021

Approved by

Juho Kim
Professor of School of Computing

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

MCS 최갑도. AlgoSolve: 학습자 크라우드소싱 기반 마이크로태스크를 통한 하위목표 학습 시스템. 전산학부 . 2021년. 32+iv 쪽. 지도교수: 김주호. (영문 논문)
Kabdo Choi. AlgoSolve: Supporting Subgoal Learning through Learner-sourced Microtasks. School of Computing . 2021. 32+iv pages. Advisor: Juho Kim. (Text in English)

초 록

알고리즘 문제 풀이 초보자의 경우 문제 풀이에 있어서 많은 어려움을 겪으며, 주로 풀이를 구현하기 전에 제대로 구상하지 못하는 데에서 기인한다. 학습자의 문제 풀이 구상을 도와주기 위해 하위목표 학습 기법이 효과적으로 적용될 수 있음이 알려져 있다. 하지만 하위목표 학습 기법은 힌트, 전문가 레이블 등의 고품질 학습 자료가 같이 제공될 때 효과적이며, 이러한 고품질 자료는 다수 자습 환경에 적용되기 어렵다는 단점이 있다. 이를 극복하기 위해 본 연구에서는 하위목표 학습을 도와주기 위한 학습자 크라우드소싱 워크플로우를 제안한다. 2개 마이크로태스크로 구성되어 있는 학습자 크라우드소싱 워크플로우를 통해 학습자로부터 고품질 하위목표 레이블을 수집하고, 이를 활용해 하위목표 학습 과정을 효과적으로 지원할 수 있게 된다. 이러한 워크플로우를 알고리즘 문제 풀이에서 하위목표 학습을 도와주는 시스템인 AlgoSolve에 구현하였다. 본 연구에서 제안하는 워크플로우의 효과를 확인하기 위해 알고리즘 문제 풀이 초보자 63명을 대상으로 하는 실험을 진행하였으며, AlgoSolve가 더 높은 품질의 하위목표 레이블을 생성하고 학습 과정에서 배운 풀이 기법을 유사한 문제에 적용하는 데 도움을 줄 수 있음을 확인하였다.

핵 심 낱 말 하위목표 학습, 알고리즘 문제 풀이, 마이크로태스크, 학습자 크라우드소싱

Abstract

Algorithmic problem-solving has enabled scalable learning opportunities for learning programming. However, novices often struggle to solve problems successfully, where one of the main reasons arises from their inability to develop solution plans before solving the problem. Subgoal learning has been shown effective in promoting learners' ability to develop more complete solution plans. However, subgoal learning is most effective when expert-crafted guidance is provided, which is not widely available in self-learning environments. To overcome this problem, we developed a learnersourcing workflow composed of two microtasks that can collect high-quality subgoal labels and provide guidance in subgoal learning using these as high-quality examples. We implemented the workflow into AlgoSolve, a prototypical interface that supports subgoal learning in algorithmic problem-solving. Results from a between-subjects study with 63 novices demonstrate that AlgoSolve successfully guides learners to create high-quality subgoal labels and develop more complete solution plans.

Keywords Subgoal learning, algorithmic problem-solving, microtasks, learnersourcing

Contents

List of Tables

List of Figures

Chapter 1. Introduction

Learning to program has gained significant popularity in recent years. Algorithmic problem-solving has been successfully applied to teaching and learning programming and program design [1]. In algorithmic problem-solving, learners submit a code that solves a given problem which typically requires time- and memory-efficient solutions, and the code gets automatically graded based on a set of testcases. Algorithmic problem-solving enabled scalable learning opportunities, not only in traditional classroom settings but also widely applicable in self-learning environments such as Massive Open Online Courses (MOOCs) or Online Judge [2] systems.

One of the characteristics of algorithmic problem-solving is that planning out the solution before coding is crucial for successfully solving the problem [3]. Despite its importance, it is well known that novices tend to overlook the planning stage, sometimes skipping this stage entirely [3, 4]. These behaviors often result in producing incorrect or inferior solutions [3].

Prior work has shown that subgoal learning – a method where students learn solution procedures by decomposing the solution into functional pieces (i.e., subgoals) – helped learners create more complete solution plans [5]. Subgoal learning is known to be most beneficial when learners create their own subgoal labels with appropriate guidance, such as expert-created labels or hints [6]. However, these types of guidance require the presence of experts, which is not always available in many self-learning environments.

In this work, we propose a learnersourcing workflow that provides guidance in subgoal learning with microtasks that are geared toward promoting learners' understanding of the subgoals in a solution, while collecting high-quality subgoal labels that reflect a complete understanding of the given code. The microtasks take advantage of the collected subgoal labels and provide scalable learning activities to learners while using the learner input to improve the system. We developed two types of microtasks that make use of high-quality peer examples: Subgoal Voting and Subgoal Labeling. Learners first choose the best subgoal label examples in the Subgoal Voting task and familiarize themselves with high-quality subgoal labels by making comparisons among the given examples, and then create their own subgoal labels, with an opportunity of improvement by comparing with high-quality examples in the Subgoal Labeling task.

To evaluate our learnersourcing workflow, we developed AlgoSolve, a prototypical interface that supports subgoal learning with microtasks. We conducted a between-subjects study with 63 novices in algorithmic problem-solving and compared AlgoSolve against a baseline interface that implements a method known to be most effective for subgoal learning [6]. Results show that AlgoSolve successfully guided learners in recognizing high-quality subgoal labels and creating high-quality examples themselves. Participants who learned subgoals using AlgoSolve were also more capable of applying the solution technique they have learned for solving novel problems.

The contributions of this work are as follows:

- A learnersourcing workflow that guides learners to provide higher-quality subgoal labels using previously submitted high-quality examples.
- AlgoSolve, a prototypical interface that supports subgoal learning by providing high-quality peer-created subgoal label examples.

- Results from a between-subjects study indicating that AlgoSolve successfully supports subgoal learning and helps learners become better at solution planning, followed by implications for designing learnersourcing workflows.

Chapter 2. Background and Related Work

We review the theoretical background of our work and previous work on existing learnersourcing approaches.

2.1 Solution Planning and Methods for Supporting Planning

The algorithmic problem-solving process is composed of several stages: 1) understanding and interpreting the problem text, 2) designing a plan that can solve the problem, 3) implementing the solution code, and 4) testing and debugging the solution [7]. Among these stages, researchers suggested that properly designing a solution plan is key for successful problem-solving [3]. Learners should be able to derive a solution plan from the given problem text, and then transform these plans into code. Being able to develop high-level plans, and then connect these plans with code, but novices often show bad planning behaviors such as diving straight into implementation without going through the planning stage [4] or failing to connect the developed plans with code [3].

Several methods that aim to support problem-solving by guiding the planning stage have been proposed, primarily focusing on using representations that guide learners to externalize their ideas into structured solution plans that they can refer to when writing code. A popular approach is the use of flowcharts [8, 9]. However, one downside of using flowcharts is that these are inherently code-level, which might not be appropriate for learners to develop high-level solution plans for complex problems.

Another body of research aims to make learners focus on higher level tasks. GPCeditor [10] is a programming environment which asks learners to decompose the problem into goals and plans, instantiate plans into code, and then compose the code into a final program. This approach of guiding learners through a process of describing plans was further investigated by other researchers [11, 12, 13], demonstrating the benefits of the approach.

2.2 Subgoal Learning

Subgoal learning [14] is a learning method where learners study solutions with subgoal labels – short textual descriptions that describe the function of a group of solution steps sharing the same goal. Subgoals highlight the high-level solution structure generalizable to other problems. By encouraging learners to memorize this conceptual structure of the solution instead of individual solution steps, they become better at solving other problems that use the same solution but differ in specific solution steps [15, 16] and building more complete solution plans [5]. Subgoals can also be directly used in the problem-solving process, such as putting pre-written comments as building blocks for code prior to implementation [17]. Numerous studies have shown that subgoal learning was effective at improving learners' problem-solving performance in procedural domains such as mathematics [15, 18], chemistry [19], and programming [6, 16, 20, 21].

Subgoal learning was initially employed using a passive method; learners passively read the pre-written labels for a given worked example. However, passive learning is known to be less effective compared to other types of learning activities (active, constructive, and interactive) [22]. Recently, Margulieux and Catrambone [6] investigated how different types of learning activity and guidance can

affect learner performance in problem-solving and discovered notable differences between learning conditions. Learners showed the best performance when they constructively engaged in learning subgoals (i.e., constructing their own subgoal labels) where the solution was pre-grouped into subgoals, but only with appropriate guidance; hints that describe the similarities of the solution steps or expert-created labels as correct response feedback. Receiving both instances of guidance decreased learner performance. Although these types of guidance are effective at guiding improving subgoal learning, these require high-quality materials, created by experts, which is not always present in self-learning settings. Based on the findings from previous research on subgoal learning, we develop a learnersourcing workflow that can provide effective guidance to learners using peer submissions.

2.3 Learnersourcing Approaches

Learnersourcing [23] is a crowdsourcing method where learners engage in creating learning materials for future learners, while having meaningful learning experiences themselves. Learnersourcing has been successfully applied in collecting learning resources that are comparable to expert quality materials, while providing scalable opportunities for learning. AXIS [24] collects explanations for math problems by asking learners to generate, revise, and evaluate explanations, where a machine learning approach is applied to provide high-quality peer explanations to future learners. Wang et al. [25] proposed UpGrade, 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 with substantially less time. Other learnersourcing approaches aim to generate hints for improving student solutions [26], subgoal labels for instructional videos [23] or mathematical problems [27], tutorial videos [28], explanations for programming misconceptions [29], or even complex design problems [30].

Researchers have explored learnersourcing workflows that can generate high-quality subgoal labels. Crowdy [23] implements a three-stage learnersourcing workflow for generating subgoal labels on how-to videos, where the labels are first generated, evaluated, and then proofread to ensure its quality. The learner-generated subgoals were found to be comparable in quality to that of experts. However, researchers also reported that there were cases where learners in the final stage did not get much practice on good subgoals, which resulted in lowered quality of the final outcome. Jin et al. [27] also developed 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. Doroudi et al. [31] has shown that crowdworkers can produce high-quality outcomes by receiving high-quality examples. Conversely, giving low-quality examples to learners might harm their learning experience. Following this stream of research, we aim to design our learnersourcing workflow so that it can collect high-quality subgoal labels and provide them as examples to encourage learners to create subgoal labels of better quality.

Chapter 3. AlgoSolve

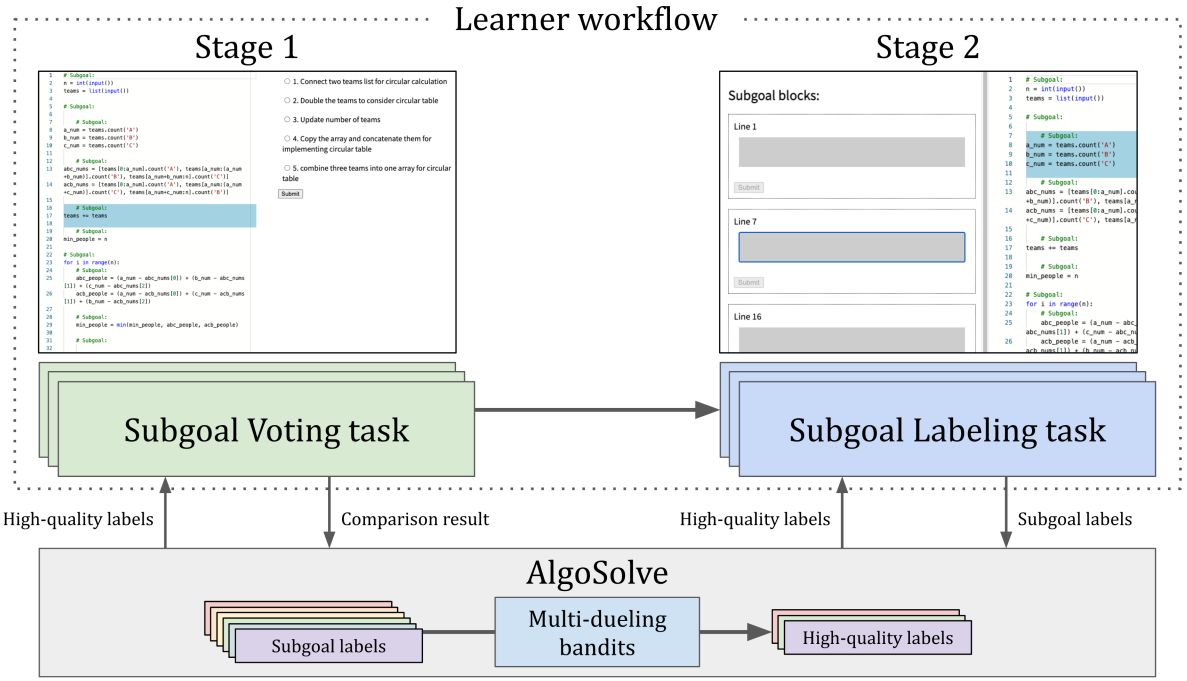

Figure 3.1: Overview of AlgoSolve and the learner workflow. Learners first gain an understanding of good examples of subgoal labels by going through a set of Subgoal Voting tasks, and then proceed to a set of Subgoal Labeling tasks where they create their own labels.

We introduce AlgoSolve, a system that gathers high-quality subgoal labels and provides the collected labels as examples during the subgoal learning microtasks to future learners. We first describe an overview of the learnersourcing workflow used in AlgoSolve, which is composed of two types of microtasks, and then explain the details of each microtask. Finally, we introduce the policy for selecting subgoal label examples.

3.1 Microtask Design

Microtasks should be able to successfully guide subgoal learning using high-quality subgoal labels. Meanwhile, the system needs to collect high-quality examples from learners to provide the best learning experience. We designed the learning activity as a workflow that consists of two types of microtasks to effectively support subgoal learning: 1) Subgoal Voting and 2) Subgoal Labeling. By learning subgoals through the two sets of microtasks, learners can familiarize themselves with high-quality subgoal labels and become able to create high-quality labels themselves.

Learners first go through a series of Subgoal Voting tasks, where they make comparisons between subgoal label examples and learn how to create good subgoal labels before providing their own labels, in the form of multiple-choice questions (Figure 3.2). For each question, they are then given the information on how many previous learners chose each of the labels as feedback (Figure 3.3). For the given example

in Figure 3.3, the learner can realize that the second option, "double the teams to consider circular table", was preferred by previous learners over the first option which the learner chose. After completing all the Subgoal Voting tasks, learners proceed to the Subgoal Labeling tasks, where they create their own subgoal labels (Figure 3.4). After completing their initial labels, they have a chance of improving their labels by comparing against high-quality labels submitted by previous learners (Figure 3.5). For example, the learner in Figure 3.5 could easily spot the differences between their own label and the provided peer labels, such as using a more precise word 'count' instead of 'get', or explicitly mentioning the team names (A, B, and C) and reflect these in their final submissions.

Both types of tasks are generated from a worked example that is already decomposed into subgoals, where the subgoals are randomly assigned to either of the tasks. The worked example and the scope and hierarchy of the subgoals for the worked example are generated prior to the activity. Deciding the appropriate subgoal scopes could be also developed as a learnersourcing task, however we did not include in the current work in order to focus on gathering quality subgoal labels.

In order to support learners in learning subgoals using high-quality examples, the system needs to have such label examples at the beginning. AlgoSolve first gathers initial examples by providing learners with the Subgoal Labeling task alone, without the support from peer examples. After the system collects three different examples (i.e., examples that are not completely identical) for each subgoal, the system provides the full set of tasks with peer example support.

3.1.1 Task 1: Subgoal Voting

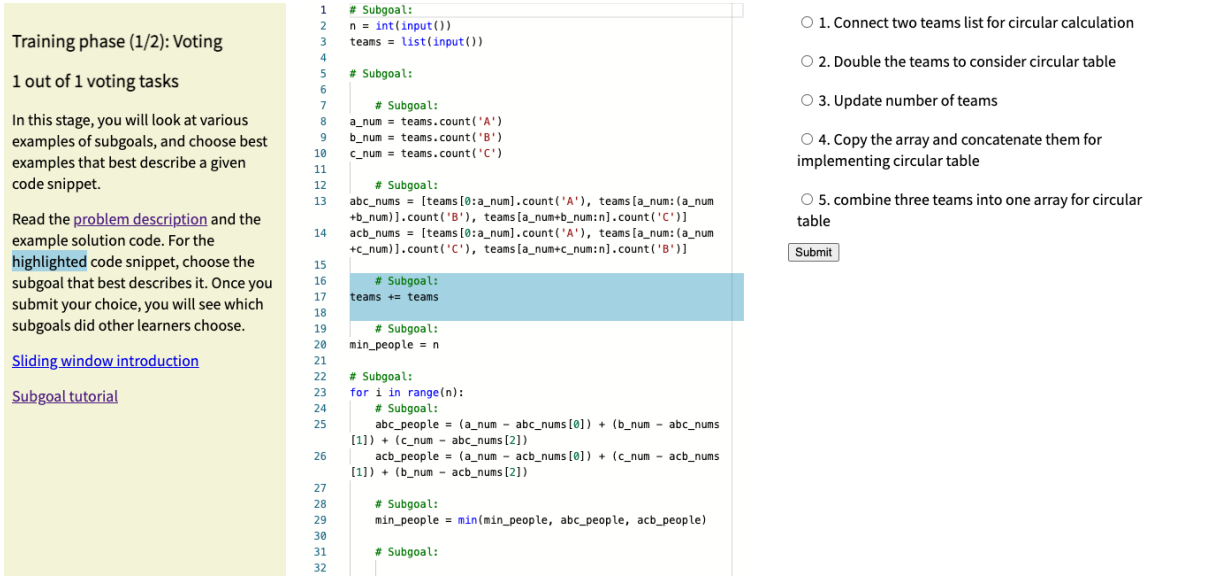

Figure 3.2: Overview of the Subgoal Voting task. Learners are given up to five subgoal labels and are asked to select a single option that best describes the given subgoal.

The Subgoal Voting task (Figure 3.2) is designed as multiple-choice questions that asks learners to select a subgoal label example that best describes a given code segment. Learners are given up to five examples, and selects a subgoal label example that best describes a given code snippet. The system uses the learners' selections to determine the quality differences between subgoal labels and which labels to show to future learners as examples.

We designed the task so that it can help learners quickly understand the examples of good subgoal labels. Learners receive three system-selected high-quality subgoal label examples and up to two randomly-selected examples. We decided to include both high-quality and random – likely to be of lower quality – so that learners could easily spot the differences between good and bad labels by differentiating high-quality examples against random examples. We describe the policy for selecting the high-quality examples in a separate section.

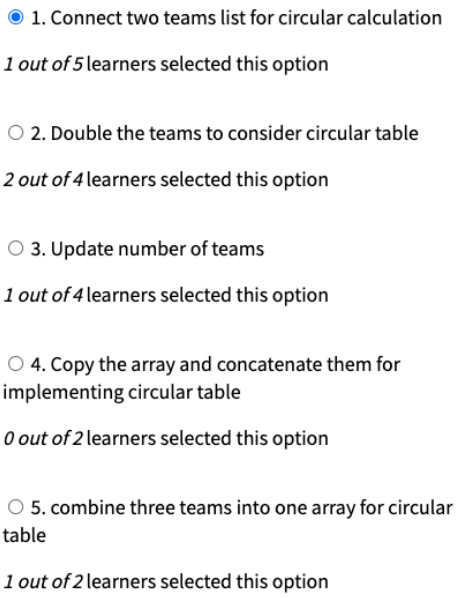

Figure 3.3: Peer consensus information after the learner submit their vote in the Subgoal Voting task.

Multiple-choice questions are typically accompanied with 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 distinction between correct and incorrect options, the system instead provides peer consensus on how other learners made the choices by showing the number of times each example was selected (Figure 3.3).

3.1.2 Task 2: Subgoal Labeling

Now that learners have gained an understanding of high-quality subgoal labels, the system then asks learners to provide their labels in the Subgoal Labeling task (Figure 3.4). Learners first submit their initial work (initial subgoal label), and then resubmit their final descriptions (final subgoal labels) after viewing feedback given by the system (Figure 3.5). Only the final, improved labels are collected by the system and provided to future learners. Labels that are identical to one of the previous labels are removed from the system to prevent unnecessary duplicates. The system provides three peer-created subgoal labels that are known to be the most high-quality examples, where learners can make comparisons with their initial labels and make iterations.

The Subgoal Labeling task is grounded on prior work on subgoal learning [6], and resembles the most effective method (guided constructive learning with feedback) for guiding learners to successfully build mental organizations of the solution. The authors of prior work [6] argue that guidance should be carefully thought over; poorly designed guidance, such as providing both hints and expert-created labels as feedback, could hinder the learning effects by dismantling the learner's prior understanding of the

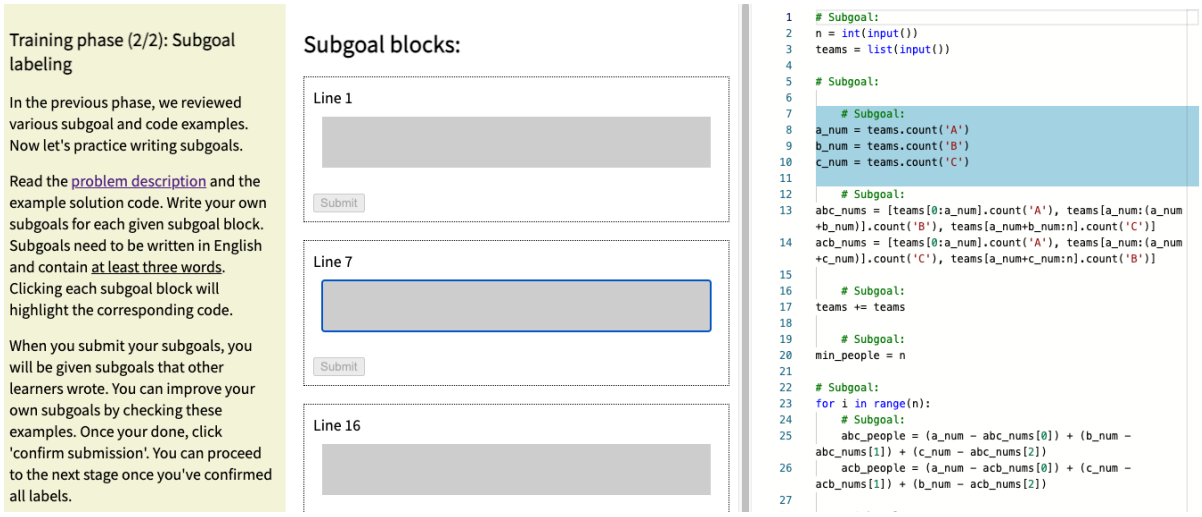

Figure 3.4: Overview of the Subgoal Labeling task. Learners provide their own subgoal labels that describe the given subgoal.

solution and making them blindly apply the expert labels, resulting in diminished learning benefits. We address this issue by intentionally not positing the examples as being feedback nor of high-quality, so that learners do not perceive the given examples as 'correct' answers and become more likely to preserve their own explanations.

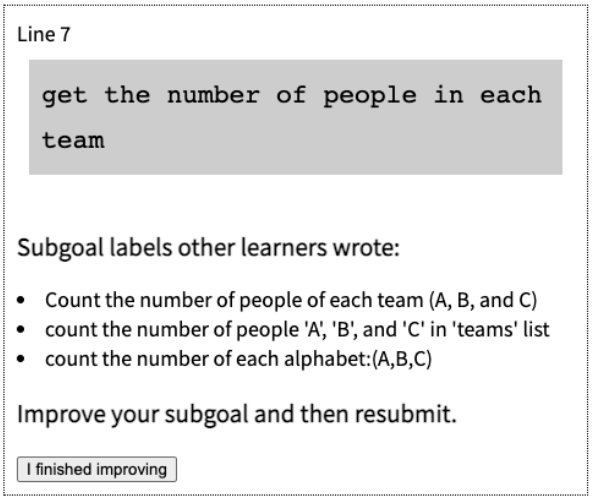

Figure 3.5: After the learner submit their initial labels in the Subgoal Labeling task, system-selected peer examples are shown to the learner.

3.2 Subgoal Label Example Selection Policy

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 [23, 26, 27] to choose the best examples. Another approach is to employ machine learning to dynamically determine examples that will be shown to learners, such as multi-armed bandits [24, 32]. 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 best quality (exploitation).

In AlgoSolve, the system 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 [33] – a variant of multi-armed bandit problem where the algorithm selects multiple arms, and then observes the result of pairwise duels between the selected arms. We use IndSelfSparring [34], a multi-dueling bandits algorithm which 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 prevented all labels that were exactly the same with a previously submitted label after removing non-alphabetical characters from being selected by the system.

Chapter 4. Evaluation

4.1 Study Design

We conducted a between-subjects study which compares AlgoSolve (**microtask** condition) to a baseline interface (**baseline** condition). The baseline interface implements the constructive learning method accompanied with expert-created subgoal labels as feedback, which is known to be most effective in subgoal learning [6]. We randomly assigned half of the subgoals to the Subgoal Voting task, and the remaining half to the Subgoal Labeling task for the microtask condition for each participant. Participants in the baseline condition were asked to write subgoal labels for all subgoals.

4.2 Participants

We recruited participants who are novices in algorithmic problem-solving from two universities and a popular online judge system. Among the participants, the first five created the initial seed subgoal labels for AlgoSolve by using the baseline interface, and were excluded from the analysis. The rest were randomly assigned to one of the conditions. In the end, 63 participants (male: 46, female: 17, mean age: 23.4) completed the study session, where 31 used the baseline interface and 32 used AlgoSolve. Participants were compensated after completing the study.

We asked participants to report their proficiency in algorithmic problem-solving, familiarity of the topic covered in the study, and confidence in writing explanations in English in a 7-point Likert scale. We found no significance between the conditions for any of the questions.

4.3 Study Materials

The Sliding Window technique [35] was selected as the topic for the current study. We chose Sliding Window since it is a relatively simple technique which requires less background knowledge on other topics in algorithms, and therefore learner performance would be less affected by their previous experience on algorithmic problem-solving. We selected two algorithmic problems (training problem [1], assessment problem [2]) which are solvable using the technique. We sampled the problems so that they are challenging enough and have similar levels of difficulty. The level of difficulty was extracted from an expertsourced repository of problem difficulty [3]. Each problem was used in the training session and the assessment task.

The subgoal-grouped worked example and expert subgoal labels were created through discussion between the experimenters and a problem-solving expert who had significant experience in problem-solving. The created worked example consisted of 14 subgoals. The worked example with expert subgoal labels is shown in Figure 4.1.

[1] https://www.acmicpc.net/problem/16310
[2] https://www.acmicpc.net/problem/3078
[3] https://solved.ac/

```
# Subgoal 1: Get the input values
n = int(input())
teams = list(input())
# Subgoal 2: Set up the initial values for sliding window
    # Subgoal 3: Calculate the number of people in each team
a_num = teams.count('A')
b_num = teams.count('B')
c_num = teams.count('C')
    # Subgoal 4: For each possible team formation, calculate the number of people who are correctly
    ↪  seated
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 5: Since both ends of the seat are connected, duplicate the sitting status list
teams += teams
    # Subgoal 6: Set up the initial minimum value holder
min_people = n
# Subgoal 7: Slide the window and update the minimum value holder
for i in range(n):
    # Subgoal 8: Calculate the number of people who have to move seats to fit the team formation
    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 9: Update the minimum value holder
    min_people = min(min_people, abc_people, acb_people)
    # Subgoal 10: Slide the window to the next index and update the number of correctly seated people
        # Subgoal 11: Handle the person who is moved out from the first team
    if teams[i] == 'A':
        abc_nums[0] -= 1
        acb_nums[0] -= 1
    elif teams[i] == 'B':
        acb_nums[2] += 1
    else:
        abc_nums[2] += 1
        # Subgoal 12: Handle the person who is moved out from the second team
    if teams[i + a_num] == 'A':
        abc_nums[0] += 1
        acb_nums[0] += 1
    elif teams[i + a_num] == 'B':
        abc_nums[1] -= 1
    else:
        acb_nums[2] -= 1
        # Subgoal 13: Handle the person who is moved out from the third team
    if teams[i + a_num + b_num] == 'B':
        abc_nums[1] += 1
    elif teams[i + a_num + b_num] == 'C':
        abc_nums[2] -= 1
    if teams[i + a_num + c_num] == 'C':
        acb_nums[1] += 1
    elif teams[i + a_num + c_num] == 'B':
        acb_nums[2] -= 1
# Subgoal 14: Print the minimal number of people who need to move seats
print(min_people)
```

Figure 4.1: Worked example used in the training session, labeled with expert-created subgoal labels. The hierarchy between subgoals are represented as indentation levels.

4.4 Procedure

An overview of the study procedure is shown in Table 4.1. Participants first completed the pre-questionnaire and reported their proficiency and expertise in algorithmic problem-solving. Before starting the training session, participants were given instructional materials on the Sliding Window technique and subgoal labels. In the training 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 that measures participants' ability to plan out solutions for a given problem using the technique they have learned from the training session.

<table>
<tr><th>Time (mins)</th><th>Baseline</th><th>Microtask</th></tr>
<tr><td colspan="3">Pre-session (20)</td></tr>
<tr><td>5</td><td colspan="2">Introduction & Consent</td></tr>
<tr><td>5</td><td colspan="2">Pre-questionnaire</td></tr>
<tr><td>5</td><td colspan="2">Sliding Window technique introduction</td></tr>
<tr><td>5</td><td colspan="2">Subgoal label tutorial</td></tr>
<tr><td colspan="3">Training session (30)</td></tr>
<tr><td rowspan="2">30</td><td rowspan="2">Baseline subgoal labeling task</td><td>Subgoal Voting task</td></tr>
<tr><td>Subgoal Labeling task</td></tr>
<tr><td colspan="3">Post-session (25)</td></tr>
<tr><td>5</td><td colspan="2">Post-questionnaire</td></tr>
<tr><td>20</td><td colspan="2">Assessment task</td></tr>
</table>

Table 4.1: Overview of the study procedure.

4.5 Measurements

We asked learners of their experience in using the system. In the post-questionnaire, learners were asked to rate their cognitive load, total time spent for the training session, helpfulness of the provided subgoal labels, and helpfulness of the activity. For measuring cognitive load, we used the CS Cognitive Load Component Survey (CS CLCS) [36], a cognitive load measurement specialized for the context of computer science 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.

In order to investigate how the microtasks affected learners in creating quality subgoal labels, we qualitatively analyzed the labels learners submitted. We developed four categories for subgoal labels, which is shown in Table 4.2. The labels were first categorized based on whether the label contained any incorrect explanation 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 understood the solution, which is the primary goal of subgoal learning. Representative examples for each level are shown in Table 4.4.

We recruited four experts to measure the quality of learner-created subgoal label examples selected by AlgoSolve. Experts had significant experience in solving algorithmic problems and/or teaching al-

Category	Label	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 lack depth in certain aspects.
Deep-level	L3	Labels that successfully explains the purpose of the given code.

Table 4.2: Categories of subgoal label quality used in the current study.

gorithmic problem-solving. We asked experts 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 are 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. Experts rated the labels as either system better, expert better, or matching in terms of quality.

To assess the learners' ability to build solution plans in the assessment task, we used SOLO taxonomy [37], which measures how well learners understood the topic and produced higher quality, structurally more complex learning output. The SOLO taxonomy consists of five categories: prestructural, unistructural, multistructural, relational, and extended abstract. We provide the criteria we used in the current study in Table 4.3, which is a variant of prior research on assessing the effect of subgoal learning in planning solutions [5].

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 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 4.3: SOLO scoring criteria used in the current study.

For coding subgoal labels and solution plans, we used a similar method to prior work [5]. The first five sets of submissions (for each subgoal for labels) were graded together by all three raters 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 in the results. After repeating the process for 20% of the submissions, we measured inter-rater reliability. For measuring inter-rater reliability,

we used the intraclass correlation coefficient with absolute agreement (i.e., $ICC(A)$) following prior work [5, 6]. The raters achieved a high level of agreement for both types of submissions; 0.90 for grading subgoal labels and 0.97 for grading solution plans. The remaining 80% of the submissions were graded by a single rater.

	L0	L1	L2	L3
S1	input grouping analysis	Declaration of variables n and teams	convert input into value	get the input
S2	settings for the overall team	Declare and initialize variables.	Preprocess for calculating output	Initial setting for Sliding Window
S3	count people in corresponding locations	Find the number of A, B, and C in a variable named team	count how many a,b,c in teams	count number of members for each team
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
S6	count number of people	Initialize min_people with n	Set initial value of result as n	Initialize the result as max to prevent wrong answer
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
S8	count abc and acb people for n times	Calculate abc_people & acb_people with in for loop.	calculate needed changes	count num. of people who are not in proper group order
S9	decide the array of circular table	update min_people variable	Calculate minimum value for output	Calculate the minimum number of people need to move
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
S11	count number of people who need to move in the first group	Update abc_nums and abc_nums based on which team i-th person belongs to	Update the number of people of the first group	Update the "correct" sitting number of people based on the value of ith value
S12	consider case of starting point is i+1	Update abc_nums and acb_nums	Update abc_nums and acb_nums according to changes in second group.	update 'ABC' and 'ACB' order by considering second group
S13	update the minimum movements for the third group	update the number of each case	Slide the starting point to the right by the number of (A+B) or (A+C), then correct the value of the list.	update 'ABC' and 'ACB' order by considering sliding window
S14	finalize the minimum number	print the value of min_people	print the number of people who moved	Print the minimal number of people who need to move

Table 4.4: Representative examples of subgoal labels for each label score. Each row represents a subgoal (e.g., S1: Subgoal 1).

Chapter 5. Results

5.1 Qualitative Analysis of the Submitted Subgoal Labels

The analysis results on the quality of the subgoal labels are shown in Table 5.1. Figure 5.1 presents the distribution of the labels in terms of quality score. Overall, participants who submitted subgoal labels through microtasks created fewer improper labels (L0 labels). 22% of the initial labels were classified as L0 (compared to 30% of baseline), which was reduced to 15% after comparison with peer examples. Meanwhile, microtask participants expressed a deeper understanding of the solution, creating more labels with deep-level explanations (L3 labels). Microtask participants created L3 labels in 27% of the total labels, later improved to 43% in their final submissions, compared to 20% in the baseline condition. The Cochran-Armitage trend test between each group of labels showed that the differences in the quality score distribution were statistically significant between each pair of groups (baseline-microtask (initial): $p < 0.004$, microtask (initial)-microtask (final): $p < 0.002$).

Among the 163 labels that were not initially counted as L3, 55 (34%) labels were improved after comparison in terms of label score. Only seven labels showed a decrease in label score. Table 5.2 summarizes the amount of change made in terms of label score. These results suggest that the microtasks were able to encourage participants to create labels that are more correct, more complete, and of higher quality.

Providing peer examples could lead to learners blindly following these examples by directly copying them. In order to identify occurrences of copying, we inspected cases where a participant's label was identical to another label. We discovered 13 labels (6% of the total labels) that were exactly the same with a previously submitted label, where four of them were submitted by a single participant.

We discuss two notable cases: 1) subgoals where the quality of the labels got drastically improved in the final submissions (S4 and S5), and 2) the differences between compound subgoals (S2, S7, S10).

5.1.1 Subgoals where Label Quality was Significantly Improved After Comparison

We observed significant differences between the initial and final submissions from microtask participants in S4 and S5. The main reason for the low initial quality was that participants were unsuccessful in figuring out the intent behind the code (e.g., concatenation in S5). In the Subgoal Labeling microtask, however, participants realized the inferiority of such labels and were able to provide high-quality labels in the end, implying that peer comparison took an important role in improving explanation quality of the submitted labels.

5.1.2 Compound Subgoals

Compound subgoals (S2, S7, S10) – high-level subgoals composed of lower-level subgoals – were expected to be challenging to learners since it requires the ability to understand a larger block of code and summarize its purpose, which could be difficult to novices in problem-solving. In general, participants showed poor performance in creating high-quality labels. However, we observed a significant difference

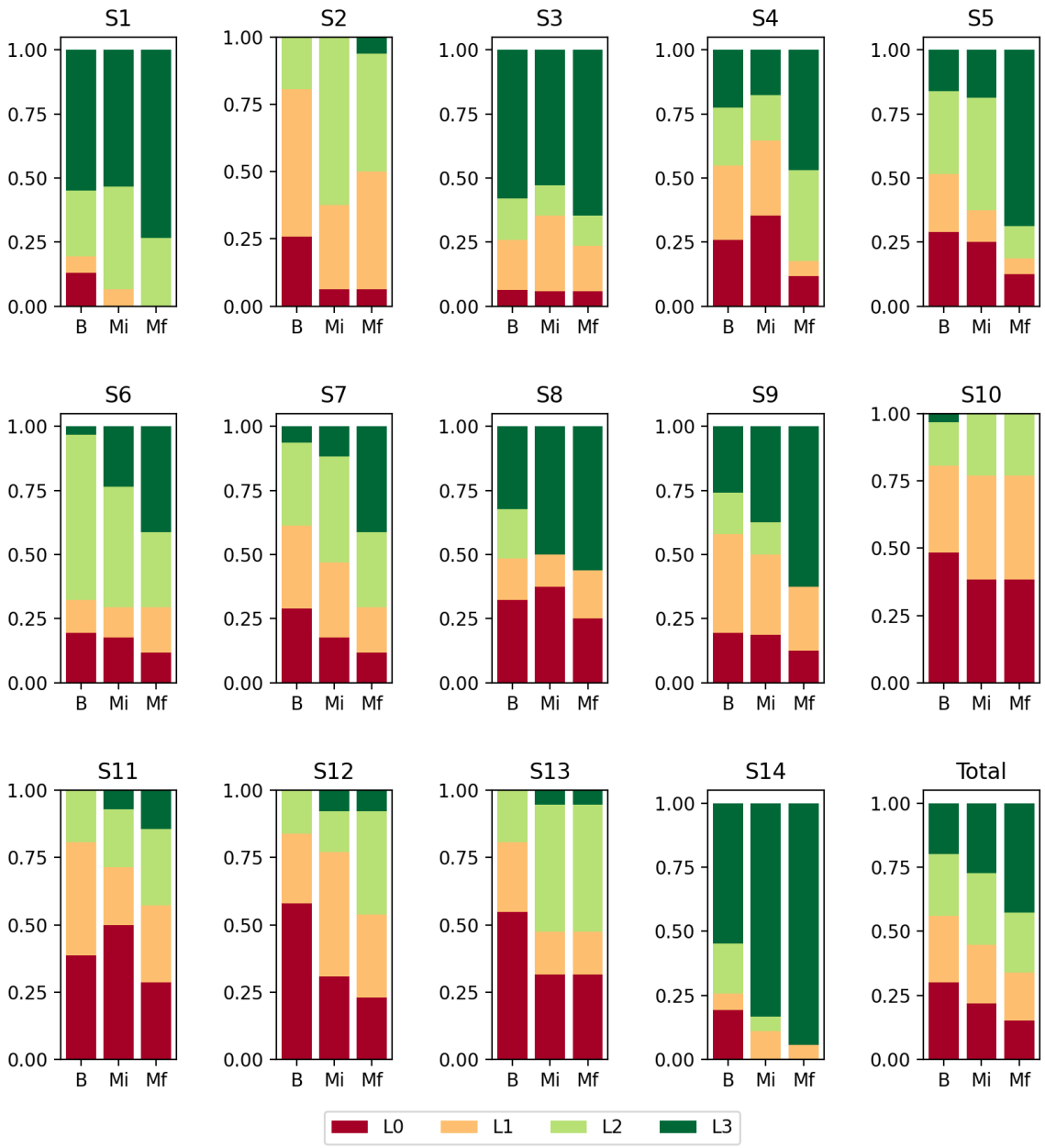

Figure 5.1: Distributions of label quality scores for each subgoal (S1 S14) and the aggregated result (B: baseline, Mi: microtask - initial submission, Mf: microtask - final submission).

in the scores between labels submitted by baseline participants and those submitted by microtask participants as final submissions in S2 and S7, and almost half of the participants being able to express a deep understanding of the subgoal (i.e., L3) in S7. Also, seven out of 15 incomplete labels (i.e., not in L3) in S7 showed an increase in terms of explanation quality, five of them reaching L3. We discovered that the voting pattern in S7 was different from other subgoals in that the majority of the participants' votes were given to labels marked as L3 (8/13, compared to 4/12 in S2 and 0/19 in S10).

S#	L0	L1	L2	L3	Total
S1	4 (13%)	2 (6%)	8 (26%)	17 (55%)	31
	0 (0%) → 0 (0%)	1 (7%) → 0 (0%)	6 (40%) → 4 (27%)	8 (53%) → 11 (73%)	15
S2	8 (26%)	17 (55%)	6 (19%)	0 (0%)	31
	1 (6%) → 1 (6%)	5 (31%) → 7 (44%)	10 (63%) → 7 (44%)	0 (0%) → 1 (6%)	16
S3	2 (6%)	6 (19%)	5 (16%)	18 (58%)	31
	1 (6%) → 1 (6%)	5 (29%) → 3 (18%)	2 (12%) → 2 (12%)	9 (53%) → 11 (65%)	17
S4	8 (26%)	9 (29%)	7 (23%)	7 (23%)	31
	6 (35%) → 2 (12%)	5 (29%) → 1 (6%)	3 (18%) → 6 (35%)	3 (18%) → 8 (47%)	17
S5	9 (29%)	7 (23%)	10 (32%)	5 (16%)	31
	4 (25%) → 2 (13%)	2 (13%) → 1 (6%)	7 (44%) → 2 (13%)	3 (19%) → 11 (69%)	16
S6	6 (19%)	4 (13%)	20 (65%)	1 (3%)	31
	3 (18%) → 2 (12%)	2 (12%) → 3 (18%)	8 (47%) → 5 (29%)	4 (24%) → 7 (41%)	17
S7	9 (29%)	10 (32%)	10 (32%)	2 (6%)	31
	3 (18%) → 2 (12%)	5 (29%) → 3 (18%)	7 (41%) → 5 (29%)	2 (12%) → 7 (41%)	17
S8	10 (32%)	5 (16%)	6 (19%)	10 (32%)	31
	6 (38%) → 4 (25%)	2 (13%) → 3 (19%)	0 (0%) → 0 (0%)	8 (50%) → 9 (56%)	16
S9	6 (19%)	12 (39%)	5 (16%)	8 (26%)	31
	3 (19%) → 2 (13%)	5 (31%) → 4 (25%)	2 (13%) → 0 (0%)	6 (38%) → 10 (63%)	16
S10	15 (48%)	10 (32%)	5 (16%)	1 (3%)	31
	5 (38%) → 5 (38%)	5 (38%) → 5 (38%)	3 (23%) → 3 (23%)	0 (0%) → 0 (0%)	13
S11	12 (39%)	13 (42%)	6 (19%)	0 (0%)	31
	7 (50%) → 4 (29%)	3 (21%) → 4 (29%)	3 (21%) → 4 (29%)	1 (7%) → 2 (14%)	14
S12	18 (58%)	8 (26%)	5 (16%)	0 (0%)	31
	4 (31%) → 3 (23%)	6 (46%) → 4 (31%)	2 (15%) → 5 (38%)	1 (8%) → 1 (8%)	13
S13	17 (55%)	8 (26%)	6 (19%)	0 (0%)	31
	6 (32%) → 6 (32%)	3 (16%) → 3 (16%)	9 (47%) → 9 (47%)	1 (5%) → 1 (5%)	19
S14	6 (19%)	2 (6%)	6 (19%)	17 (55%)	31
	0 (0%) → 0 (0%)	2 (11%) → 1 (6%)	1 (6%) → 0 (0%)	15 (83%) → 17 (94%)	18
Total	130 (30%)	113 (26%)	105 (24%)	86 (20%)	434
	49 (22%) → 34 (15%)	51 (23%) → 42 (19%)	63 (28%) → 52 (23%)	61 (27%) → 96 (43%)	224

Table 5.1: Qualitative analysis results for learner-submitted subgoal labels. For each row, the first row represents the baseline participants and the second row represents the microtask participants, denoted as initial → final.

5.2 Effects on Solution Planning Ability

The frequency of SOLO scores on participants' solution plans are shown in Table 5.3. In both of the conditions, the majority of the participants received the lowest score (i.e., prestructural), and none received the highest score possible (i.e., extended abstract). It should be noted that not all of the solution plans were considered as nonsensical answers, but were inefficient solutions that did not make use of the solution technique they have learned. The Cochran-Armitage trend test revealed statistically significant differences between the conditions ($p < 0.05$). 31% of the microtask participants received a score of three or higher, meaning that they were able to correctly apply the technique to novel problems, in contrast to only 6% for the baseline condition, implying that the microtasks can successfully guide learners in

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

Table 5.2: Changes made in the labels during the Subgoal Labeling microtask, in terms of label quality score.

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

thoroughly understanding the conceptual structure of the solution.

5.3 Quality of the System-selected Subgoal Labels

Table 5.4 presents the results of the comparative evaluation between system-selected labels and expert-created labels. Each of the labels received an equal number of preferences of three. Three labels were considered to have comparable quality. Five labels showed no majority, where two of the labels were a tie between 'matching' and system label preference. This result indicates that, even from a small population of roughly 30 learners, the system is able to determine and provide subgoal labels of decent quality that are comparable with expert-created labels.

Experts preferred the expert-created labels because these were better at describing the high-level, abstract purpose of the subgoal. For example in S10, the expert label included the phrase "slide the window", which is a key term for explaining its purpose. Meanwhile, the system-selected labels were perceived as better labels as these provided a more detailed explanation of the code, such as "in the worst case as n" (S6) or "for two patterns of ordering" (S8). However, experts noted that the system label in S13 can easily be misinterpreted, which implies the necessity of microtasks that are designed for validation or proofreading of subgoal labels.

5.4 Peer Consensus on Subgoal Label Examples

Most of the subgoals selected by the system were clearly preferred by participants (i.e., selected at least half of the time it was shown). However, we found that two subgoals did not reach a clear consensus: Subgoals 6 and 10. We discuss each of the subgoals in detail below.

In subgoal 6, none of the labels received three or votes from participants. We discovered that participants' votes were spread out to similar, suboptimal (e.g., labels graded as L1 or L2 in Table 4.4) labels. Some participants even preferred these labels over L3 labels. This result leads to two interpretations. Even though the given label lacks explanation depth, learners could still think the level of explanation as sufficient. Second, for shorter, simpler subgoals, the variety in the labels learners create might be

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 majority (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 majority (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 majority (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 majority (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 majority (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 5.4: System-selected labels with the highest mean score among labels that received three votes or better (except for S6). We also provide the expert 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).

limited. This would lead to the system providing multiple examples that have subtle differences, making it difficult for learners to reach a consensus, and ultimately causing unwanted confusion in the system.

Subgoal 10 was considered to be most difficult to participants, since only one participant (in the baseline condition) was able to create a label scored as L3, and most of the participants did not even reach L2. Since none of the examples had a complete explanation of the code, participants might have

been more inclined to select other incomplete options, resulting in low level of agreement.

5.5 Learner Experience

	Intrinsic load (IL)	**Extraneous load (EL)**	**Germane load (GL)**
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 5.5: Mean (standard deviation) score of cognitive load.

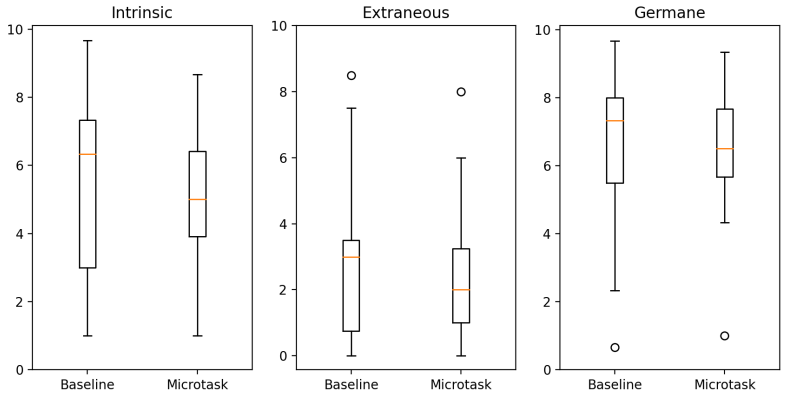

Figure 5.2: Boxplots of the cognitive load of each condition in terms of each load aspect.

The summary of the cognitive load reported by participants are shown in Table 5.5 and Figure 5.2. A Kruskal-Wallis H test revealed no significant difference in the each cognitive load measures between the conditions ($p_{IL} = 0.26$, $p_{EL} = 0.94$, $p_{GL} = 0.54$). Total time spent on the training session was also comparable between the two conditions ($p = 0.81$) – 28.87 minutes for baseline participants and 29.63 minutes for microtask participants, indicating that participants in both conditions put a similar amount of mental effort.

Participants were asked to rate the helpfulness of the provided labels and learning activities, and provide an open-ended response on how did or did not the activities helped them understand subgoals. The summary of the helpfulness is shown in Figure 5.3 and Figure 5.4. The peer examples were comparable to expert-created labels in terms of helpfulness. Overall, participants found the subgoal learning activity helpful for enhancing their understanding of the solution. By going through the tasks that center around subgoal learning, participants reported that they were more geared towards understanding the solution in terms of high-level, abstract structure, which successfully replicates the benefits of subgoal learning. However, we also discovered a sharp contrast between tasks in terms of helpfulness, which we discuss in detail below.

5.5.1 Learner Experience on the Subgoal Voting Task

Although the majority of participants (19) thought the Subgoal Voting task as being helpful in learning subgoals, the Cochran-Armitage trend test revealed a significant difference compared to other

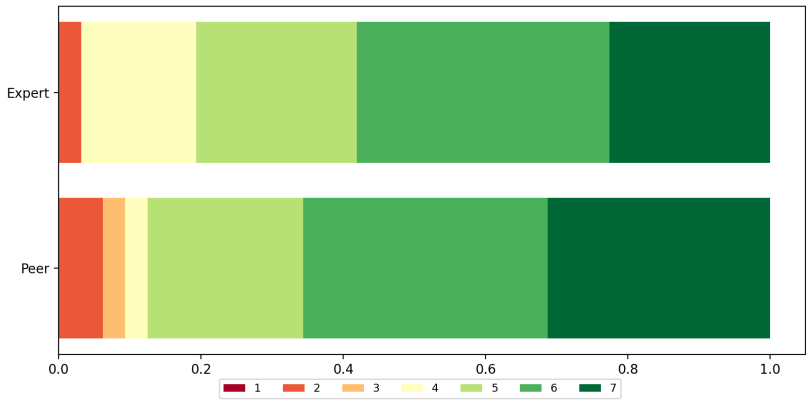

Figure 5.3: Helpfulness of the given labels; expert-created labels in baseline, peer examples in microtask (1: not helpful at all, 7: very helpful).

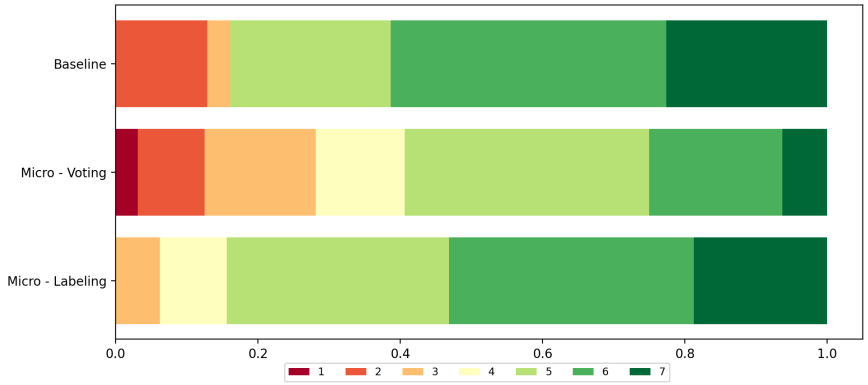

Figure 5.4: Helpfulness of the learning activities (1: not helpful at all, 7: very helpful).

tasks (baseline task: $p < 0.02$, Subgoal Labeling task: $p < 0.02$). The Subgoal Voting task acted as a scaffold for understanding the solution when novice learners struggled to comprehend the given worked example. Participants reported that the subgoal label examples acted as hints for grasping the meaning of code segments they did not understand well. 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 their understanding of the solution but rather distracted from it. When there were multiple options having the same meaning with only subtle or subjective differences, participants were confused about which option they should choose. One participant also reported that it was difficult to choose when there were options that have ambiguous meaning, and wished to see more descriptive options.

The peer consensus was generally perceived as unhelpful, mainly when the example was recently added and the consensus was not built. Since there is no consensus for recently added labels (thereby shown as '0 out of 0'), learners were confused by its meaning and did not find use of the information.

5.5.2 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 adjusting 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 short labels, and later showing more detailed, lengthier explanations.

Chapter 6. Discussion

Our learnersourcing workflow with the current microtasks were able to provide a reasonable learning experience compared to a baseline learning method with expert-created resources, while encouraging learners to provide higher-quality subgoal labels and gain a better conceptual understanding of the solution technique, thereby becoming better at transferring the obtained knowledge in their attempts to solve a new problem. In this section, we discuss the findings of our work and possible improvements in the microtask design to further boost the performance of the workflow.

6.1 Benefits of Viewing High-Quality Peer Examples

The microtasks are designed to make use of high-quality, learner-created subgoal labels as peer examples. Through a comparison against a baseline interface which does not provide such support while creating subgoal labels, we observed notable effects of the microtasks in creating better labels. In general, both microtasks improved the quality of the labels participants submitted. Even for subgoals where microtask participants did not show much difference from baseline participants, we observed cases where peer examples guided them to create more high-quality labels. Participants also noted that the examples were helpful throughout the microtasks, specifically at understanding the task at hand (i.e., the worked example), understanding good examples of subgoal labels, and fixing or improving their own labels. This result demonstrates the usefulness of peer submissions in guiding learners to create better learning contents.

We also observed notable differences between compound subgoals (S2, S7, S10), which were subgoals presumed to be most challenging to learners. Both baseline and microtask participants were unsuccessful in creating high-quality labels, but participants were able to improve their explanation quality of the labels in S7. We suspect this difference arose from different voting results in the Subgoal Voting task: If voters successfully identify and select high-quality labels, the system can in turn provide better quality labels as examples, which the learner can use the given examples as a guidance for improving their own labels. Although inconclusive, this result implies incentivizing learners to select high-quality explanations takes a key role in propagating the same level of explanation in future learner contributions.

6.2 Improving the Microtask Design

We observed a notable difference in terms of label quality between compound subgoals. Participants initially submitted similar labels quality wise, but were able to improve their labels in the final submission. We suspect that participants being able to successfully select high-quality labels in the Subgoal Voting task, which leads to better exposure of these high-quality examples in the Subgoal Labeling task, enabled learners to create better labels despite their unsuccessful initial attempt. However, we also observed that they could easily select worse explanations over high-quality examples, damaging the system's ability in choosing high-quality labels and learner performance. In future iterations, the Subgoal Voting task would have to better incentivize learners to select high-quality labels to provide better peer examples.

While both microtasks were perceived as being helpful by the majority of the participants, we observed a significant difference between the two tasks. We asked participants to select a single best ex-

ample in the Subgoal Voting task. This was to induce learners to make careful decisions on their selection rather than simply selecting all options that makes sense, and help the system quickly make distinctions between labels. However, this decision lad to unwanted confusion to both the system and learners when multiple plausible examples without significant differences were being shown to the learners. To address this issue, we could use more advanced methods for filtering similar examples.

In the Subgoal Voting task, we showed participants how many times each example was chosen over the number of occurrences to reflect peer consensus. However, participants were confused about its meaning, particularly during the early stage when no consensus was formed yet. Although a larger-scale study with a larger number of participants could yield different outcomes, the current result indicates that how to display such information should be carefully considered.

In the current design, we provide peer examples without any modifications. Although only a few participants made identical copies of previously submitted labels, there is still a high possibility of such cheating behavior. We also observed that participants showed desires for receiving the examples as hints, not final answers. The Subgoal Labeling task could be designed so that it can better serve as an intermediate guidance, such as providing frequently used terms or phrases.

Chapter 7. Limitation and Future Work

7.1 Scaling to a Larger, More Diverse Population

We investigated the usefulness of AlgoSolve with slightly more than 60 novices. However, this is a relatively small-scale evaluation compared to other research on learnersourcing and computer science education. Although the results strongly suggest that the microtasks were helpful in creating better labels and better solution plans, a larger-scale evaluation would have to follow. Also, participants came from a rather limited population of Korean university students/graduates, and were mostly not very familiar with explaining solutions in English. Further investigation on a more diverse pool of learners would strengthen the findings of our work.

7.2 Defining the Scope and Hierarchy of Subgoals

In our work, the scope and hierarchy of subgoals were determined before the sessions, decided through expert discussion. We predetermined the subgoal scope and hierarchy so that the system can solely focus on the task of gathering high-quality labels. Also, prior work has shown that making learners to both group the solution and create labels did not lead to a clear learning gain compared to doing the labeling on grouped solutions [6]. In order to develop AlgoSolve into a system that can fully operate without any expert intervention, tasks that are designed for determining the scope and hierarchy of the subgoals could be introduced in future work.

Chapter 8. Conclusion

In this work, we introduced a learnersourcing workflow that collectes high-quality subgoal labels by designing the microtasks to make use of high-quality examples, and implemented the workflow in AlgoSolve, a prototypical interface for learning subgoals for algorithmic problem-solving. Through a between-subjects study that compares AlgoSolve against a baseline interface, we observed notable improvements in quality of the submitted labels and learners' ability to create more complete plans.

Bibliography

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

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

[3] Francisco Enrique Vicente Castro and Kathi Fisler. 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*, pages 487–493, 2020.

[4] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. 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*, pages 125–180. 2001.

[5] Adrienne Decker, Lauren E Margulieux, and Briana B Morrison. Using the solo taxonomy to understand subgoal labels effect in cs1. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pages 209–217, 2019.

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

[7] Dastyni Loksa, Andrew J Ko, Will Jernigan, Alannah Oleson, Christopher J Mendez, and Margaret M Burnett. Programming, problem solving, and self-awareness: Effects of explicit guidance. In *Proceedings of the 2016 CHI conference on human factors in computing systems*, pages 1449–1461, 2016.

[8] Tia Watts. The sfc editor a graphical tool for algorithm development. *Journal of Computing Sciences in Colleges*, 20(2):73–85, 2004.

[9] Martin C Carlisle, Terry A Wilson, Jeffrey W Humphries, and Steven M Hadfield. Raptor: a visual programming environment for teaching algorithmic problem solving. *Acm Sigcse Bulletin*, 37(1):176–180, 2005.

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

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

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

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

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

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

[16] Lauren E Margulieux, Mark Guzdial, and Richard Catrambone. 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*, pages 71–78, 2012.

[17] Lauren E Margulieux, Briana B Morrison, Baker Franke, and Harivololona Ramilison. Effect of implementing subgoals in code. org's intro to programming unit in computer science principles. *ACM Transactions on Computing Education (TOCE)*, 20(4):1–24, 2020.

[18] Robert K Atkinson, Richard Catrambone, and Mary Margaret Merrill. Aiding transfer in statistics: Examining the use of conceptually oriented equations and elaborations during subgoal learning. *Journal of Educational Psychology*, 95(4):762, 2003.

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

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

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

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

[23] Sarah Weir, Juho Kim, Krzysztof Z Gajos, and Robert C Miller. Learnersourcing subgoal labels for how-to videos. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 405–416, 2015.

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

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

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

[27] Hyoungwook Jin, Minsuk Chang, and Juho Kim. 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*, pages 1–6, 2019.

[28] Jacob Whitehill and Margo Seltzer. A crowdsourcing approach to collecting tutorial videos–toward personalized learning-at-scale. In *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale*, pages 157–160, 2017.

[29] Philip J Guo, Julia M Markel, and Xiong Zhang. 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*, pages 301–304, 2020.

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

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

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

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

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

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

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

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

Acknowledgment

I would first like to thank my advisor, Juho Kim, for guiding me to grow up as a researcher. I would like to thank my awesome collaborators, Minsuk Chang, Sally Chen, Hyoungwook Jin, Hyungyu Shin, and Meng Xia, for their tremendous help throughout this work. I also thank my colleagues in KIXLAB for their valuable comments.

This work was deeply motivated by my personal experience in learning programming and algorithms. I was very lucky to get to learn programming and algorithmic problem-solving where almost no one knew how to program. I want to thank Heondae Kim, who led me to the world of programming. Finally, I am extremely grateful to my family for their unconditional love and support.

Curriculum Vitae in Korean

이 름: 최 갑 도

생 년 월 일: 1996년 12월 23일

학 력

2012. 3. – 2014. 2. 경남과학고등학교

2014. 3. – 2019. 2. 한국과학기술원 전산학부 (학사)

2019. 3. – 2021. 8. 한국과학기술원 전산학부 (석사)

경 력

2019. 3. – 2020. 8. 한국과학기술원 전산학부 조교

연 구 업 적

1. **Kabdo Choi**, Sally Chen, Hyungyu Shin, Jinho Son, and Juho Kim, "AlgoPlan: Supporting Planning in Algorithmic Problem-Solving with Subgoal Diagrams", In *Proceedings of the Seventh ACM Conference on Learning @ Scale*.