AlgoPlan: Supporting Planning in Algorithmic Problem-Solving with Subgoal Diagrams

Kabdo Choi[1] **Sally Chen**[2] **Hyungyu Shin**[1] **Jinho Son**[3] **Juho Kim**[1]

[1]KAIST, Daejeon, Republic of Korea
[2]Brown University, Providence, RI, USA
[3]Algorithm LABS, Seoul, Republic of Korea
{kabdo.choi, hyungyu.sh, juhokim}@kaist.ac.kr sally_chen@brown.edu sjhfam@algorithmlabs.co.kr

ABSTRACT

Planning a solution before writing code is essential in algorithmic problem-solving. However, novices often skip planning and jump straight into coding. Even if they set up a plan, some do not connect to their plan when writing code. Learners solving algorithmic problems often struggle with high-level components such as solution techniques and sub-problems, but existing representations that guide learners in planning, such as flowcharts, focus on presenting lower-level details. We use subgoal diagrams – diagrams made of subgoal labels and the relationships between them – as a representation that guides learners to focus on high-level plans when they develop solutions. We introduce AlgoPlan, an interface that enables learners to build their own subgoal diagram and use it to guide their problem-solving process. A preliminary study with seven students shows that subgoal diagrams help learners focus on high-level plans and connect these plans to their code.

Author Keywords

Algorithmic problem-solving; planning; subgoal diagram;

CCS Concepts

•**Human-centered computing** → *Human computer interaction (HCI);* •**Applied computing** → *Computer-assisted instruction;*

INTRODUCTION

Algorithmic problem-solving [1] is a complex activity that involves multiple stages, including problem comprehension, solution planning, implementation, and testing [8]. Since algorithmic problems often have tight time and space constraints, learners have to develop efficient solutions to address the given constraints. Meanwhile, novices often skip the planning stage and go straight into coding [9]. Moreover, some students, despite setting up a solution plan, do not connect their plan to code and fail to solve the problem [3].

To guide novices in planning a solution, researchers have proposed various representations, such as hierarchical plan decompositions [5], flowcharts [2, 12], and data flow networks [6]. These capture low-level details such as a line of code or simple operations. However, learners solving algorithmic problems mainly struggle from high-level components such as solution techniques and sub-problems [14], which cannot be well captured in these low-level details.

We designed a representation that shows these high-level components using subgoals, which can promote learner's understanding of the solution structure [4]. We propose *subgoal diagrams*—diagrams made of subgoal labels and the relationships between them (Figure 1). By formulating their solution ideas into subgoal diagrams, learners can focus on the high-level structure of the solution. Subgoal diagrams also act as an intermediate representation that bridges plan and code.

We introduce AlgoPlan ('algo' means 'know' in Korean), a problem-solving interface that supports solution planning using subgoal diagrams. AlgoPlan supports learners through a three-stage problem-solving process: 1) Diagram Generation, where learners build their own subgoal diagram, 2) Diagram Comparison, where learners compare their diagram with a reference diagram and improve theirs, and 3) Implementation, where learners write code while referring to their diagram.

In a preliminary study with seven students who are learning algorithms, participants found subgoal diagrams useful for planning a solution by making them think in terms of subtasks and the algorithmic flow. They were also able to connect these plans with code, which led to an easier coding experience.

Our contributions are as follows:

- Subgoal diagram, a representation that guides learners to focus on high-level plans when developing solutions
- AlgoPlan, an interface that supports algorithmic problem-solving using subgoal diagrams
- Results from a preliminary study showing the usefulness of subgoal diagrams in algorithmic problem-solving

BACKGROUND

We review prior research on representations that aim to support planning and how novices work with plans.

Representations for Supporting Planning

Researchers have developed representations that could help learners design programs. A common approach is the use of flowcharts. RAPTOR [2] is a visual programming environment that uses flowcharts to help learners design and execute

L@S '20, August 12–14, 2020, Virtual Event, USA.

ACM ISBN 978-1-4503-7951-9/20/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/3386527.3406750

algorithms without syntax issues. Smetsers-Weeda and Smetsers [12] suggested that flowcharts enable students to brainstorm and plan solutions before diving into code. However, flowcharts still show code-level details by representing each line of code, which makes it difficult to understand high-level plans as solutions become more complex.

Another line of research focuses on representing plans to guide the problem-solving process. Guzdial et al. [5] introduced GPCeditor, a tool that scaffolds the process where students decompose the problem into goals and plans and compose plans into a complete program. However, students were required to fully decompose the problem to code-level regardless of expertise, which made them feel constrained. Also, students did not receive any feedback on their decomposition, resulting in low-quality products and ultimately diminished benefits. Hu et al. [6] developed a representation using a network of goals and plans to guide the process. However, the goals and plans were still written in terms of simple operations (e.g., add, divide), which could be difficult for students to grasp high-level components for complex solutions. Our design, however, involves a representation that focuses on high-level plans rather than code- or operation-level details.

Novice Behaviors When Working with Plans

Multiple studies reported that novices, in contrast to experts, do not put much effort into planning solutions [3, 9]. Some skip the planning stage entirely [9] or identify tasks (i.e., high-level plans) on-the-fly as they code [3], resulting in unsuccessful problem-solving. Castro and Fisler [3] further investigated how novices move between tasks and code, and found that students who clearly described tasks and connecting them with code showed most success in solving problems. Our system aims to guide learners to plan before coding, and make connections between their plans and code.

The level of detail when planning solutions also depends on expertise. Rist [11] noted that the degree a learner can plan ahead differs between novices and experts. As expertise develops, learners can more easily retrieve larger plans and can design larger chunks of the solution at once. We design our representation to capture multiple levels of subgoals so that it can support learners with varying planning ability.

DESIGN GOALS

We list three design goals for supporting planning in algorithmic problem-solving process.

- **G1**: Match the learner's expertise on planning solutions.
- **G2**: Connect plans with code to guide learners during implementation.
- **G3**: Provide feedback on their solution plans.

SUBGOAL DIAGRAM

We introduce subgoal diagram (Figure 1), a representation with which learners distill their solution ideas into high-level plans and use it to make connections between plans and code.

Each node contains a subgoal label describing the functional component of the solution. Nodes have different colors, which give visual cues of the connection between the diagram and the code editor (Figure 3). The ordering of the node sequence implies the program execution path. This consistency enables learners to consider a program flow when planning solutions, so that they can easily connect their plans with code (**G2**).

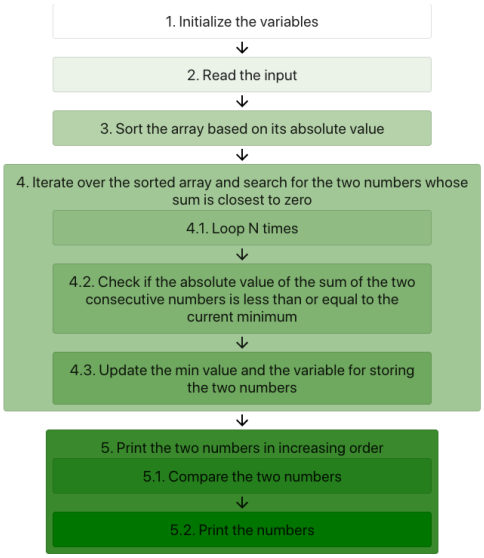

Figure 1. An example reference subgoal diagram. Subgoal diagrams guide learners in planning by encouraging them to focus on high-level plans.

Subgoal diagrams contain multiple levels of detail, spanning from the high-level, abstract subgoals to lowest-level implementation details. The hierarchy within subgoals is shown as subdiagrams. Learners face only the top-level nodes first, and can optionally expand the nodes (Figure 2) to see lower-level plans. This is to accommodate a wider range of learners with varying expertise in planning abilities (**G1**) while still maintaining focus on the high-level structure, since showing all details at once could overwhelm learners.

SYSTEM OVERVIEW

We provide a walk-through of AlgoPlan, a problem-solving interface that guides the learner's problem-solving process using subgoal diagrams: Learners first build their own diagram (Diagram Generation), improve it by comparing against a reference diagram (Diagram Comparison), and then write code by connecting it with the diagram (Implementation).

Diagram Generation

The *skeleton subgoal diagram* (Figure 2) is provided alongside the problem text. The skeleton subgoal diagram is a replica of a *reference subgoal diagram* – a diagram that shows a correct solution – but with incomplete subgoal labels. The first words of each subgoal, which are verbs, are provided in the skeleton, enabling learners to get a sense of how the solution is being structured and get useful hints in planning their solution.

Diagram Comparison

As learners complete their diagram, the reference subgoal diagram is shown next to their initial diagram. As learners compare the two diagrams, they can improve theirs by editing their subgoal labels (**G3**).

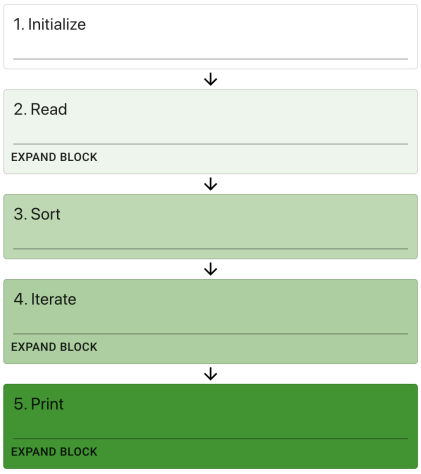

Figure 2. An example skeleton subgoal diagram. Learners structure their solution ideas into diagrams using the skeleton subgoal diagrams.

Implementation

Right after improving their diagram, learners proceed to implementation. The final diagram and the code editor are shown side-by-side (Figure 3). The subgoal labels in the diagram are shown as code comments in the editor. The two panes are connected through *block gutters*, which are colored blocks located at the left side of each line that matches the color of the corresponding diagram node. These together support learners in connecting their plan with code and guide their implementation (**G2**).

EVALUATION

We conducted a preliminary study to evaluate the effectiveness of subgoal diagrams in guiding learners to structure their solution using high-level plans.

Participants

We recruited seven undergraduate and graduate students (male: 6, female: 1) who are currently learning algorithms. Their learning experiences varied from self-learning (e.g., online judge sites, MOOCs) to offline lectures. The level of expertise in problem-solving also varied from a casual learner who solves problems as a hobby to an expert learner who has participated in several programming contests.

Study Design and Materials

Participants solved one algorithmic problem using AlgoPlan. After solving the problem, participants rated the usability of the system through a five-point Likert scale (1: strongly disagree, 5: strongly agree) and open-ended questions.

We selected the problem from the Korea Olympiad in Informatics to ensure adequate difficulty. The reference subgoal diagram was made by the first author by manually inspecting three correct code examples and identifying their subgoals.

Results

All except one participant reported a positive experience using the system. Overall, they found subgoal diagrams useful for structuring their plans and writing code.

Support on planning

Participants were able to easily formulate ideas into a diagram structure (average: 3.71) and found the skeleton diagram useful for planning a solution (average: 4.00). Participants noted that the diagram structure helped by guiding them to think in a step-by-step manner and consider the algorithm flow.

However, participants also mentioned that the system lacked flexibility when thinking of different solutions, forcing them to think of a specific solution. One participant noted that the provided words can act as hints, and wished not to see them for their learning purpose.

Comparison

Participants found the reference diagram useful for improving their plans (average: 4.00). All except one participant improved their diagram. Three participants completely changed their diagram, after realizing that their solution was more complex than the reference diagram. One participant missed important conditions in their initial diagram, but managed to fix the issues by comparing with the reference diagram.

Support on coding

Participants found subgoal diagrams useful for implementing their solution (average: 4.14). The diagram acted as an outline, helping them remember which components to implement. Some reported they had already gained a clear mental structure of the program by building the diagrams and had no difficulty writing code.

However, participants showed mixed impressions on the usefulness of block gutters (average: 3.29). Several participants did not make use of the gutters at all by not using the code comments. One participant, who noted that they didn't utilize the gutters well, still found it useful for knowing their implementation progress at a glance.

CONCLUSION AND FUTURE WORK

We introduced subgoal diagrams, a representation that guides learners in planning solutions by making them focus on high-level plans, and designed an interface that enables learners to solve algorithmic problems with subgoal diagrams. Results from a preliminary study show that the system helps problem-solving by enabling users to structure their solution ideas into high-level plans and use it to guide their implementations.

Immediate future work is to further evaluate the benefits of subgoal diagrams and AlgoPlan. We plan to run a large-scale study comparing subgoal diagrams against other representations that can support algorithmic problem-solving.

As learners continue to use our system, we can gather the mappings between subgoal diagram nodes and corresponding code snippets. These resources can then be applied as valuable feedback to the learner's implementation, such as providing good code examples for specific parts of the program. We plan to extend our system so that it can cover stages beyond learners' problem solving such as reflection, since reflecting on their solutions can enrich their learning experiences [10].

The authoring cost of subgoal diagrams is an important factor for the scalability of the approach. We are currently working

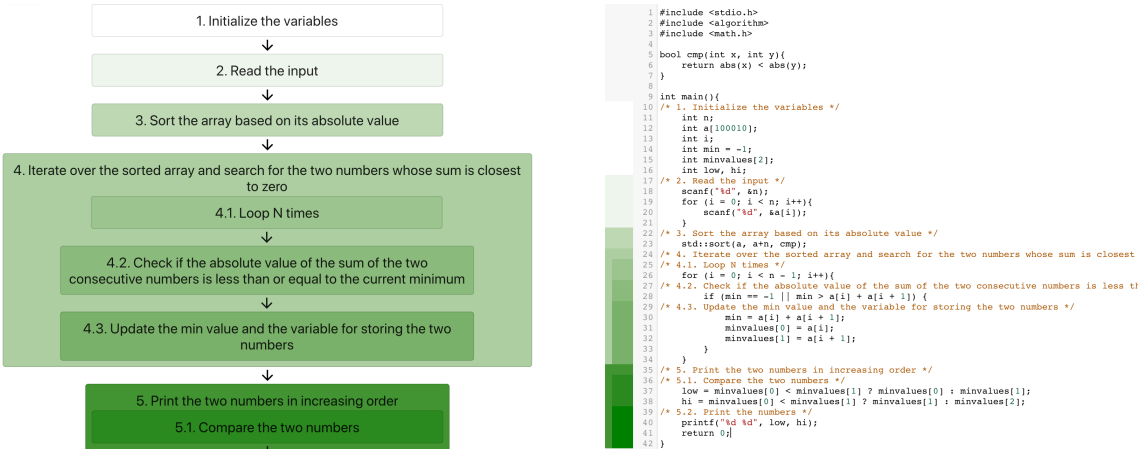

Figure 3. Layout of AlgoPlan in the Implementation stage. The subgoal diagram and the code editor are shown side-by-side, where block gutters, attached to code comments showing subgoal labels, connects the two representations.

on replacing the authoring process of subgoal diagrams with a learnersourcing workflow for labeling subgoals [7, 13], so that learners can have a meaningful learning experience while little to no expert effort is required for authoring.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] Dastyni Loksa, Amy J Ko, Will Jernigan, Alannah Oleson, Christopher J Mendez, and Margaret M Burnett. 2016. Programming, problem solving, and self-awareness: effects of explicit guidance. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 1449–1461.

[9] 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.

[10] Thomas W Price, Joseph Jay Williams, Jaemarie Solyst, and Samiha Marwan. 2020. Engaging Students with Instructor Solutions in Online Programming Homework. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–7.

[11] 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.

[12] Renske Smetsers-Weeda and Sjaak Smetsers. 2017. Problem solving and algorithmic development with flowcharts. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*. 25–34.

[13] 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.

[14] Shamama Zehra, Aishwarya Ramanathan, Larry Yueli Zhang, and Daniel Zingaro. 2018. Student misconceptions of dynamic programming. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 556–561.