Wireless Pers Commun (2017) 93:131–145
DOI 10.1007/s11277-016-3679-9

Puzzle Based Algorithm Learning for Cultivating Computational Thinking

Jeongwon Choi[1] · **Youngjun Lee**[1] · **Eunkyoung Lee**[2]

Published online: 7 September 2016

Abstract Computational thinking (CT), which is the core of the Computer Science field, is an essential thinking process to solve problems effectively and efficiently using computing systems. Learners must be able to design algorithms, identify the appropriate algorithm design skill for a specific problem, and apply it to the problem. Aiming to stimulate learners' interest in learning algorithm design skills, we developed puzzle-based algorithm learning program that has a user-friendly format tailored to real-world scenario. We investigated the effect of this puzzle-based algorithm learning program on learners' CT abilities. The results provide evidence that puzzle based algorithm learning program is effective for developing learners' CT. The study suggests that puzzle based algorithm learning is worth as a learning model for improving CT of learners.

Keywords Computational thinking · Algorithm learning · Puzzle based learning · Design based research

✉ Eunkyoung Lee
eklee76@kice.re.kr

Jeongwon Choi
cjw0829@daum.net

Youngjun Lee
yjlee@knue.ac.kr

[1] Department of Computer Education, Korea National University of Education, Cheongju, Republic of Korea

[2] Center for Teacher Selection Tests, Korea Institute for Curriculum and Evaluation, Seoul, Republic of Korea

1 Introduction

Rather than simply acquiring knowledge and skills, twenty first century learners should develop their abilities to apply the gained knowledge and skills into solving real world problems. Computer Science (CS) has been pivotal in developing the problem-solving skills and identifying tools to resolve real-world problems [1–3].

Computational thinking (CT) is an essential competency for twenty first century learners, and which is defined as a problem solving skills necessary for designing algorithms for solving complex problems effectively and efficiently based on fundamental CS concepts, principles, and perspectives [4]. CT consists of abstraction and automation. Automation means process representing in a form that computing system can performs using a programming language or software. Abstraction refers to the logical process of designing algorithms and is vital, since only well-designed algorithms can generate solutions efficiently with a minimum of trial and error. It is similar to the process of drawing accurate building plans, which includes planning for material and work stages as well as the final architectural design, and avoiding any possible mistakes, to build a perfect building.

Generally, algorithm design skills are used for problem solving; to design effective and efficient algorithms, learners should have opportunities to use various algorithm design skills to solve problems. However, the current algorithm learning programs in schools focus on imparting knowledge on general algorithms, rather than applying the various algorithm design skills to solving real problems. In fact, many of the current algorithm learning programs require learners to memorize algorithms and experess them as pseudo codes. They also require learners to learn representative algorithm design skills such as sorting and searching [5–7].

Occasionally, in the process of learning CT, learners should memorize algorithms frequently used and applying the appropriate algorithm depending on the type of problem [7]. In this learning process, learners are most likely to perceive mastering algorithm is boring, difficult, and even unnecessary [5, 7]. In addition, as noted previously, many of the current algorithm learning programs do not allow learners to apply a specific type of algorithm design skill to solving a specific real problem. Hence, based on typical learning programs, simply learning algorithms does not ensure that learners are improving their performance in CT. An effective algorithm learning program enables learners to select an algorithm design skill that can efficiently solve real-problems, as well as equipping the learners with the knowledge to make algorithms using the algorithm design skills.

Puzzles can be used to improve the effects of algorithm learning when learning programs are designed to teach students how to apply algorithm design skills to solve problems. Puzzles can motivate learners to learn algorithms a variety of real-world problem that are fun to solve, and help learners focus on problem solving processes because the given situations are connected with their real-life as well as fun and simple. In addition, fun and simple problem situations help learners easily understand problem situations and their solutions. Further it helps them apply these to other problem situations [8, 9]. Simply put, using puzzles, learners can practice how to apply the appropriate algorithm design skills to solve problems; this shows the value of using puzzles in algorithm learning.

While many studies have been devoted to exploring perceptions or experiences of students who were involved in progressive learning programs in CT such as puzzle-based algorithm learning, relatively fewer studies measured its effects on student performance in

CT. In response, we investigated the effect of puzzle-based algorithm learning on student ability level in CT.

2 Related Work

2.1 Computational Thinking

The twenty first century society defines a talented person as a creative person who uses higher-order thinking skills, including critical thinking, and logical thinking skills, to solve problems, and not as someone who merely has a vast amount of knowledge [10, 11]. CS has played a pivotal role in societal development. The core capacity of CS is CT, which involves logical thinking skills to solve problems effectively and efficiently using algorithms designed based on the concepts and principles of CS and allowing the algorithms to be executed by computer systems [4]. Wing [12] stated that CT can be compared to thinking like a computer scientist when facing problems; however, it is a fundemental thinking skill for everyone, not limited to computer scientists, and involves capacities beyond programming skills. According to the author, CT helps solve problems, design systems, and understand human behaviors based on the fundamental concepts of CS [12–14].

Though the term "CT" was first discussed by Wing in 2006, we had already been using it for problem solving. In addition, it originated in the CS field; however, the demand for CT began to emerge from various domains, and the importance of fostering talents who could solve problems quickly and accurately through communication between human beings and computing systems using CT is being emphasized. Hence, CT is an essential and fundamental ability for everyone, and not only for computer scientists. Starting with the argument by Perlis [15] that not only computer scientists but also university students must have CT abilities, Papert also stated CT education should be expanded to include young students. Since then, more emphasis has been placed on CT education and more programs have been developed, as CT is perceived as a fundamental skill for anyone living in the twenty first century [16–18].

CT consists of abstraction and automation [14]. Abstraction refers to the process of analyzing problems and designing algorithms; automation is defined as the process of expressing designed algorithms that computing systems can execute. Particularly, abstraction undergoes complicated stages, including defining the current status of a problem and the future goal, collecting and analyzing necessary information for problem solving, exploring various problem solving strategies, and selecting the most appropriate one in the process of algorithm designing. Therefore, learners cannot improve their CT abilities only by studying algorithm design skills. Effective algorithm learning requires not only being familiar with algorithm design skills, but also practicing with actual problems and applying appropriate skills to solve them.

However, so many studies for improving CT are focused on automation represented programming. Learning of concepts and principles of programming like selection of programming instructions, and implementing control structures of sequence, iteration, conditional branch, and debugging are concentrated in learning programs [19–23]. Sometimes, abstraction is simply adrressed at a level of representing briefly algorithms [24]. Abstraction is critical to understanding clearly the problem and to design problem

solving method and process. It is inevitable that learning focused on automation is half-learning without abstraction.

Therefore, in this study we designed puzzle based algorithm learning (PBAL) focused on abstraction and examined effect of PBAL for cultivating CT.

2.2 Algorithm Learning

A CS learner receives related education on various skills to design algorithms. The representative skills include a variety of sorting skills that list problems in order, and search skills that search information desired efficiently among varied data.

Algorithm design skills are not essential elements to solve problems; however, the abstraction process of CT directly influences the efficiency of problem solving. Specifically, the amount of knowledge in algorithm design skills play a critical role in determining the degree of effectiveness in problem solving processes and results.

However, as addressed earlier, the most current algorithm learning programs tend to focus on understanding the concepts and principles of algorithm design methods, and acquiring searching or sorting skills instead of exercises to solve a problem by selecting an appropriate algorithm design skill. Further, at the current stage, algorithm learning involves memorizing the process of expressing algorithm design skills in a programming language [5–7].

Some educational institutions teach the principles of algorithm design skills based on competition-oriented learning, such as the Informatics Olympiad, which asks students to compete against each other for coding algorithms in a programming language and practicing what algorithm design skills should be applied to a certain type of problem [25, 26]. This educational style typically causes learners to perceive studying algorithms is challenging, which in turn leads them to avoid algorithm learning [7]. It also affects how learners are aware of familiar or unfamiliar problems; they tend to face challenges when encountering new kinds of problems while finding it easy to solve familiar ones.

The important criterion for algorithm learning to improve CT is not related to whether learners precisely understand algorithm design skills. Simply knowing about algorithm design skills does not guarantee effective or efficient problem solving; it is only possible when learned skills can be applied to actual problem solving. Therefore, learning should enable learners to acquire capacities to apply algorithm design skills to solve problems.

2.3 Puzzle Based Learning

Aiming to promote learners to apply algorithm design skills learned from classrooms to solve real-world problems, we adopted puzzles for algorithm learning. Providing experiences of solving real-world problems could be an appropriate learning method; however, there are a number of variables to consider for learners dealing with real-world problems; these variables change constantly, thus requiring significant time to solve them as well as define them. Comparatively, puzzles have advantages, allowing learners to focus on problem solving processes, as problem situations are simple and structured to some extent [9]. Puzzles are appropriate for offering various problem-solving experiences for algorithm learning to improve learners' CT.

Puzzles generate cognitive satisfaction in the process of finding solutions through play. Problem solving experiences accumulated from puzzle solving help learners acquire knowledge about how to design a problem solving process, including defining a problem, searching for the right solution, and applying it [7, 8]. This study considers puzzles as an

effective algorithm learning tool because they can improve learners' problem solving skills, logical thinking skills, and creative thinking skills [9, 27, 28]. Puzzles have the following important characteristics.

First, puzzles are characterized by independence. Puzzle problems are not restricted to certain domains; a variety of problems can be tested, as puzzle solving does not require specific knowledge about certain domains.

Second, puzzles have generality. This means that puzzles involve general problem solving principles. Problem solving skills are acquired when solving problems, which helps develop appropriate problem solving strategies and principles. In addition, generality explains the characteristic that learned problem-solving principles could be applied to solve new problems or future problems.

Third, puzzles have simplicity. They can expand the application of problem solving strategies, and make learners easily recall and explain a problem solving process because they make it easy to remember a problem itself as well as the problem solving process.

Fourth, puzzles have the "eureka factor." This means that learners can apply the previously acquired problem solving skills intuitionally in the process of problem solving. This might present difficulties in problem solving and lead to frustration; however, it also allows learners to discover new skills and stimulates their interest if they overcome their frustration and discover new insights. Thus, a thorough exploration of problem solving processes leads to new insights; this is what Martin Gardner calls the "Eureka Phenomenon." The moment of realization serves as learners' compensation; it also motivates learners to solve other problems.

Fifth, puzzles have the "entertainment factor." Puzzles should be interesting; otherwise, motivation for problem solving will be diminished.

The characteristic of generality makes algorithm design skills a strategic instrument to solve puzzle-based problems. Simplicity implies that puzzles improve the effects of learning more by making the memorization of algorithm design skills easy for learners. That puzzles have the eureka factor indicates that learners can systematically and logically plan a problem solving process. Puzzles' entertainment factor means that learners can be continuously motivated to learn and improve CT. Therefore, puzzles are valuable resources for algorithm learning to improve CT.

Many case studies that proved various learning effects of puzzles in CS classes also showed that puzzles were useful instruments. The studies revealed that puzzles used in overview classes targeting CS major students stimulated the students' interest and improved their participation in the programs [29, 30]. In addition, puzzles helped understand algorithm design skills such as brute force, decrease-and-conquer, divide-and-conquer, and transform-and-conquer, as well as less general strategies regardless of the programing language; thus, puzzles can be used in various real-world situations, not limited to the field of CS [31]. Until recently, puzzles were mostly adopted in programs for university students; however, they are being integrated into programs for young students in elementary and secondary school. It was suggested that young learners could also focus on abstraction of CT when the mathematical knowledge required for puzzle problem solving was at the appropriate level for learners [32]. However, teachers need strategies to intervene at the appropriate moment for scaffolding and providing feedback, since puzzle problem solving requires an open approach, and it could pressure young students' cognitive processes [33].

3 Research Methods

3.1 Design Based Research

Design based research (DBR) uses natural educational settings, in which study methods or programs created by researchers are implemented and tested iteratively. Its continuous cycle of investigating results and improving researchers' models contributes to addressing challenges, improving educational effects, and expanding the application of those models [34, 35].

Prior experimental studies measured effects by devising a hypothesis in a controlled environment, and controlling certain variables; however, the results of experimental studies in education tended to have different educational effects when implemented in complex real-world situations; thus, the generalization had limitations. In contrast, DBR verifies the effects of changing the learning environment and helps overcome the disadvantage of experimental studies. In addition, DBR provides results that can be generalized, as it involves analyzing results to constantly improve the models being tested, and re-implements them in the learning environment [36, 37].

DBR is a combination of qualitative analysis and quantitative analysis, which allows the improvement of education. It also comprises core elements such as a research design, a theory, a problem, and a naturalistic context, which are inter-connected. It helps develop a learning theory that can be generalized and implemented in a complex educational environment by designing a theory about a research problem and repeatedly improving it in naturalistic contexts.

3.2 Participants and Settings

The study designed PBAL and studied 82 Informatics-gifted South Korean students for 48 h (an hour per a session) to verify improvements in learners' CT. These students are in the 4 to 6th grade; 44 participated in an experimental group and 38 participated in a control group. As background, the Republic of Korea has been focusing on fostering talented students who excel in CS for the future of society, because such gifted students are more likely to suggest innovative ideas based on CT and recreate them in practice with their creativity, task commitment, sharp observation and analytical skills, reasoning, and problem solving skills [38, 39].

Algorithm education for Informatics-gifted students has not provided opportunities to learn in fun ways, although these students are accomplished at understanding and learning quickly. It has been proposed that explanations related to real-life scenarios should be provided to improve the situation; however, in most classes, examples of applying certain algorithm design skills to certain problem situations are provided and explained by teachers [40].

The designed PBAL was implemented in the experimental group. The group was asked to solve problems, with 15 min being allocated for each puzzle problem, and the results of the algorithm were verified by a teacher. Learners could explain the results of the algorithm in their preferred way, either through a written explanation or a diagram. The traditional algorithm learning method was implemented in the controlled group. The learners in the group were taught about the concepts or principles of algorithms, and tested to see whether they were able to express them.

The study designed PBAL for improving learners' CT, repeated a process of modifying PBAL three times, and completed a PBAL that reflected aspects for improvement. Then we measured learners' CT improvement. To design a sophisticated PBAL and observe learners' problem solving processes closely, two associate teachers with previous teaching experience were recruited to write field notes, and interview target students about difficulties during the learning process.

3.3 Data Collection and Analysis

The study modified and complemented a CT measurement instrument created by Lee [41] to design a PBAL and measure the level of improvement in the learner's CT. It meets validity requirements with three professional reviewers. The self-report instrument developed by Lee [41] has open-ended questions containing problem situations related to real-life matters to which learners can apply algorithm design skills. It is the evaluation instrument that can examine overall problem solving processes including whether learners select and apply proper algorithm design skills to given problems. Regarding the evaluation of self-reports, two points are awarded when a learner designs an algorithm based on the correct skill for a problem and arrives the correct solution to the problem. One point is awarded when the algorithm is not complete and part of the process is missing but the correct technique and solution are found. For any other scenario, no points are awarded. The self-report instrument has six questionnaires.

4 Results

Table 1 shows the overview of PBAL workshops designed iteratively by DBR methodology. In the workshop 1, researchers designed and implemented the PBAL during 8 sessions with 15 puzzles. For the workshop 2, the PBAL was refined based on the reflections of workshop1 and implemented during 20 sessions with 27 puzzles. Finally, the PBAL was completed and implemented during 20 sessions with 22 puzzles in the workshop 3.

4.1 Workshop 1: PBAL Design

In Workshop 1, 15 puzzles are chosen so that learners can apply algorithm design skills. Puzzles are given to participants eight times and the PBAL is examined to check whether it can improve learners' CT. The self-reports contain open-ended questions to investigate learners' problem solving processes. The questionnaires in Workshop 1 contain basic algorithm design skills, such as loop and branch, as well as a variety of other algorithm design skills, including sorting, backtracking, minimum spanning tree, dynamic programming, recursion, divide-and-conquer scheduling.

For strategies to solve puzzles, 3 stages of puzzle problem solving suggested by Michalewicz and Michalewicz [8] were taught to participants: extracting the core elements of problems, removing intuition, and modeling. The first stage of extracting core elements involves understanding the core purpose of solving puzzles. The second stage is to review whether intuition is involved or can be avoided, and keeping learners from solving problems rather quickly based on mere experience or speculation. The third stage of modeling is to explore the core elements of puzzles, plan problem solving processes, and

Table 1 Overview of PBAL workshops

	Workshop 1	Workshop 2	Workshop 3
Overview	8 sessions 15 puzzles	20 sessions 27 puzzles	20 sessions 22 puzzles
Problem solving strategies	Do not rely on your intuition too much Abstraction Modeling	Do not rely on your intuition too much Abstraction Modeling Review and correction	Do not rely on your intuition too much Set problem solving goals Analysis on information and constraints Abstraction Modeling Review and correction Represent data as Table, Chart, Graph Problem decomposition Pattern recognition
Algorithm design skills	Sorting Backtracking Minimum spanning tree Divide-and-conquer Scheduling Recursion Dynamic programming	Sorting Backtracking Minimum spanning tree Divide-and-conquer Scheduling Recursion Dynamic programming Decrease-and-conquer Brute-force algorithm	Sorting Backtracking Minimum spanning tree Divide-and-conquer Scheduling Recursion Dynamic programming Decrease-and-conquer Brute-force algorithm Binary search Encryption algorithm Deadlock solving Greedy algorithm
Feedback and Scaffolding	Explain problem solving process and correct answer after learners' problem solving Encourage learners themselves reviewed their problem solving and error correction	Explain problem solving process and correct answer after learners' problem solving Respond to questions from students Observe learners' problem solving process and Provide appropriate scaffoldings	Guide the reasons why learners studied algorithms and solved puzzles Respond to questions from students Observe learners' problem solving process and Provide appropriate scaffoldings Guide to solve the problem through communication with teacher and whole class
Learning materials	Problem and blank paper for answer	Problem and paper for answer with question based on 4 problem solving strategies	Problem and paper for answer with question based on 9 problem solving strategies

execute them for identifying answers. The three strategies to solve puzzles allow learners to systematically solve problems, as they can consciously consider problem-solving processes with informed problem solving strategies. Learners are asked to write the process of problem solving within a given time. Then, teachers explain the correct problem solving method and process; learners are requested to review their processes and answers, and modify errors if any.

The results of Workshop 1 show that learners are willing to solve puzzles while their achievement level is low; therefore, we analyzed learners' problem solving processes and found what their approaches were and what knowledge was needed to solve problems. The self-reports of learners were divided into two: those who respond with incorrect answers and those who respond with correct answers. Based on the incorrect answers and learners' approaches and strategies, the PBAL was modified as below.

First, the problem situations were modified to accommodate the level of understanding of learners. Those who gave incorrect answers did not have a good understanding about the concept of specific terms and could not interpret unfamiliar terms, purposes of questions, and statements provided. Therefore, three elementary school teachers reviewed the questionnaires and modified difficult sentences or expressions.

Second, proper feedback and scaffolding were provided when learners experienced difficulties. Those who gave incorrect answers did not accurately comprehend the appropriate algorithm design skills and principles, or could not apply them to the problems. In this case, learning can be improved when appropriate feedback or scaffolding is provided to help learners completely understand algorithm design skills and principles. In contrast, it is difficult to plan in advance when certain feedback or scaffolding should be given, since every learner is on a different cognitive level; therefore, we offered questions that can enlighten learners about potentially effective and necessary strategies, clues, and information, instead of providing direct approaches to solutions. Some examples of questions are as follows: "Would you please look for every piece of information you can discern in the problem situation of the puzzles?", "Do you see any missing information?", "What can we do to understand the material given more easily?", and "Would you like to draw a picture?"

Third, the three problem solving strategies in Workshop 1 were expanded to four stages including "Review and Modification." The three strategies in Workshop 1 resulted in the end of learning, as students were forced to stop without a review of their problem solving process when they could not find a solution or move onto the next question. This can result in overlooking possible errors in each individual's problem solving process, including ones that happen when relying on intuition or understanding terms incorrectly. Therefore, the "Review and Modification" strategy contributes to successful problem solving by correcting an incorrect problem solving process through reflective thinking that looks back on where certain errors have occurred and why they have appeared.

Fourth, four problem solving strategies were added as sub-items to self-reports to explore learners' overall problem solving processes. Though the self-reports provided in Workshop 1 had enough space to write algorithms, learners failed to use the space effectively. Learners were observed to use the space to draw graphs, charts, or tables to overcome difficulties in the process of problem solving, or to write down arithmetic operations instead of using the space to describe the details of problem solving processes. Optimally using self-reports offers convenience in reviewing problem solving processes and modifying errors for learners. It will also provide opportunities for researchers to analyze learners' problem solving processes.

Fifth, puzzle questions were added to provide enough experiences of problem solving to learners, as compared to Workshop 1. To control the level of difficulty, puzzles that were too abstract or difficult were removed; those puzzles that have too structured problem situations or direct clues to problem solving were also eliminated. Some puzzles had easy questions involving the extraction of core elements to solve problems or specific questions to hone logical judgment skilss when intuition distracts learners; these were different from

those questions that were developed to apply algorithm design skills only. The added puzzle questions had algorithm design skills, including brute-force and decrease-conquer.

4.2 Workshop 2: Refining PBAL

Workshop 2 was performed for 20 h in total. In Workshop 2, 27 puzzles were selected and developed, reflecting the points of improvement found in Workshop 1. Since there is enough time in Workshop 2, more puzzles than Workshop 1 were included in order to provide a variety of experiences to apply algorithm techniques into problem solving. Compared to Workshop 1, this round has provided the chance for studies to carefully examine details of learners' problem solving processes; however, many learners still failed to solve problems and clearly express the thinking processes in their problem solving.

The analysis of the problem solving process of learners who provided wrong answers revealed the following points for improvement.

First, 9 specific problem solving strategies were provided to participants. The difficult questions were primarily about breaking a complex problem into small problems until participants were able to identify the appropriate algorithm design skills and apply them, expressing the data relationship in a graph or a table, and finding patterns. Whether learners can consciously devise strategies they can apply to solve problems has significant effects on problem solving; therefore, methods to visualize the strategies are required.

Since the 4 strategies in Workshop 2 contain a variety of sub-strategies, these were modified to be more specific and increased learning effects. The first strategy of "understanding the core elements" was divided into these 3 stages: "setting purposes for problem solving," "understanding information," and "restrictions of problems." The third strategy of "modeling" was specified to easily approach to the problem and it was divided into the following stages: "selecting algorithm design skills necessary for problem solving and removing unnecessary elements," "expressing in graphs, tables, or charts," "selecting algorithm design skills necessary for problem solving," "breaking a big problem into small problems," "finding patterns," and "designing algorithms." The second strategy of "removing intuition" and the fourth strategy of "execute, review, and modify" remained the same as in Workshop 2. More sophisticated strategies reflected some of the elements related to abstraction among CT elements described in CSTA (Computer Science Teachers Association) and ISTE (International Society for Technology in Education) in 2011 [42]. These can work as scaffolding for learners to overcome their difficulties in finding answers to problems.

Second, PBAL proved how useful it is to solve real-life problems. Puzzles' problem situations motivated learners to participate in the program; however, it was observed that the motivation could not be sustained. The reason for this was revealed in the interviews with learners; the learners were found to have no clear understanding of how puzzles were related to CS and why puzzle problems should be solved. Therefore, we informed learners about why they need to learn puzzles and how puzzles were related to real-world problems and algorithm design skills. In addition, we notified learners about what they could do with the accumulated experiences of puzzle solving for each questionnaire.

Third, we controlled the puzzles' level of difficulty by deleting, modifying, and adding puzzles. If puzzles offered the chance to learn algorithm design skills but requested too much mathematical knowledge, the questionnaires focusing too much on mathematical problem solving were removed. Further to provide a variety of problem solving experiences, we added a binary tree, an encryption algorithm, deadlock solving, and greedy algorithms.

Fourth, we designed a specific strategy of feedback and scaffolding that teachers could provide. We planned to provide scaffolding at the appropriate time by specifying what knowledge should be imparted, what problem solving strategies should be applied, and when this should happen. The self-reports reflected the problem solving strategies and guided learners through the problem solving strategies, which worked as scaffolding. The strategies included in the self-reports were provided as sub-questionnaires as, for example, "What is the goal in this puzzle?" and "What information is inferred in this puzzle?"

Fifth, in the last stage of explaining puzzle answers, learners were encouraged to share problem-solving methods with each other and perform a comparative analysis of the efficiency of each other's method without having the teachers explain and deliver the answers.

4.3 Workshop 3: The Completion of PBAL

In Workshop 3, PBAL was tested for a total of 20 h, reflecting the points for improvement found in Workshop 2. A total of 22 puzzles were given to participants in Workshop 3 as 5 puzzles not proper students' level of understanding were removed.It was confirmed that learners try to apply the nine strategies provided for problem solving and their problem solving became more sophisticated. Learners with high accuracy started to express in more abstract forms on the self-reports as the learning continued. These learners did not use strategies to collect information and conditions necessary for goals and problem solving, but started from the stage of finding necessary elements for problem solving, or skipped the stage of expressing simple information. In addition, they performed debugging to find which part went wrong when errors occurred in problem solving.

In the interviews, we verified that many learners found the program interesting. The learners answered that they realized why puzzle solving was helpful and responded that they would love to have more opportunities to access to these puzzles. There were also answers stating that puzzles offered a creative way to learn, away from simple memorizing. However, there were still respondents who said puzzles were difficult. Those who failed to find the correct answers could not find the algorithm design skills appropriate for problem solving or express problem solving processes logically; some said, "It could be done this way," and modified the level of the problem situation to a level they could handle.

4.4 Effects of PBAL

To verify the learning effects of PBAL, The *T* test was conducted between PBAL group and traditional algorithm learning group and differences of learning effect was compared and analyzed.

Learners' CT abilities before and after adopting PBAL are measured. The results are described in Tables 2 and 3.

Table 2 The pre-test and post-test results

	Group	N	M	SD	*t*	*p*
Pre-test	Experiment	44	1.91	1.939	1.781	.079
	Control	38	1.24	1.384		
Post-test	Experiment	44	3.91	2.550	2.862	.005
	Control	38	2.40	2.188		

Table 3 The difference between pre-test and post-test scores

	N	M	SD	*t*	*p*
Experiment	44	−2.00	2.736	−4.848	<.000
Control	38	−1.16	2.260	−3.158	.003

A 12-point scale is used; learners can get a maximum of 12 points and a minimum of zero. After adopting the PBAL, it is confirmed that learners' CT capacity in the experimental group improves significantly as compared to that in the control group. PBAL learning is revealed to be more effective. The average score of learners is 3.91 out of 12, which implies that there should be continued efforts to improve PBAL.

The problem solving process of respondents who have given wrong answers indicates that they can make a use diagramming strategies to better understand the relationship between material necessary for problem solving. However, in most cases, they are observed to fail in breaking a problem into smaller problems to find patterns, or fail to find patterns after breaking a problem into smaller problems. Therefore, the reasons for the failures should be thoroughly investigated to understand why they occurred, whether due to a lack of experience in puzzle solving or the level of the algorithm's difficulty for elementary school students.

In general, most learners are used to a learning method that requires them to find an answer to a problem. This has discouraged learners from effectively exploring problem solving processes in the early stage of learning. In addition, learners did not realize the importance of systematically planning a problem solving process in advance, and rather focused on just finding a solution. Our study, however, shows that learners' perception has been improved as they have realized the importance of planning a problem solving process, and they have adopted a systematic approach to find a solution.

5 Conclusions

CT is a term that appears in CS; however, it is a core capacity for any individual living in the twenty first century. Abstraction, an element of CT, is a process of designing algorithms to solve problems; only well-designed algorithms can reduce trial and error iterations in problem solving and help find answers. Particularly, algorithm design skills support problem solving effectively and efficiently; that is the essential thing for people learning CS. Effective algorithm learning is completed when learners understand the concepts and principles of algorithm design skills as well as having experience in solving problems by applying the suitable algorithm design skill to a problem; this can eventually improve learners' CT.

Traditionally, students have learned algorithm design skills and developing programs separately or implemented well-defined algorithms to develop programs. Students have little chance to analyze problems using abstraction principles and apply algorithm design skills under traditional algorithm learning method. Even though it is essential for learners to have actual experiences of applying various algorithm design skills into the various kind of real-world problem situations, they have to consume much time and efforts for solving real world complex problems and learners' interests may decreased if they fail for solving

problems. Puzzles are suitable learning tools to overcome these disadvantages that could be raised when learning algorithms real-world problem solving situations. As using well designed puzzles, teachers are able to provide various problems and induce learners' motivation easily in the classroom.

In this study, we developed the PBAL including well-designed puzzles for Informatics-gifted South Korean students (4th grade to 6th grade), especially, focusing on providing effective algorithm learning experiences for acquiring CT competency. Also, we revised the PBAL during the 3 workshops iteratively based on the results of students' outcomes and responses of interview according to DBR methodology to improve the quality of the PBAL.

In conclusion, the research results show that PBAL has better effects on improving CT competency compared to the traditional algorithm learning method. Despite of the positive research results, the PBAL need to be revised for applying various classroom fields and the systematical research is required to develop the CT assessment instrument to provide better validity and reliability of measuring CT competency.

References

1. Chowdhry, B. S. (2013). Successful transformation of ICT graduate program: A role model for developing countries. *Wireless Personal Communications, 69*(3), 1013–1023.
2. Park, J. B., Ji, H. S., Jo, J. C., & Lim, H. S. (2015). A method for measuring cooperative activities in a social network supported learning environment. *Wireless Personal Communications, 89*(3), 863–879.
3. Lee, J.-Y. (2010). r-Learning and educational information policies. *Journal of the Korea Convergence Society, 1*(1), 1–15.
4. Lee, Y. J., Paik, S. H., Shin, J. H., Yu, H. C., An, S. J., Choi, J. W., Jeon, S. G. (2014). *Research for introducing computational thinking into primary and secondary education.* Seoul: Korea Foundation for the Advancement of Science and Creativity.
5. Jeon, H. S., Kim, G. M., & Kim, S. S. (2012). The effect of unplugged algorithm learning on gifted and talented students' academic achievement. *Korean Journal of Teacher Education, 28*(1), 111–127.
6. Barab, S. A., MaKinster, J., & Scheckler, R. (2003). Designing system dualities: Characterizing a web-supported teacher professional development community. *Information Society, 19*(3), 237–256.
7. Seo, Y. M., & Lee, Y. J. (2010). A subject integration robot programming instruction model to enhance the creativity of information gifted students. *The Journal of Korean Association of Computer Education, 13*(1), 19–26.
8. Michalewicz, Z., Falkner, N., & Sooriamurthi, R. (2011). Puzzle-based learning: An introduction to critical thinking and problem solving. *Decision Line, 42*(5), 6–9.
9. Michalewicz, Z., & Michalewicz, M. (2010). *Puzzle-based learning: An introduction to critical thinking, mathematics, and problem solving*. Melbourne: Hybrid.
10. Santiprasitkul, S., Sithivong, K., & Polnueangma, O. (2013). The first year nursing students' achievement and critical thinking in local wisdom course using problem based learning process. *Wireless Personal Communications, 69*(3), 1077–1085.
11. Shinde, V. V., & Inamdar, S. S. (2013). Problem based learning (PBL) for engineering education in India: Need and recommendations. *Wireless Personal Communications, 69*(3), 1097–1105.
12. Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35.
13. Woo, Hee-Sun, Yeom, Mi-Ryeong, & Jung, Doo-Yong. (2016). An analysis on the UCC media for STEAM integrated education. *Journal of the Korea Convergence Society, 6*(7), 43–48.
14. Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 366*(1881), 3717–3725.
15. Perlis, A. (1962). The computer in the university. In M. Greenberger (Ed.), *Computers and the world of the future*. Cambridge, MA: MIT Press.
16. Peter, W. (2010). *Mathematical puzzles: A Connoisseur's collection*. Natick, MA: A K. Peters, Ltd.
17. Lee, Yong-Bae. (2014). Analysis on computer education in elementary schools in North Korea and South Korea with further prospect. *Journal of the Korea Convergence Society, 5*(4), 49–60.

18. Guzdial, M. (2008). Education: Paving the way of computational thinking. *Communications of the ACM, 51*(8), 25–27.
19. Park, J. H. (2015). Effects of storytelling based software education on computational thinking. *Journal of the Korean Assoiation of Information Education, 19*(1), 57–68.
20. Kim, S. H. (2015). Effects of teaching and learning strategies of learner-centered learning for improving computational thinking. *Journal of The Korean Association of Information Education, 19*(3), 323–332.
21. Choi, H. S. (2014). Developing lessons and rubrics to promote computational thinking. *Journal of The Korean Association of Information Education, 18*(1), 57–64.
22. Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior, 41*, 51–61.
23. Rubinstein, A., & Chor, B. (2014). Computational thinking in life science education. *PLoS Computational Biology, 10*(11), e1003897.
24. Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers and Education, 72*, 145–157.
25. Cho, H. K. (2001). 2011 International Olympiad in informatics observation. *Communications of the Korean Institute of Information Scientists and Engineer, 19*(11), 43–46.
26. Jang, J. H. (2007). Olympiad in informatics and informatics education. *Communications of the Korean Institute of Information Scientists and Engineer, 25*(7), 61–66.
27. Falkner, N., Sooriamurthi, R., & Michalewicz, Z. (2010). Puzzle-based learning for engineering and computer Science. *IEEE Computer, 43*(4), 20–28.
28. Merrick, K. E. (2010). An empirical evaluation of puzzle-based learning as an interest approach for teaching introductory Computer Science Education. *IEEE Transactions on, 53*(4), 677–680.
29. Parhami, B. (2008). Motivating computer engineering freshmen through mathematical and logical puzzles. *IEEE Transactions on Education, 52*(3), 360–364.
30. Parhami, B. (2009). Puzzling problems in computer engineering. *IEEE Computer, 42*(3), 26–29.
31. Levitin, A., & Papalaskari, M. A. (2002). Using puzzles in teaching algorithms. *ACM SIGCSE Bulletin-Inroads: Paving the way towards excellence in computing education, 34*(1), 292–296.
32. Choi, J. W., Lee, E. K., & Lee, Y. J. (2014). Studying the possibility of puzzle based learning for informatics gifted elementary students education. *The Korean Association of Computer Education, 16*(5), 9–16.
33. Lee, E. K., Choi, J. W., & Lee, Y. J. (2014). The analysis of informatics gifted elementary students' computational problem solving approaches in puzzle-based learning. *Journal of the Korea Society of Computer and Information, 19*(1), 191–201.
34. Barab, S. A. (2006). Design-based research. In R. K. Sawyer (Ed.), *The Cambridge handbook of the learning sciences* (p. 2006). Cambridge: Cambridge University Press.
35. Wang, F., & Hannafin, M. J. (2005). Technology-enhanced learning environments. *Educational Technology Research and Development, 53*(4), 5–23.
36. Brown, A. L. (1992). Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *The Journal of the Learning Sciences, 2*(2), 141–178.
37. Collins, A. (1992). Towards a design science of education. In E. Scanlon & T. O'Shea (Eds.), *New directions in educational technology*. New York: Springer.
38. Davis, G. A., Rimm, S. B., & Siegle, D. (2011). *Education of the gifted and talented* (6th ed.). Boston: Pearson.
39. Renzulli, J. S. (2012). Reexamining the role of gifted education and talent development for the 21st century: A four-part theoretical approach. *Gifted Child Quarterly, 56*(3), 150–159.
40. Lee, J. H., & Oh, H. J. (2009). Design and validation of education contents of algorithm for the gifted elementary students of Computer Science. *Journal of Gifted/Talented Education, 19*(2), 353–380.
41. Lee, E. K. (2009). A robot programming teaching and learning model to enhance computational thinking ability. Doctoral dissertation, Graduated School of Korea National University of Education.
42. CSTA and ISTE. (2015). Computational thinking teacher resources second edition," Retrieved from https://csta.acm.org/Curriculum/sub/CurrFiles/472.11CTTeacherResources_2ed-SP-vF.pdf, March 5, 2015.

Jeongwon Choi received her Ph.D. degree in Informatics Gifted Education from the Korea National University of Education in 2015. She has taught Computer Science in secondary school as a teacher since 2003. Her current research interests include computational thinking, learning science, STEAM education, and Informatics gifted education.

Youngjun Lee received his Ph.D. degree in Computer Science from University of Minnesota in 1994 . He currently serves as a professor at the Korea National University of Education since 2003. His current research interests are in the area of computational thinking and learning science.

Eunkyoung Lee received her Ph.D. degree in Computer Science Education from the Korea National University of Education in 2009. She currently serves as an associate research fellow at Korea Institute for Curriculum and Evaluation since 2013. Her current interests are in the area of computational thinking and learning science.