Helping Students Solve Parsons Puzzles Better

Amruth N. Kumar
Computer Science
Ramapo College of New Jersey
Mahwah, NJ, USA
amruth@ramapo.edu

ABSTRACT
In a Parsons puzzle, the student must re-assemble the lines of a program that are provided in scrambled order, and eliminate any distracters included within the scrambled lines of code. We investigated two issues in terms of whether they helped students solve the puzzles better: 1) Would it be better to present distracters and the lines of code of which they are a variant paired together or randomly separated apart? 2) Would telling students that they would not be penalized for any mistakes they make before submitting a complete solution for the first time enable them to get closer to the correct solution when they first submit it? We conducted a controlled study over five semesters and used ANOVA to analyze the data collected from introductory programming students who solved Parsons puzzles on if-else statements. We found that when students were told that their puzzle-solving actions before the first submission would not be penalized, they took significantly more exploratory actions. But, their solution was not significantly closer to being correct. So, trial-and-error exploration was no better for solving Parsons puzzles than deliberate approach. Presenting distracters paired together with the original line of code was found to be beneficial only on the longer puzzle.

ACM Reference format:

1 Introduction
In a Parsons puzzle [17], a problem statement and a program written for the problem are presented to the student. But, the lines in the program are presented scrambled. The student is tasked with rearranging the lines in their correct order. A few distracters, which are incorrect variants of lines of code in the program, are mixed in with the scrambled lines. The student must properly identify and eliminate these distracters. The puzzles were designed to be an engaging way to learn programming [17].

Parsons puzzles have gained popularity for use in exams [4], since they are easier to grade than code-writing exercises. Scores on Parsons puzzles were found to correlate with scores on code-writing exercises [4]. Solving Parsons puzzles was found to take significantly less time than fixing errors in code or writing equivalent code, but resulted in the same learning performance and retention [7]. In electronic books, students were found to prefer solving Parsons puzzles more than other low-cognitive-load activities such as answering multiple choice questions and high-cognitive-load activities such as writing code [6]. Researchers have placed Parsons puzzles in a hierarchy of programming skills alongside code-tracing [15]. Software to administer Parsons puzzles have been developed for Turbo Pascal [17], Python (e.g., [3,10]) and C++/Java/C# [12].

The focus in Parsons puzzles research is lately on how students solve them and what does/does not help students solve them better, e.g., the patterns in how students go about solving the puzzles [9,11]; that sub-goal labels help students solve puzzles significantly better [16]; adaptive practice of Parsons puzzles is more efficient while being just as effective as writing code [5]; but, motivational supports [13] and the use of mnemonic variable names [19] do not help students while solving puzzles.

In this vein, we investigated two issues in terms of whether they helped students solve Parsons puzzles better, i.e., with fewer corrective actions and/or less time:

1. A distracter is an incorrect variant of a line of code. Would distracters and the lines of code of which they are a variant be better presented paired together or separated randomly apart?
2. Students could be penalized for every mistake while solving a puzzle, or only those that they commit after submitting a complete solution for the first time. Would telling students that they would not be penalized for any mistakes they make before submitting a complete solution for the first time prompt them to explore the solution more before submitting it for the first time? Would this enable them to get closer to the correct solution when they first submit it?
2 The Study

2.1 Distracters
A distracter (also spelled distractor in literature) is an incorrect variant of an existing line of code in the puzzle. It may be a syntactically incorrect variant (e.g., 5 = variable; instead of variable = 5); semantically incorrect (e.g., variable += 2; instead of variable *= 2); or pragmatically inadvisable (e.g., int x; instead of int months) [17]. Each distracter and the line of code of which it is a variant may be presented paired together in the scrambled code, or randomly separated apart. Presenting the two paired together reduces the cognitive load that is not relevant to problem-solving or programming [4]. So, we hypothesized that students would be able to solve Parsons puzzles with fewer mistakes and hence, fewer actions when distracters were paired together rather than separated apart.

2.2 Grading Schemes
The solution to a Parsons puzzle is complete when all the lines in the puzzle have been accounted for, i.e., they have either been placed in the solution space or have been discarded as distracters. The solution is also correct when all and only the distracters have been discarded and all the lines in the solution space have been placed in their correct order. A correct solution is necessarily complete, but the vice versa is not true.

If a puzzle contains n lines of code (including distracters), it can be solved completely and correctly with n actions, assuming each line is individually manipulable. The possible puzzle-solving actions include 1) moving a line of code from problem/puzzle space to (its correct location in) solution/program space; 2) discarding a distracter from problem space; 3) reordering a line of code within solution space. A student who is not sure of the correct solution typically takes many more than n actions to solve a puzzle, and all the actions beyond the first n are redundant. During grading, these redundant actions are penalized. We refer to this scheme of penalizing all the redundant actions as aggressive grading, the grading that was in effect early in our use of Parsons puzzles.

In response to feedback from students who thought that aggressive grading scheme was too unforgiving, later in our use of Parsons puzzles, we instituted a new grading scheme: a student would be allowed to take as many actions without any grading penalty until the student submitted a complete solution for the first time (henceforth referred to as pre-completion actions). The student would be penalized for only the actions taken after the submission of the first complete solution to fix any remaining errors in the solution (henceforth referred to as post-completion actions). We refer to this scheme of penalizing only post-completion actions as lenient grading.

A student must carry out at least n actions before submitting a complete solution for the first time. So, the following relations hold:

\[ \text{pre-completion actions} \geq n \]

In aggressive grading scheme, all the actions beyond the first n are penalized, i.e., negatively graded when computing the student’s grade on the puzzle. So, the grade penalty is applied to:

\[ \text{pre-completion actions} + \text{post-completion actions} - n \]

In lenient grading scheme, negative penalty is applied only to post-completion actions.

Cognitive strategies used to solve Part-Complete Solution Method (PCSM), a variant of Parsons puzzles [20], have been categorized into five levels: unplanned and random at the bottom end, followed by low-level consideration, some level of consideration, high level of consideration with some strategy and finally, deliberate approach at the top end [20]. Given that students with aggressive grading would want to minimize all redundant actions, they had greater incentive to use deliberate approach during pre-completion actions than those with lenient grading scheme. In contrast, since they would not be penalized, students with lenient grading had the incentive to engage in trial-and-error exploration during pre-completion actions. We wanted to find out which of the two grading schemes led students closer to the correct solution when they first submitted their solution – aggressive grading that incentivizes deliberate approach or lenient grading that incentivizes trial-and-error approach.

So, our experimental hypotheses were two-fold:
1. Students with lenient grading would engage in more trial-and-error exploratory actions before submitting a complete solution for the first time, i.e., pre-completion actions would be significantly more with lenient grading than with aggressive grading;
2. Lenient grading that incentivizes trial-and-error exploration would result in a solution closer to being correct when first submitted than aggressive grading that incentivizes deliberate approach. (Null hypothesis would be to find no difference between the two grading schemes.) So, post-completion actions would be significantly less for lenient grading than aggressive grading.

3 The Experiment

3.1 The Instrument
For this study, we used a Parsons puzzle software tutor from a suite of tutors called epplets [12]. The tutor presents a series of Parsons puzzles, lets the student solve the puzzles, provides feedback to help the student correct the solutions, and grades each solution. Its user interface contains an Instruction Panel where the problem statement is presented, a Problem Panel where scrambled lines are listed, a Solution Panel where the student re-assembles the lines of code in their correct order, a Trash Panel to which the student drags distracters and a Feedback Panel where feedback is presented to the student [12].

- this layout with drag-and-drop interface affords the puzzle-
solving actions described in Section 2.2. It is accessible online (epplets.org) and can be used by students on their own time.

Students using the tutor cannot submit their solution to a puzzle until it is complete, i.e., they have accounted for all the lines in the puzzle. With the aid of the feedback provided by the tutor, students can fix errors in their solution and resubmit their solution as often as they want till the solution is correct. They are required to correctly solve a Parsons puzzle before moving on to the next puzzle.

The tutor is adaptive, i.e., if it determines that a student does not know how to solve a puzzle, it will present additional practice puzzles to the student. So, students had the incentive to solve each puzzle with the best grade possible, i.e., with the fewest redundant actions possible. The more mistakes students made on the initial puzzles, the more puzzles the tutor would have them solve.

The tutor used the grading schemes described earlier to determine whether a student knew how to solve a puzzle: it used aggressive grading scheme from Fall 2016 to Fall 2017 and lenient grading scheme in Spring 2018 and Fall 2018. Before each tutoring session, the grading scheme in effect was explained to the students and they were asked to aim for the best grade possible when solving puzzles.

3.2 The Puzzles

For this study, we had students solve Parsons puzzles on \texttt{if-else} statements. The first puzzle was on a single \texttt{if-else} statement and the second was on nested \texttt{if-else} statements. All the students solved these puzzles. The puzzles that they solved after the first two, if any, depended on how well they had solved these two puzzles. Therefore, for our study, we considered data from only the first two puzzles.

The first puzzle was on a program to read two numbers and print the smaller value among them. This program contained 16 lines that were provided scrambled, which included all the lines of code within \texttt{main} as well as 2 distracters. The shell of \texttt{main} and the comments within it were provided as context [20]. The largest number of lines of code students had to re-assemble under a single comment/context was 8. Following is the puzzle program in C++:

```c++
int main() {
   // Declare firstInput
   short firstInput;
   // Declare secondNum
   short secondNum;
   // Read firstInput
   cin >> firstInput;
   // Read secondNum
   cin >> secondNum;
   if( firstInput < secondNum )
   {
      cout << firstInput;
   } // End of if-clause
   else
   {
      cout << secondNum;
   } // End of else-clause
} // End of function main
```

Note that the puzzle contains 16 lines. It includes two distracters that have been presented paired.

The second puzzle was on a program to read numerical grade, convert it to letter grade - A (90 and up), B (80-89), C (70-79), D (55-69) and F otherwise - and print it. This program contained 36 lines that were provided scrambled, which included all the lines of code within \texttt{main} as well as 2 distracters. Once again, the shell of \texttt{main} and the comments within it were provided as context. The largest number of lines of code students had to re-assemble under a single comment/context was 29. Following is the program in C++:

```c++
long number;
char letterGrade;
int main() {
   // Declare number
   cin >> number;
   // Declare letterGrade
   letterGrade = 'A';
   if( number >= 90 )
   {
      letterGrade = 'A';
   } // End of if-clause
   else if( number >= 80 )
   {
      letterGrade = 'B';
   } // End of else-if-clause
   else if( number >= 70 )
   {
      letterGrade = 'C';
   } // End of else-if-clause
   else if( number >= 60 )
   {
      letterGrade = 'D';
   } // End of else-if-clause
   else
   {
      letterGrade = 'F';
   } // End of else-clause
   cout << letterGrade;
} // End of function main
```
cin >> number;
    // Compute letterGrade based on number
    if (number >= 90)
    {
        letterGrade = 'A';
    } // End of if-clause
    else
    {
        if (number >= 80)
        {
            letterGrade = 'B';
        } // End of if-clause
        else
        {
            if (number >= 70)
            {
                letterGrade = 'C';
            } // End of if-clause
            else
            {
                if (number >= 55)
                {
                    letterGrade = 'D';
                } // End of if-clause
                else
                {
                    letterGrade = 'F';
                } // End of else-clause
            } // End of else-clause
        } // End of else-clause
    } // End of if-clause
}
// Print letterGrade
    cout << letterGrade;
} // End of function main

4 Data Collection and Analysis

4.1 Variables

The dependent variables of the study were:

- **Pre-completion ratio**: calculated as pre-completion actions / n, i.e., the number of actions taken by the student before submitting it for the first time, divided by the number of lines in the puzzle. We considered pre-completion ratio instead of the number of pre-completion actions so that we could compare student performance across two puzzles with different number of scrambled lines.

- **Post-completion ratio**: calculated as post-completion actions / n, i.e., the number of steps, if any, taken by the student to correct the solution after submitting it for the first time, divided by the number of lines in the puzzle.

- The time spent per line to solve the puzzle, in seconds

- The grade on the puzzle. In order to be able to compare aggressive and lenient grading schemes, we used lenient grading scheme to re-calculate the grades of all the students based on their post-completion actions only.

The independent variables of the study were:

- Grading scheme: the grading scheme that students were told was being used when they were solving the puzzles - aggressive or lenient.

- Pairing of distracters: whether each distracter was presented paired together or separated apart from the correct line of code of which it was a variant.

- The puzzle number: whether the data pertained to the first or the second puzzle presented to the student.

4.2 Data Collection

The subjects of the study were students in introductory programming courses, who were asked to solve the puzzles as after-class assignment, and who had given IRB permission for their data to be used in the study. We collected data over 5 semesters: Fall 2016 through Fall 2018. In each semester, students from multiple institutions used the tutor. The institutions were randomly assigned to control group (distracters separated apart) and experimental group (distracters paired together). The tutor used aggressive grading during the first three semesters (Fall 2016 – Fall 2017) and lenient grading in the last two semesters (Spring 2018, Fall 2018). The number of students in each of the four conditions (pairing of distracters X grading scheme) is listed in Table 1. Varying levels of retention and assignment submission rates explain the differences in the number of students in the four conditions.

Students could use the tutor as often as they wanted. If a student used the tutor more than once, data from only the first session wherein the student had solved the most number of puzzles was considered for the study.

<table>
<thead>
<tr>
<th>Treatment</th>
<th>Aggressive</th>
<th>Lenient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separated apart</td>
<td>192</td>
<td>56</td>
</tr>
<tr>
<td>Paired together</td>
<td>101</td>
<td>162</td>
</tr>
</tbody>
</table>

4.3 Data Analysis

**Pre-completion ratio**: One-way ANOVA analysis of pre-completion ratio with grading policy, pairing of distracters and puzzle number as fixed factors found a significant main effect for grading policy: $[F(1,828) = 10.732, p = 0.001]$. Mean pre-completion ratio was 1.201 ± 0.032 with aggressive grading (at 95% confidence interval) and 1.288 ± 0.041 with lenient grading. In other words, with lenient grading, students took 28.8% more pre-completion actions than the number of lines in the puzzle whereas with aggressive grading, they took 20.1% more actions on average. This confirms the hypothesis that, with lenient grading, students would carry out more trial-and-error exploratory actions before submitting the complete solution for the first time.
The other significant main effect found was for the puzzle number \( [F(1,828) = 21.184, p < 0.001] \): Mean pre-completion ratio was 1.306 ± 0.035 for the first puzzle which contained 16 scrambled lines and 1.184 ± 0.039 for the second puzzle which contained 36 scrambled lines. In other words, students took 30.6% (4.9) more pre-completion actions than the number of lines (16) on the first puzzle, but took 18.4% (6.6) more pre-completion actions on the second puzzle with 36 lines. So, the number of exploratory pre-completion actions did not increase linearly with the number of lines in the puzzle. One possible explanation is practice effect, i.e., students felt more confident at solving Parsons puzzles and therefore, took a smaller percentage of pre-completion actions on the second puzzle than on the first.

**Post-completion ratio:** Similar ANOVA analysis of post-completion ratio found a significant main effect for puzzle number \( [F(1,828) = 8.426, p = 0.004] \): Mean post-completion ratio was 0.496 ± 0.081 for the first puzzle and 0.315 ± 0.091 for the second puzzle. In other words, the post-completion actions students took were 49.6% (7.9) of the number of lines (16) on the first puzzle, and 31.5% (11.3) of the number of lines (36) on the second puzzle. Just as in the case of pre-completion actions, post-completion actions did not increase linearly with the number of lines in the puzzle either.

Students were required to solve each puzzle correctly. Since neither pre-completion actions nor post-completion actions increased linearly with the number of lines in the puzzle, students took proportionately fewer unnecessary actions to solve the second puzzle with 36 lines of code than the first puzzle with 16 lines of code. Apart from practice effect, another possible explanation is that the level of difficulty of solving Parsons puzzles does not increase linearly with the number of lines in the puzzle, possibly because of more duplicated lines (e.g., else). This is a hypothesis worth exploring further with more data.

We did not find a main effect for grading policy, though. Post-completion ratio was statistically no different between students with lenient versus aggressive grading. So, the hypothesis that students with lenient grading would be closer to the correct solution than those with aggressive grading by the time they submitted a complete solution for the first time was not confirmed by the data. **Trial-and-error approach incentivized by lenient grading did not lead to a more correct solution than deliberate approach incentivized by aggressive grading.**

**Time spent per line:** ANOVA analysis of the time spent per line yielded a marginal main effect for grading scheme \( [F(1,827) = 3.318, p = 0.069] \): students spent a mean of 18.741 ± 1.332 seconds per line with aggressive grading compared to 20.736 ± 1.687 seconds per line with lenient grading. So, students spent marginally more time per line with lenient grading than with aggressive grading. Lenient grading may have prompted students to think more before each action.

ANOVA analysis of the time spent per line yielded a significant main effect for puzzle number \( [P(1,827) = 44.977, p < 0.001] \): students took 23.411 ± 1.426 seconds per line on the first puzzle and 16.067 ± 1.608 seconds per line on the second puzzle.

One explanation is practice effect – **students got faster at solving puzzles with practice.**

**Grade:** ANOVA analysis of the grade did not yield any significant main effect, but yielded a significant interaction between pairing of distracters and puzzle number \( [F(1,828) = 5.009, p = 0.025] \), as shown in Table 2. When distracters were separated apart, students scored more on the shorter first puzzle (16 lines) than on the longer second puzzle (36 lines), but this difference was not statistically significant. When distracters were paired together, students scored more on the longer second puzzle than on the shorter first puzzle, and this difference was statistically significant \( [F(1,438) = 6.076, p = 0.014] \). So, the hypothesis that puzzles with distracters paired together are easier to solve than those with distracters separated apart was found to be true only for the longer puzzle.

**5 Discussion**

We found that with lenient grading, students indeed carried out more exploratory actions to solve Parsons puzzles before submitting a complete solution for the first time. They also took marginally more time per line to solve the puzzle. In spite of these, they were no closer to the correct solution than students with aggressive grading because the post-completion actions they needed to correct their solution were no fewer than those needed by students with aggressive grading. In other words, students did not benefit from lenient grading. Trial-and-error exploration was no better for solving Parsons puzzles than deliberate approach.

Empirical research has consistently shown that anxiety negatively affects educational achievement [21]. With aggressive grading, pre-completion actions that are incorrect are irrecoverable in the sense that they will be penalized. In contrast, with lenient grading, they are recoverable, i.e., bear no penalty. So, a disadvantage of aggressive grading, and a reason why students must prefer lenient grading over it is that aggressive grading is likely to induce more anxiety during pre-completion actions. So, it is reasonable to expect that students would do better with lenient grading than with aggressive grading. But, our study did not support this hypothesis.

Anxiety can be task-directed, i.e., facilitating, or task-irrelevant, i.e., debilitating [22]. One explanation for why lenient grading was no different from aggressive grading might be that the stress induced by aggressive grading was task-directed or facilitating. So, unpleasant as the anxiety may have been to the students, it helped them focus on solving the puzzle with as few mistakes as possible.
Presenting distracters paired together reduces cognitive load not relevant to problem-solving or programming [4]. But, we found that pairing together distracters was beneficial only on the longer puzzle.

Parsons puzzles are rapidly gaining popularity in programming education. In this context, research is called for to answer questions such as what affects how efficiently students solve the puzzles and what affects the level of difficulty of solving the puzzles. In this study, we have attempted to answer some questions (Does aggressive grading lead to inefficient puzzle solutions? Does pairing together distracters make a puzzle easier to solve?) while raising another (Is the number of lines in a puzzle an accurate measure of the level of difficulty of solving the puzzle?) We see our study as adding another piece of evidence to the body of literature (e.g., [5,9,11,13,16,19]) on how to effectively use Parsons puzzles for programming education.

ACKNOWLEDGMENTS
Partial support for this work was provided by the National Science Foundation under grants DUE-1432190 and DUE-1502564.

REFERENCES