**Ericson, B. J., Denny, P., Prather, J., Duran, R., Hellas, A., Leinonen, J., ... & Rodger, S. H. (2022). Parsons Problems and Beyond: Systematic Literature Review and Empirical Study Designs. Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education, 191-234.**

**Motivation**

As Parsons problems become popular in programming education, review of past research on Parsons problems is needed to understand the its utility in education context, evidences for its effectiveness, and the remaining gaps, in order to foster future research and use of parsons problems in education.

**Definition and Feature of Parsons Problems:**

Logical arrangement problems are considered as foundation for computing. A lot of problems people met in programming are due to wrong order of some necessary steps instead of conceptual failures. In Parsons problems, users need to construct a solution by placing fragments into correct orders and adding possible distractors to help the programmers learn about the logic behind coding and some common mistakes. Parsons problems can help find students’ difficulties, provide immediate feedback, and improve learning performance due to (1) lower cognitive load; (2) worked example effect; (3) higher self-efficacy of students (4) increase metacognition skills of novice programmers. To produce an up-to-date survey of use of Parsons problems in computing education research, this paper uses title and abstract scanning and defined inclusion criteria to review the majority of related data and research.

**Research Method**

By searching with refined search terms in ACM Digital Library, IEEE Xplore, and Scopus, and capturing relevant ones using forwarding snowballing, 494 articles are found and finally 386 pieces are analyzed. Articles are excluded according to abstract, title, and certain criteria that determine whether it contain Parsons problem related content. Pieces not written in English are also excluded.

For the research background of these papers, frequencies of occurrence for each category are computed for questions with predefined fields, and if needed, cleaned and standardized the answers before computing the frequency. Bibliometric data are directly extracted from the BitTES entries. For research questions, two researchers directly use the research question stated in paper or identify it from the paper. Finally, for evidence of the advantages and limitation of Parsons problems, the validity of experiments in paper and various characteristics of these paper are exanimated.

**Data about How Parsons Problems are Researched in Past Studies**

With all articles analyzed, the paper defines Parsons problems as “a type of exercise used for learning and assessing the ability to construct programs” and their feature is “a limited set of fragments that the user needs to use to produce a solution”. With a positive slope for the publication trend and citation trend, the impact of Parsons problems and their widespread usage is proved, and the interactive online learning material may be one factor fostered such phenomena. Research are mainly classroom based and are in United States. In most research, Parsons problems are contained in the courses, graded for being solved, and used relatively extensively. Averagely each research have 455 participants due to a few of research with large number of participants.

Basic feature of Parsons problem occurs in majority of the paper which can identify what features are used. Various tools to implement Parsons problem are used among the articles. The research direction of the paper including student or instructor identity, pedagogy , using Parsons problems as scaffolding interventions, problem solving solution path, Parsons problems delivery, learning via gamification, and some categories with few papers. The most common research theme is Learning Programming.

The majority of students in the research found Parsons problems useful for learning. There are also doubts about whether solving Parsons problems really improve students’ coding skills and speed, but there is evidence that Parsons problems can impact student cognitive load. A few research have clear research questions or defined the limitation of validity.

To compensate the limitation in current research, future research should focus more on the use of Parsons problems (1) in advanced programming courses (2) in larger classes (3) for students with different backgrounds (4) with error messages (5) with more participants, etc. Possible bias may also occur due to participants who first studied to learn system.

**Process of In-a-box Research with Runestone**

To conduct more comprehensive research on effectiveness of Parsons problems, researchers use Runestone Academy with an introduction video and a feedback box. Participants are divided to solve Parsons problem or write code. Questions include writing and fixing code and graded based on percent of unit tests passed.

The participants are novice programmers from DePaul University and the University of Virginia with foundamental knowledge in Python 3, resulting in a higher score in Parsons problems group than writing code group. The final study design contains 4 parts: pre-survey, introduction, practice problems, and post survey, randomly assigning students to either test groups. Problem sets include Solving Adaptive Parsons vs Write Code, Solving Adaptive Parsons Problems with Distractors vs No Distractors, and Solving Write Code versus Write Code with Adaptive Parsons as Scaffolding.

Future studies can be implemented into anonymous class activities and reward with credits, and can also be done online through Runestone. Except current research gaps, evaluations on new types of Parsons problems are also expected.

**Curious**:

Since different people have different coding styles, how can we select a standard code for a problem to make its Parsons version? How can different styles of the original code influence people’s learning rate?

**Haynes-Magyar, C., & Ericson, B. (2022, November). The Impact of Solving Adaptive Parsons Problems with Common and Uncommon Solutions. In Proceedings of the 22nd Koli Calling International Conference on Computing Education Research (pp. 1-14).**

**Motivation**

To help novice programmers efficiently and effectively learn programming, it is important to investigate when a Parsons problem is more efficient to solve rather than writing code. While previous research proved that Parsons problems may impact both cognitive learning outcomes and behavioral and affective learning outcomes, it is possible to improve how adaptive Parsons problems are sequenced by analyzing the relationships between factors such as problem- solving efficiency and cognitive load ratings.

**Background**

Parsons problems are defined as code completion problems that require learners to place code blocks in the correct order and can vary by dimensions, fill-in-the-blank variable options, feedback, adaptation, and the use of distractors. When a Parsons problem is adaptive, its difficulty will be based on a learner’s performance. Evidence shows that Parsons problem can help students learn coding more efficiently with less time and lower cognitive load, but this is not always true. Prior studies showed that (1)students are more likely to be influenced by a pattern when they first meet a Parsons problem or a code writing question, (2) cognitive load of Parsons problems with unusual solution is similar to writing the equivalent code, and (3) most students used the unusual Parsons problem solution to later solve an equivalent write-code problem. As a result, the researchers hypothesized that changing the unusual Parsons problem solution to the most common student-written solution would make that problem significantly more efficient to solve.

To test the hypothesis and answer its related questions about the influence of common and uncommon Parsons problem solutions on problem- solving efficiency , a mixed within- between-subjects experiment is conducted, avoiding bias due to intra-individual variation, in order to answer

**Experiment Method**

In the first part, participants are students in an intermediate Python programming course which requires prior programming experience in a research institution in the northern Midwest of the United States. Two versions of problem sets are adopted from previous study. Students will solve either writing code problems or Parsons problems, and exchange to another type of question for next time. After each question, they rate how much mental effort they invested in solving the problem. Individual completion of each problem set will give the student 10 credits. When the course ends, students can complete two more problem set for extra credit. In total, 95 participants completed both the adaptive Parsons version and equivalent code writing version for some or all of the problems correctly.

In the second part, the online think-aloud observation collects information of participant’s prior programming experience, ask participants to do random problem sets and express idea when solving. The researchers then perform a quantitative analysis on the participants to understand people’s reaction to the modified process and new-added help-seeking option.

**Result**

This research found that 1) the efficiency is significantly higher when students use a more common solution to solve.a Parsons problem, and is significantly lower when students solve a Parsons problem with an uncommon solution than solving the equivalent write-code problem. (2) Students’ coding behaviors when solving a writing code problem would be influenced by the solution of a similar Parsons problem, indicating that Parsons problems should be generated from the most common student-written code to effectively improve leaner’s efficiency. (3) Students who solve similar Parsons problem first appear to have lower cognitive load than those who directly solve the writing code problem even when the solution is uncommon. (4) If students need to solve Parsons problem with uncommon solutions, it is more likely that they will need more help in some certain aspects, as reflected in the Paas scale. Also, consistent, disciplined self-regulation during problem-solving like reporting cognitive load may help with problem solving, while distractors can slow the problem-solving process.

To support both high-end students and struggling students, the ability to switch from a Parsons problem to an equivalent write- code problem is added, and more types of problems and functions are added in CodeSpec to help learners. Future study should be replicated in other contexts and languages in future to eliminate the limitation of current research. Future research can focus on the combination of other programming task variables and hypothesis modifying the type and amount of programming problems.

**Helminen, J., Ihantola, P., Karavirta, V., & Malmi, L. (2012, September). How do students solve parsons programming problems? an analysis of interaction traces. In Proceedings of the ninth annual international conference on International computing education research (pp. 119-126).**

**Motivation**

To visualize and analyze students’ programming process and their difficulties to provide feedback and improve their performance.

**Background-previous research and technical support**

When students solve programming problems, it’s hard for teachers to trace the process, find their difficulties and provide feedback and guidance. Previous research found a correlation between Parsons scores and code writing scores and tried to model students’ programming process with machine learning and records of compilation events, which subsequently, js-parsons tool was implemented to record learners’ interaction for analysis.

To get enough effective data for tracing students’ programming process, an experiment was conducted to solve Parsons problems under js-parsons environment, which enables researchers to examine relative data in terms of program construction sessions.

**Experiment Procedure**

The mode of js-parsons used in this experiment enables students to drag-and-drop the code blocks from left to right to generate solution with modified feedback function when flaws occur. Changes in the input, solution, and all feedback requests are recorded for analysis.

Data is collected from students in a Web Soft- ware Development (WSD) course in Spring 2011 and a CS2 course taught with Python in Spring 2012 at Aalto University, Finland, solving five different problems, each has a unique solution, and a background survey about prior programming experience. The analysis focus on data from WSD courses. Some amount of data was ignored due to an error in problem 1 and limitation of problem 2.

**Result and Analysis**

Solution for the parsons problems are viewed as a path from a state of empty code to a state with the correct solution, and the number of steps in the solution path is the number of edges traversed. The result shows that the hardest problems are problem 4 and 5 by analyzing the median of data. Feedback was used for trial-and-error like behavior, which sometime generates uncommon, nonsensical states in the solving process. Aggregate graphs and charts of steps and states are generated as well as individual solution paths, focus on transitions that are common among students. Data also shows that students have some block-driven or structured control-flow logic instead of classical linear logic when solving Parsons problem. Data about students’ difficulties are normalized to equal weight and common incorrect states can be found, and students appear to have difficulties in problems with loops.

One limitation of the study is that the sources of errors and flaws are speculated. For future studies, data about people’s thoughts can be collected to solve this limitation by recalling or talking. Also, more data should be collected to examine some special states of learners, like the jumbled state graphs or concentrated sidetracks.. Automatic feedback is expected to prevent students from such ineffective patterns. Also, more factors like distractors should be added to intimate real coding situations.

**Curiosity**

The study mentions that some students uses feedback each step to check whether their last action is correct and thus sometime generate some strange states in the tool, which may be a noise in the data and is helpless for analyzing students’ learning curve and performance. Though feedback is demonstrated as an immediate response that is important for improving students’ coding skills, it now seems to also cause problems. How should we prevent them from overusing feedback?

**Koedinger, K. R., Carvalho, P., Liu, R., & McLaughlin, E. A. (2023). An Astonishing Regularity in Student Learning Rate.**

**Motivation**

To figure out whether and why are students’ learning rates different and how to improve their learning rate.

**Why this is important?**

Academic learning behaviors had long been an important topic to improve education and foster scientific progress. Past research in people’s academic learning behaviors had proved the effectiveness of the knowledge component learning model. This model/theory claims that by dividing learning process to discrete units (knowledge components), the prediction produced can be tested across contexts. This research is going to combine the cognitive models with the statistic models to figure out the relation between the amount of practice and initial performance, final correctness, and learning rates.

As the researchers found that the learning opportunities for different students varies, they then investigated the factors influencing this variation. The data will also provide important information for the debate about the relationship between number of practice opportunities and learning behaviors. There’s also an argument about whether substantial difference occurs between learning rates of students. So the researchers will also identify the influence on learning rates from students’ background knowledge and learning skills, but can be fixed by instructional support.

The research is important since the factors that influence learning behaviors is essential to the fundamental questions regarding education and equity.

**Modeling**

The researchers developed a learning model based on 27 datasets from Datashop with over 1.3 million student performance observations from 6946 learners in 12 different courses, ranging from all subjects, all education levels, with various education technologies and favorable learning conditions. Most questions need students to fill in the answer, perform user interface actions, or do multiple choices with a lot of choices. Only the first attempt of a problem is considered. In this process, scientific learning support is provided, including immediate feedback, explanatory instruction, observation of correct answers before moving on, tasks specific to practice cognitive competences, and repeated opportunities.

The researchers combine cognitive model, logistic regression, and a growth component to model students’ performance and success of tasks, taking initial knowledge, learning rate and learning opportunity as factors for success and using the slope of the resulting linear function to represent learning rate per learning opportunity. Initial knowledge and learning rates are further broken down to take both students and the knowledge components into consideration. The model thus results in a matrix representing cognitive model and six predictor variables: overall initial knowledge, overall learning rate, student initial knowledge, student learning rate, knowledge component initial knowledge, and knowledge component learning rate, with the learning rate variables multiplied by learning opportunities to be the growth component of this model. The KC models used generalize the knowledge component needed for a task and are selected depends on its fitness in the students’ data, that can improve students’ performance with not too high initial knowledge.

**Observation**

The model shows a decrease in the increment of learning rate from opportunity to opportunity when the learning opportunities are high (the correctness gets closer to 100%). Also, learning rate variation is relatively constant for students and higher for knowledge components.

The model can also recover high student slope variation (non-parallel lines) in simulated data with high or low student intercept variation (initial knowledge).

Averagely, students have 65% correct rate with their initial knowledge and need 7 learning opportunities to master the knowledge component.

**Result**

Between the simplified model that excludes students learning rate (AFM) and the original model(iAFM), iAFM is better for more datasets, while there are also some datasets that prefer AFM, which proved that for some situations, there is observable difference in students’ learning rate, while in some other cases not. It is also provded that the learning growth is from the knowledge component specific learning opportunities by comparing with the Time based-additive model.

Though there’s differences in initial knowledge and the number of opportunities to reach mastery, and the learning rate of knowledge components varies, their student learning rate can be very similar in the way it changes no matter the difficult level of knowledge components, especially in language learning.

Due to the phenomenon that difference in initial knowledge (different previous experiences) did not cause difference in student learning rates, disjunctive learning path hypothesis is proposed, that students may have different mental representation that produce the same outcome, especially for logical areas like math which have many ways to solve a problem. Theoretically, students can always master certain knowledge if they have enough learning opportunities with favorable learning conditions.

One limitation is that though the model emphasizes the benefit of repeated practice and high quality of feedback, it also failed to count for some forms of feedbacks.

**Flynn Fromont, Hiruna Jayamanne, and Paul Denny. 2023. Exploring the Difficulty of Faded Parsons Problems for Programming Education. In Proceedings of the 25th Australasian Computing Education Conference (ACE '23). Association for Computing Machinery, New York, NY, USA, 113–122. https://doi.org/10.1145/3576123.3576136**

**Motivation**

To investigate and quantify the effect of different fading strategies used in fading Parsons problems on students’ coding behaviors and how fading Parsons problems help students learn. This may help educators better use fading Parsons problems to train the programmers as well as providing practice resources.

**Background**

Parsons problems are considered an interesting and helpful scaffolding for novice programmers to get familiar with codes, while normal programmers may found uncomfortable due to limited solutions. Fading Parsons problem is a more complex variation of the standard Parsons problems, with some blank sections in the code blocks for students to fill in. Such problems are proved to help students learn coding and reinforce good coding habits.

The researchers use UPP, a novel open-source tool, to investigate how different fading strategies may influence the difficulty level and how students feel about such exercises. The tool enables students to drag-and-drop code blocks, fill in the blanks, and get feedback after testing the solution. There are four fading strategies included: Conditional, Variable, and Operator. The evaluation of the difficulty level is analyzed both qualitative and quantitatively: attempts, grades, log analysis, pre/post-test, time, think-aloud, and survey.

**Experiment**

Students are randomly divided into four groups, taking turns to deal with all four different fading strategies with general instructions about how to use the tool. There is also a post survey about students’ opinions on the difficult level and helpfulness of the problems and the tool, using five-point Likert scale and open-ended questions. 667 participants (about 71.6%) completed the survey.

Time taken and number of attempts are the primary performance metrics. Other relevant data are also recorded through the logs of students’ interactions with the tool. One limitation is that the way to calculated the time taken may produce outliers due to some students who leave the tool opening without really doing the task, while the extreme data (data over the 95th quartile) are deleted in the research.

**Analysis**

The data includes first attempts of students doing a certain problem. The evaluation of the difficulty level is based mainly on the attempts and time students used for a certain fading Parsons problem.

For both time and number of attempts, Kruska;-Wallis test is applied to the data set for difference in time distributions for students used to solve a certain problem, and then pairwise Mann-Whiteney-Wilcoxon tests are used to further compare the data. Benjamin & Hochberg correction method is used to adjust the p value.

There is evidence that different fading strategies do influence the difficulty level of questions. Among the three strategies, conditional fading is the most time-consuming one, taking about twice time of others, and the next is variable fading, though its difference with operator fading is not significant. All fading questions consume more time than standard Parsons problems.

For the number of attempts used, the trend is similar for all three strategies. The outliers may occur due to students trying to stress the tool. The rule observed also enhances the observation in the time data, while variable and operator fading have little differences. This is probably due to the smaller number of possible answers for Variable or Operator fading.

As the majority of post survey result support, the positive influence of fading Parsons problems on people’s coding ability, it seem to be a useful education tool that can adjust difficulty level and create large learning resources for novice programmers as well as training experiencer to view others’ solutions.

**Rivers, K., Harpstead, E., & Koedinger, K. R. (2016, September). Learning curve analysis for programming: Which concepts do students struggle with?. In ICER (Vol. 16, pp. 143-151).**

**Motivation**

To find what programming concepts students have difficulties with by analyzing programming data with education data mining methods like knowledge-based learning curve analysis. This research also explores how these learning analysis tools can further apply to programming education, in order to improve the future design of programming curricula.

**Methodology**

The learning curve analysis suppose that students’ error rate will decrease when practicing a certain skill over time. Such skills are usually considered a knowledge component and a task may contain multiple knowledge components. Additive Factors Model (AFM) is a special logical regression to predict students’ performance based on their practice opportunities, with students’ initial knowledge represented by student intercept and students’ speed to master a knowledge component represented by the slope.

In this research, knowledge components (KC) in programming are defined as the nodes on abstract syntax trees parsed from student codes and will be refined to be closer to true knowledge students learn in future work. Each submission is considered a sequence of opportunities while each opportunity corresponds to a knowledge component. Two KC models are used, one only includes data from first attempts and the other include all. The criteria for a successful application of KC is that it occurs in the correct part of the students’ program, and they are either all evaluated in each attempt or only measure the ones changed from attempt to attempt.

**Experiment**

The subjects are from two introductory programming courses in Carnegie Mellon University in Spring 2016. The exercise set contains 40 Python problems with multiple topics. Feedback is provided through submission. Data is collected from 89 students in the classes including 2907 submissions and 380 hint requests, resulting in 3287 states over 40 problems.

**Analysis**

Tools like AST library of Python and ITAP’s path construction methodology are used to adjust the data for Datashop to generate learning curve analysis by performing AFM. The data points with fewer than 9 students will be excluded.

Among the four models divided by all-attempts V.S. first attempt and all-steps V.S. modified steps, first-attempts models are a better fit and the modified-step model is closer to traditional ITS model, though the two step classified models are equally good.

To explore why there’s no reduction in error rate as expected, the set of AST node KCs are categorized into Little-data, Already-learned, No-learning, Still-learning, and Good-learning according to the learning curve trend. Among the curves of these KCs, some concepts did not gain enough practice, some show no learning, while some do show learning pattern. Some KCs show unusual cases, e.g., function definition KC seems to be different from other KC.

**Limitations**

The ways to use AFM and learning curve analysis in the research are not expected by these tools any may cause bias since the programming education is different with the traditional intelligent tutoring systems that the concepts can be separately taught, while in programming, old knowledge component will be used to build new knowledge component.

The definition of KC is also a rough approximation of true KC. The choice of students’ states may also not been optimized. And some other factors like students’ design decisions are not considered. Future work may be needed to improve the model e.g. by combining some AST nodes to form KC or use Datashop to build more semantic KC models and add test items observe whether the model can properly represent students’ learning states.

**Discovering, Autogenerating, and Evaluating Distractors for Python Parsons Problems in CS1**

**Motivation**

To construct and test a methodology for collecting distractor templates and automating the construction of Parsons problems with these templates and figure out the influence of distractors on students’ performance.

**Background**

Distractors refer to the code blocks containing error or are not used in final solution in Parsons problems. They are supposed to reflect the programming error and misconceptions that students may have. The design of the distractor should be plausible but incorrect and be able to provide high quality feedback about where students’ difficulties are to the teachers. There have been ways to collect proper designs of distractors, such as using common mistakes in related questions. In recent designs, distractors are grouped with related correct code block to regain the efficiency.

The researchers aim to create a set of distractor templates.

**Experiment**

Submissions from 2853 students for each of 42 questions in the set. The submissions are parsed into abstract syntax trees with some part with un-parsable grammar mistakes excluded and finally leave over 5000 submissions. The researchers began with manually reviewing mistakes in all submissions to construct an AST matching all submissions of certain errors. Then a partial AST is created for each correct statement so that such statements in the solutions can be detected and matched with the associated distractor template.

For each of the questions, three solutions and two Parsons problems are generated, one with distractor while one has. The distractors are selected from the error categories analyzed to generate the distractor template. These problems are finally included in the final exam of a Python introduction course containing 494 students. Each student was randomly assigned to one of the six Parsons questions and the score they gain would decrease according to attempts they used and numbers of correct blocks and indentation. The errors are denoted with different color.

**Analysis**

As the data is not normally distributed, Mann- Whitney U tests are used for analysis. Generally, students spend more time on questions with distractors, but there’s no significant influence on their score. Though the distractors in Parsons problems seems not helpful in distinguishing students’ performance, this may be because students are familiar with the list statements since it was final exam.

The distractors can also be used to allocate the difficulties that students have by including Parsons problems in formative assessments.

**Limitation and Future work**

There was a problem in the research that it used a less common mistake as distractor, while it may have minimal effect on efficiency, scores, or item discrimination. Other limitations may be that the current distractor templates are limited to single statements which means loops and if-conditions are not included. Also, the number of problems included in the research is relatively small.

For future work, adding the Parsons problems containing distractors in homework or other tasks during students’ learning process may be helpful to figure out whether distractors help distinguish students’ performance and increase the item discrimination of the problem. Influence and effect of implementing these problems in other times throughout the semester should also be researched on to find when it is used most effectively, as well as the influence of distractors on learning performance. Since the distractors are visually paired with the correct code, future work should take this into consideration and research on the influence of distractor on performance and item discrimination.