Construction and Preliminary Validation of a Dynamic Programming Concept Inventory

Matthew Ferland*
Dickinson College
Carlisle, PA, USA
ferlandm@dickinson.edu

Varun Nagaraj Rao*
Princeton University
Princeton, NJ, USA
varunrao@princeton.edu

Arushi Arora
University of California, Irvine
Irvine, CA, USA
arushia2@uci.edu

Drew van der Poel
Northeastern University
Boston, USA
a.vanderpoel@northeastern.edu

Michael Luu
University of California, Irvine
Irvine, CA, USA
luum6@uci.edu

Randy Huynh
University of California, Irvine
Irvine, CA, USA
randylh@uci.edu

Freddy Reiber
Boston University
Boston, MA, USA
freddyr@bu.edu

Sandra Ossman
University of California, Irvine
Irvine, CA, USA
sossman@uci.edu

Seth Poulsen
Utah State University
Logan, UT, USA
seth.poulsen@usu.edu

Michael Shindler
University of California, Irvine
Irvine, CA, USA
mikes@uci.edu

Abstract

Concept inventories are standardized assessments that evaluate student understanding of key concepts within academic disciplines. While prevalent across STEM fields, their development lags for advanced computer science topics like dynamic programming (DP)—an algorithmic technique that poses significant conceptual challenges for undergraduates. To fill this gap, we developed and validated a Dynamic Programming Concept Inventory (DPCI). We detail the iterative process used to formulate multiple-choice questions targeting known student misconceptions about DP concepts identified through prior research studies. We discuss key decisions, tradeoffs, and challenges faced in crafting probing questions to subtly reveal these conceptual misunderstandings. We conducted a preliminary psychometric validation by administering the DPCI to 172 undergraduate CS students finding our questions to be of appropriate difficulty and effectively discriminating between differing levels of student understanding. Taken together, our validated DPCI will enable instructors to accurately assess student mastery of DP. Moreover, our approach for devising a concept inventory for an advanced theoretical computer science concept can guide future efforts to create assessments for other under-evaluated areas currently lacking coverage.

*Both authors contributed equally to this research.

CCS Concepts

• **Social and professional topics → Computer science education**; **Student assessment**; **Model curricula**.

Keywords

dynamic programming; algorithms education; concept inventory

1 Introduction

Concept inventories are standardized assessment instruments used in education research to evaluate student understanding of key concepts within a discipline. They are typically comprised of multiple-choice questions targeting common misconceptions identified. These misconceptions are derived via education research. Concept inventories have been developed and validated across many science, technology, engineering, and math (STEM) topics. One of the first and most widely adopted was the Force Concept Inventory (FCI) in physics education [15, 19]. The FCI was devised based on studies of prevalent misconceptions physics students held regarding Newtonian mechanical concepts. It has become a standard instrument for measuring the effectiveness of high school and university physics instruction on improving the conceptual understanding of Newtonian mechanics. Following the FCI's lead, additional concept

inventories have been created across other natural science disciplines as well as engineering, including, chemistry [27], astronomy [4], biology[3], mathematics [12], statistics[36], and geosciences [22], among others [2].

Within computer science, concept inventories have been formulated for introductory CS1/CS2 courses [9, 14], architecture [32], operating systems [38], and cybersecurity [34]. However, concept inventory development remains lacking for more theoretical and advanced CS topics. An unvalidated concept inventory focused on algorithms and data structures was proposed by Paul and Vahrenhold [31], but no rigorously validated instrument currently exists for the challenging concept of dynamic programming (DP).

DP is an algorithm design technique, taught in many undergraduate algorithms courses, that builds solutions to problems using optimal substructure and overlapping subproblems. As an advanced algorithmic approach, DP is considered among the most conceptually difficult topics for CS students [11, 35, 40]. Prior work by Shindler et al. [35] examined prevalent student misconceptions related to dynamic programming, suggesting the need for a concept inventory to assess student understanding. However, despite its reputation as a challenging topic fraught with subtleties students struggle to grasp, and the cataloged misunderstandings they harbor, no concept inventory has yet been formulated and validated to evaluate dynamic programming comprehension.

This paper discusses the development and initial validation of a research-based Dynamic Programming Concept Inventory (DPCI). The DPCI can be deployed by instructors to measure effectiveness of teaching DP to undergraduate computer science majors. We detail the iterative process of crafting valid DPCI multiple choice questions which reveal the presence of student misconceptions identified by prior education studies. Carefully formulating MCQs to subtly tease out conceptual misunderstandings around the intricacies of dynamic programming is difficult and time intensive. We discuss key steps, decisions, tradeoffs and consequences, challenges encountered in producing a high quality concept inventory and preliminary psychometric validation administering our DPCI to 172 undergraduate computer science students.

Our validated DPCI will enable instructors to evaluate student mastery of dynamic programming within undergraduate algorithms courses. Moreover, our experiences devising a concept inventory to measure understanding of an advanced conceptual CS topic will inform development of additional assessments for theoretical areas of computer science presently lacking coverage. With a broad foundation of validated concept inventories across the CS curriculum, instructors can better evaluate the efficacy of pedagogical techniques and curricular interventions. Our contributions include: (i) steps involved in constructing a validated DPCI, (ii) publishing a list of questions to be administered as part of the DPCI,[1] and (iii) psychometric validation results using classical test theory. Our study was approved as exempt by the IRB.

2 Related Work

2.1 Concept Inventories

A *concept inventory (CI)* is a validated assessment for a topic or set of topics that allows researchers and instructors to measure

what students have learned. Concept inventory research started with the Force Concept inventory, which allowed for measuring effectiveness of pedagogical techniques against one another in introductory physics classes, helping build the case for more active learning in these courses [15, 19].

Recently, computing education researchers have been creating CIs, so our discipline can also benefit from them. Despite their relatively recent introduction in computer science, concept inventories have already been useful for many purposes in computing education research. Introductory programming concept inventories have been used to examine the relationship between spatial ability and learning programming [7], to evaluate the effectiveness of teaching students using both block- and text-based programming languages [6], to understand the impact of students' educational background on learning topics in computer science [5], and to evaluate novel instructional practices [28, 39]. Concept inventories for other topics in CS have been used to compare outcomes between digital logic courses which use differing pedagogical approaches [16], to measure misconceptions of students learning algorithm analysis [13], and to evaluate the effectiveness of ways of teaching about the memory model of the Rust programming language [8]. This progress in CS education afforded by the creation and existence of concept inventories gives a strong argument for their continued development.

Many concept inventories have been created and fully validated for many introductory programming topics, with validated CIs for CS1, first year computer science, middle-grades computer science, object oriented programming, and basic data structures as well as topic-specific CIs for recursion and Java arrays [2].

Theoretical and advanced topics in computer science have not yet received as much coverage in concept inventories. There are fully validated concept inventories only for digital logic and cybersecurity [17, 18, 34]. Concept inventory work has begun, but not been completed for operating systems, algorithm analysis, and compiler construction [2]. This leaves huge gaps in many theoretical and upper-level computing topics regarded as core topics in the ACM curricular guidelines, including all types of algorithm design as well as human-computer interaction, artificial intelligence, networking, and parallel and distributed computing [20]. Thus, we find it important that concept inventories are developed for more advanced and theoretical computing topics, to drive the same educational innovation and research that has happened for introductory programming. For extensive reviews of assessment instruments used in computing education research, see [2, 10, 26].

2.2 Work on Algorithms Education

While many students begin learning algorithms as a part of their introductory course sequence, the ACM curricular guidelines [20] recommend that they take a more advanced algorithms-focused course later on in their curriculum. Such courses almost always cover dynamic programming [25]. Dynamic programming is the area of algorithms education with the most research about it [24].

Though there is a draft concept inventory for algorithm analysis [13], there have not been any concept inventories created to cover the broad space of algorithm design techniques, including but not limited to greedy algorithms, divide-and-conquer algorithms,

---

[1]DPCI link: https://us.prairielearn.com/pl/public/course/5595/questions

algorithmic reductions, and dynamic programming (DP) algorithms. Many of these areas of algorithms design are so understudied that creating a concept inventory would not yet be possible until researchers first complete the foundational work of discovering common students misconceptions and struggles in those areas [24]. On the other hand, studies have been done to understand the nature of student difficulties in dynamic programming [23, 35, 40], making it a good topic to design a concept inventory around at this point in time. Specifically, prior studies have highlighted that students often fail to recognize that DP is the best solution to a problem, and once they do attempt a DP solution, they often fail to select the correct base case or determine the proper overlapping subproblems.

3 Methods

When developing a Concept Inventory, researchers typically follow a predefined set of steps established in previous research on the topic. For our study, we followed the methodology outlined by [1, 37]. In general, concept inventory studies typically follow a five-step pattern:

(1) Identify the topics to be addressed in the concept inventory.
(2) Establish common misconceptions held by students in the specific course or subject area.
(3) Create questions designed to identify these misconceptions.
(4) Conduct a validation process in which instructors assess whether the questions effectively identify the presence of misconceptions in the test-takers.
(5) Revise the inventory questions and repeat step 4 as necessary.

Creating an effective concept inventory is a meticulous and time-consuming process that demands a high level of attention to detail. Therefore, our concept inventory must concentrate on fundamental topics, rather than attempting to encompass the entirety of the overarching subject. Given that the assessment is designed to be completed within roughly thirty minutes, it is important to address only a select subset of topics. This ensures that test-takers have enough time to complete the assessment.

Most concept inventories encompass a wide range of topics. In our case, we chose to focus on dynamic programming and thus completed step one of the process. Furthermore, step two of the process, establishing common misconceptions about dynamic programming has been completed by [40] and replicated by [35]. Additionally, we have completed step three by creating questions to identify the misconceptions highlighted by our prior work.

4 Construction of the DPCI

We attempted to write problems based on Shindler et al. [35], but found it difficult. The problems identified in the paper were primarily about what students did that was incorrect, rather than the student beliefs that caused them to make these mistakes. With IRB approval, we were able to obtain a copy of the 64 anonymized transcripts used in that study, and relabel them to be focused on what student misconceptions were present. We then created an initial set of questions. Then, after testing with students, several questions were removed from the set.

Number	Description
1	2D DP solutions always traverse top left to bottom right
2	DP always involves minimization or maximization
3	If a problem has minimization or maximization, it must be possible to efficiently solve it with DP
4	If a problem has minimization or maximization, there is no DP solution for it
5	Conflating recursion with DP
6	Memoization is not a DP approach
7	Aspects of greedy problems are mistakenly associated with DP
8	Ignoring specific subproblems (and knowing which ones to solve)
9	Polynomial number of subproblems indicates exponential runtime
10	DP involves "non-adjacency" of subproblems
11	Every problem has an efficient DP solution
12	DP must memoize every permutation of decisions made in a problem
13	DP involves tracking aspects of state to remember how a solution was constructed
14	DP always requires a 2D array
15	DP requires nested for loops, with the inner loop solving a recurrence

Table 1: List of misconceptions found during re-analysis of interview data, explanations of what they mean, and (maybe) examples.

4.1 Reanalysis of Interview Data

We examined the interview transcripts to search for misconceptions. So long as a misconception was seen at least once, it was labeled as a misconception, and the numbers found were not quantified. The idea was that uncommon misconceptions would prove to create easy questions that could later be removed.

Anecdotally, the most common misconceptions at this stage were misconceptions 2, 5, and 14.

4.2 Construction of Questions to Match Misconceptions

As previously mentioned, initial work on the questions happened before the relabeling that resulted from re-analysis of the interview data. As such, some questions were left over from this initial stage, and were later relabeled to match the updated misconception numbers.

The rest of the questions, however, were generated after the misconception list was identified. First, each author was given full freedom to create as many questions as they wished, targeting at least one misconception identified. Simultaneously, the old questions were relabeled, as mentioned previously, and then removed if they had no match. Then, we identified which misconceptions lacked any questions, and made a question for each of them.

After testing, some questions were rewritten, seperated into multiple parts, or removed, leaving the current questions remaining.

Question ID	Misconception ID	Answer Choice for misconception	% who Chose UCI/USU
VR5	1	A	12 / 14
DV14.1	2	B	— / 46
AA2	3	D	14/ 9
VR2	4	A	5 / 7
AA2	5	B	7 / 7
MF3	5	A, B, or D	16 / 39
DV14.3	7	True	— / 20
DV6	8	Not A	13 / 37
MF6	11	A	37 / 31
AA2	11	A	3 / 4
ML2	13	A	7 / 30
DV10	14	A or B	10 / 5
VR2	14	B	9 / 3
DV10	15	A or C	8 / 7

Table 2: Frequency at which misconceptions were chosen. The four questions labeled 'other' are questions left over from the initial stage that don't map to one of our core misconceptions, but we found useful to gauge student difficulty with dynamic programming. Some questions appear in multiple rows; this indicates that were were different incorrect answer choices that mapped to different misconceptions.

4.3 What we can and cannot test with Multiple Choice

While multiple choice questions are well-suited for probing conceptual understanding, there are limitations in what they can effectively assess when it comes to dynamic programming. Our DPCI focuses on evaluating mastery of the core principles and requisite reasoning about optimal substructure and overlapping subproblems. However, we decided against including questions that attempt to test other skills like constructing recurrence relations or transition functions. Such procedural tasks are better evaluated through other formats like write-out problems or coding exercises.

4.4 Pseudocode

Nine of the questions on the DPCI ask students to answer questions about snippets of code. For our code snippets, we used a pseudocode which was designed to be understandable by students familiar with any common imperative programming language, similar to the approach taken by the SCS1 and BDSI [30, 33]. We included a pseudocode guide defining the language constructs at the beginning of the assessment.[2] Our pseudocode differed from the SCS1 pseudocode in a few key ways. Because we had no need for object oriented constructs, we did not include these in the language. On the other hand, we found that for defining algorithms in a clear manner, it was useful to include various built-in functions such as `min`, `max`, `len`, and `floor`, which have roughly the same notation as their counterparts in Python. We defined these functions and

---

[2]Links to the pseudocode guide and DPCI questions are omitted for anonymouse review but will be included in the final version.

their asymptotic performance behavior as a part of the pseudocode guide.

4.5 Outside Expert Review

We had a draft of our questions by the end of January. We prepared a draft of the questions on PrairieLearn, which would offer both the question and prompt for feedback on each. We asked several experts to review the questions and provide commentary. Based on their feedback, we made many changes described below.

Most revisions primarily focused on correcting syntax errors to adhere to our pseudocode guide, refining time complexities, clarifying ambiguities within questions, and ensuring accurate variable assignments. Feedback from experts also helped us to ensure the questions focused on testing conceptual knowledge rather than just debugging code. We also reworded some questions so that students could answer them correctly whether or not their instructor had taught them that recursion with memoization is a valid form of Dynamic Programming (along with this, we are no longer attempting to measure Misconception 6, since not all instructors agree that it is a misconception).

We also had feedback that the concept inventory didn't cover certain classes of problems, such as dynamic programming on graph structures, or problems where the subproblems are not defined as a suffix or prefix. We felt that it was appropriate to omit these advanced problem types in the interest of focusing on assessing basic dynamic programming concepts. Future work could create a concept inventory for more advanced topics. Finally, we also had feedback that the DPCI did not have questions asking students to create a recurrence for a DP problem, or perform some other technical skill related to DP. We feel that this is appropriate, since as a concept inventory the DPCI is designed to test conceptual knowledge, not necessarily technical skill. Creating assessment questions to test these skills is another avenue for future work in DP assessment.

5 Preliminary Psychometric Validation

5.1 Methods

Two primary psychometric validation methods, Classical Test Theory (CTT) and Item Response Theory (IRT), have been used in prior work on concept inventories [21, 29, 33]. CTT is utilized when sample sizes are smaller and rapid iteration is necessary. In contrast, IRT is applied under conditions of larger datasets and far more reliable, as it offers robust estimations less influenced by sample specifics. We opted for CTT due to its practicality with our limited sample size and as an initial testing of an assessment instrument.

Reliability within CTT is established using the statistical measure of Cronbach's α, which reflects the internal consistency of the assessment. A Cronbach's α close to 1 indicates high reliability, with values above 0.8 considered good and above 0.7 satisfactory [21].

To assess question validity, we also calculate difficulty and discrimination. Difficulty is defined as the proportion of examinees who answer a question correctly, with an ideal range from 0.2 to 0.8, ensuring a balanced challenge. Discrimination is measured by the point-biserial correlation between a questions score and the total test score [21], with values above 0.2 indicating effective discrimination between differing levels of student understanding.

5.2 Data Collection

Students in the algorithms course at UC Irvine were provided the DPCI as an optional practice exam. The course covered dynamic programming in class for two weeks, and the DPCI optional practice exam was made available three days before an in-class quiz on the topic. Of the 316 students in the class, 109 completed the exam, and of those, 93 agreed to allow us to use their data for research. That quarter, students were expected to complete 19 warm-up activities before various lecture meetings as part of their grade; students who took the DPCI practice exam, regardless of whether or not their data was used for research, were allowed to drop one warm-up activity from their course grade.

A few weeks later after some minor revisions to the DPCI questions (detailed in Section 5.3.2), students at Utah State University were provide the DPCI as homework assignment to complete (graded only on completion) before their midterm on Dynamic Programming. Out of the 73 students in the class, 63 completed the exam, and the terms of our IRB allowed us to use all of their data.

5.3 Results

5.3.1 Round 1. Our analysis of question difficulties revealed a range from 0.20 to 0.84, with the exception of item Q3, which presented a notably lower difficulty of 0.09 (see Table 3). A lower difficulty score indicates a more challenging question, and while most items fell within the accepted difficulty range of 0.20 to 0.80, DV14 was significantly more difficult. This variation in difficulty levels suggests that the DPCI offers a balanced mix of easy, medium, and hard questions, catering to students with varying proficiency in dynamic programming concepts.

The discrimination values ranged from 0.28 to 0.56, indicating that most questions effectively differentiate between students of differing skill levels (see Table 3). However, questions DV13 and DV14 were outliers, with discrimination values of -0.08 and 0.13, respectively, falling outside the recommended minimum of 0.20. Except for these two questions, the discrimination values provide substantial evidence that higher question scores correlate positively with overall proficiency as measured by total performance on the assessment.

We obtained a Cronbach's α = 0.76. The Cronbach's α after removing questions DV13 and DV14 was 0.78. This value is close to Jorion et al. [21] recommendation ($\geq$ 0.8) for good reliability and is strong when compared to published values of other CIs.

5.3.2 Revisions based on Round 1. We made several modifications to questions in the CI based on our initial round of validation. Our primary focus was addressing the two problems with discrimination values below 20%.

Problem DV14, the most difficult problem and the second least discriminating, was split into three new problems. Originally a "select all" style problem with 5 possible choices, we determined it was likely this problem had such a low correctness score simply due to the binary grading scheme. Even if a student correctly selected/didn't select four of the five choices, they were still marked with a zero. To rectify this, we opted to split the problem into three new problems, with each new problem focused on a particular theme (e.g. DP and optimization). The hope was that this would

allow us to determine which (if any) of the themes of the problem were causing such a high difficulty metric.

For problem DV13, which was the second most difficult problem and had a negative discrimination value, we simplified the problem by only using asymptotic analysis in the latter part of the implication. We also referred explicitly to the number of elements filled in in the DP table during the course of the algorithm, as opposed to the asking about its asymptotic runtime.

In addition to our edits based on the student metrics, we also made several changes based on student feedback (gathered through comment boxes on PrairieLearn). These included ensuring consistent capitalization of variable names and adding clarifying details.

5.3.3 Round 2. The results of the second round of validation can also be seen in Table 3. The results were largely similar, with some variation that is expected when using the same questions on a different student population from a different university. While the newly created questions DV14.1 and DV14.3 performed well, DV13 and DV14.2 had very poor discrimination measures.

For the second round, we obtained a Cronbach's α = 0.76. The Cronbach's α after removing questions DV13 and DV14.1 was also 0.76. This value is similar to that previously obtained, close to Jorion et al. [21] recommendation for good reliability and is strong when compared to published values of other CIs.

Q	Round 1		Round 2		Q	Round 1		Round 2	
	Dif.	Dis.	Dif.	Dis.		Dif.	Dis.	Dif.	Dis.
MF1	0.46	0.31	0.48	0.33	DV4	0.80	0.48	0.81	0.43
MF3	0.83	0.28	0.62	0.41	DV5	0.70	0.57	0.73	0.47
MF6	0.62	0.29	0.68	0.35	DV6	0.70	0.56	0.64	0.36
AA2	0.76	0.52	0.78	0.29	DV7	0.31	0.42	0.29	0.32
AA3	0.46	0.46	0.44	0.49	DV8	0.27	0.40	0.27	0.37
ML2	0.53	0.37	0.48	0.46	DV9	0.24	0.31	0.40	0.39
RH1	0.25	0.34	0.41	0.38	DV10	0.84	0.53	0.91	0.65
VR2	0.74	0.43	0.86	0.49	DV11	0.81	0.40	0.82	0.68
VR3.2	0.47	0.46	0.33	0.31	DV12	0.59	0.45	0.70	0.30
VR2	0.40	0.38	0.43	0.16	DV13	0.20	-0.08	0.21	0.09
VR5	0.83	0.34	0.81	0.38	DV14	0.09	0.13	—	—
DV1	0.63	0.40	0.59	0.34	DV14.1	—	—	0.13	0.40
DV2	0.35	0.43	0.17	0.37	DV14.2	—	—	0.81	0.11
DV3	0.75	0.46	0.82	0.51	DV14.3	—	—	0.79	0.37

Table 3: Difficulty and Discrimination numbers for data collection at UC Irvine (Round 1) and Utah State University (Round 2). Most of our questions fall within the preferred range of discrimination greater than 0.2 and difficulty between 0.2 and 0.8.

6 Discussion

6.1 Overall Question Quality

Overall, our questions performed well with 9 out of the 15 misconceptions being selected by 40% or more of the students. However, as the results indicated, DV13 and DV14.2 continued to perform poorly even after revisions. Consequently, we removed these questions from the concept inventory, as they proved to be outliers.
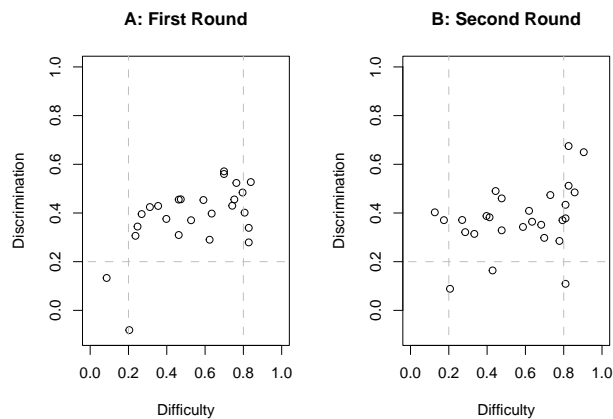
Figure 1: Difficulty and discrimination for both rounds of validation.

6.2 Evidence that the Misconceptions Are Measured by the Questions

Misconceptions chosen by 15% or more of the students provide strong evidence that the misconception both targets and accurately measures the intended misunderstanding, thus validating our hypothesis about its prevalence. Highly prevalent misconceptions were:

- Misconception 2: DP always involves minimization or maximization
- Misconception 5: Conflating recursion with DP
- Misconception 7: Aspects of greedy problems are mistakenly associated with DP
- Misconception 8: Ignoring specific subproblems (and knowing which ones to solve)
- Misconception 11: Every problem has an efficient DP solution
- Misconception 13: DP involves tracking aspects of state to remember how a solution was constructed

Low-prevalence misconceptions (less than 15%) may suggest questions that fail to present the misconception clearly, or that our hypothesized misconception is less widespread than anticipated. Low-prevalence misconceptions were:

- Misconception 1: 2D DP solutions always traverse top left to bottom right
- Misconception 3: If a problem has minimization or maximization, it must be possible to efficiently solve it with DP
- Misconceptions 4: If a problem has minimization or maximization, there is no DP solution for it
- Misconception 14: DP always requires a 2D array
- Misconceptions 15: DP requires nested for loops, with the inner loop solving a recurrence

By analyzing selection rates across all categories, we evaluate each question's construct validity, assessing its ability to measure the intended misconception genuinely. This analysis helps guide refinements to ensure our concept inventory reliably identifies and distinguishes between various misconceptions.

7 Conclusions and Future Work

In this work, we detail our process of developing and validating the first Dynamic Programming Concept Inventory (DPCI). We administered the DPCI to 172 undergraduate computer science students across two large public universities in the U.S., and our psychometric validation revealed that the majority of difficulty and discrimination values across questions fell within acceptable parameters. This indicates that the questions part of the DPCI were of appropriate difficulty and effectively distinguished between students of varied levels of conceptual understanding.

Future research could extend the DPCI's validation across additional universities, broadening our understanding of student misconceptions in dynamic programming. This initial step in identifying misconceptions will inform future research on targeted pedagogical interventions. Ultimately, by identifying and evaluating these interventions' impact on learning outcomes, we can refine instructional methodologies to enhance student comprehension of dynamic programming concepts.

Acknowledgements

References

[1] Wendy K Adams and Carl E Wieman. 2011. Development and validation of instruments to measure learning of expert-like thinking. *International journal of science education* 33, 9 (2011), 1289–1312.

[2] Murtaza Ali, Sourojit Ghosh, Prerna Rao, Raveena Dhegaskar, Sophia Jawort, Alix Medler, Mengqi Shi, and Sayamindu Dasgupta. 2023. Taking Stock of Concept Inventories in Computing Education: A Systematic Literature Review. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1* (Chicago, IL, USA) *(ICER '23)*. Association for Computing Machinery, New York, NY, USA, 397–415. https://doi.org/10.1145/3568813.3600120

[3] Dianne L Anderson, Kathleen M Fisher, and Gregory J Norman. 2002. Development and evaluation of the conceptual inventory of natural selection. *Journal of research in science teaching* 39, 10 (2002), 952–978.

[4] Erin M Bardar, Edward E Prather, Kenneth Brecher, and Timothy F Slater. 2007. Development and validation of the light and spectroscopy concept inventory. *Astronomy Education Review* 5, 2 (2007), 103–113.

[5] Yifat Ben-David Kolikant and Sara Genut. 2017. The effect of prior education on students' competency in digital logic: the case of ultraorthodox Jewish students. *Computer Science Education* 27, 3-4 (2017), 149–174.

[6] Jeremiah Blanchard, Christina Gardner-McCune, and Lisa Anthony. 2020. Dual-modality instruction and learning: a case study in CS1. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 818–824. https://doi.org/10.1145/3328778.3366865

[7] Ryan Bockmon, Stephen Cooper, William Koperski, Jonathan Gratch, Sheryl Sorby, and Mohsen Dorodchi. 2020. A CS1 spatial skills intervention and the impact on introductory programming abilities. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 766–772. https://doi.org/10.1145/3328778.3366829

[8] Will Crichton, Gavin Gray, and Shriram Krishnamurthi. 2023. A Grounded Conceptual Model for Ownership Types in Rust. *Proceedings of the ACM on Programming Languages* 7, OOPSLA2 (2023), 1224–1252.

[9] Holger Danielsiek, Wolfgang Paul, and Jan Vahrenhold. 2012. Detecting and understanding students' misconceptions related to algorithms and data structures. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education.* 21–26.

[10] Adrienne Decker and Monica M. McGill. 2019. A topical review of evaluation instruments for computing education. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 558–564. https://doi.org/10.1145/3287324.3287393

[11] Emma Enström and Viggo Kann. 2017. Iteratively intervening with the "most difficult" topics of an algorithms and complexity course. *ACM Transactions on Computing Education (TOCE)* 17, 1 (2017), 1–38.

[12] Jerome Epstein. 2007. Development and validation of the Calculus Concept Inventory. In *Proceedings of the ninth international conference on mathematics education in a global community*, Vol. 9. Citeseer, 165–170.

[13] Mohammed F Farghally, Kyu Han Koh, Jeremy V Ernst, and Clifford A Shaffer. 2017. Towards a concept inventory for algorithm analysis topics. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 207–212.

[14] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey L Herman, Lisa Kaczmarczyk, Michael C Loui, and Craig Zilles. 2010. Setting the scope of concept inventories for introductory computing subjects. *ACM Transactions on Computing Education (TOCE)* 10, 2 (2010), 1–29.

[15] Richard R. Hake. 1998. Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American Journal of Physics* 66, 1 (1998), 64–74. https://doi.org/10.1119/1.18809

[16] Geoffrey L Herman and Joseph Handzik. 2010. A preliminary pedagogical comparison study using the digital logic concept inventory. In *2010 IEEE Frontiers in Education Conference (FIE)*. IEEE, F1G–1.

[17] Geoffrey L Herman, Shan Huang, Peter A Peterson, Linda Oliva, Enis Golaszewski, and Alan T Sherman. 2023. Psychometric Evaluation of the Cybersecurity Curriculum Assessment. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 228–234.

[18] Geoffrey L Herman, Craig Zilles, and Michael C Loui. 2014. A psychometric evaluation of the digital logic concept inventory. *Computer Science Education* 24, 4 (2014), 277–303.

[19] David Hestenes, Malcolm Wells, and Gregg Swackhamer. 1992. Force concept inventory. *The Physics Teacher* 30, 3 (1992), 141–158. https://doi.org/10.1119/1.2343497

[20] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Association for Computing Machinery, New York, NY, USA.

[21] Natalie Jorion, Brian D Gane, Katie James, Lianne Schroeder, Louis V DiBello, and James W Pellegrino. 2015. An analytic framework for evaluating the validity of concept inventory claims. *Journal of Engineering Education* 104, 4 (2015), 454–496.

[22] Julie C Libarkin and Steven W Anderson. 2005. Assessment of learning in entry-level geoscience courses: Results from the Geoscience Concept Inventory. *Journal of Geoscience Education* 53, 4 (2005), 394–401.

[23] Jonathan Liu, Erica Goodwin, and Diana Franklin. 2025. Student Utilization of Metacognitive Strategies in Solving Dynamic Programming Problems. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education* (Pittsburgh, Pennsylvania, USA) *(SIGCSE '25)*. Association for Computing Machinery, New York, NY, USA, 6 pages.

[24] Jonathan Liu, Seth Poulsen, Erica Goodwin, Hongxuan Chen, Grace Williams, Yael Gertner, and Diana Franklin. 2024. Teaching Algorithm Design: A Literature Review. arXiv:2405.00832 [cs.DS]

[25] Michael Luu, Matthew Ferland, Varun Nagaraj Rao, Arushi Arora, Randy Huynh, Frederick Reiber, Jennifer Wong-Ma, and Michael Shindler. 2023. What is an Algorithms Course? Survey Results of Introductory Undergraduate Algorithms Courses in the U.S.. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. ACM, 284–290.

[26] Lauren Margulieux, Tuba Ayer Ketenci, and Adrienne Decker. 2019. Review of measurements used in computing education research and suggestions for increasing standardization. *Computer Science Education* 29, 1 (Jan. 2019), 49–78. https://doi.org/10.1080/08993408.2018.1562145 Publisher: Routledge _eprint: https://doi.org/10.1080/08993408.2018.1562145.

[27] Douglas R Mulford and William R Robinson. 2002. An inventory for alternate conceptions among first-semester general chemistry students. *Journal of chemical education* 79, 6 (2002), 739.

[28] Greg L Nelson, Benjamin Xie, and Amy J Ko. 2017. Comprehension first: evaluating a novel pedagogy and tutoring system for program tracing in CS1. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*. 2–11.

[29] Spencer Offenberger, Geoffrey L Herman, Peter Peterson, Alan T Sherman, Enis Golaszewski, Travis Scheponik, and Linda Oliva. 2019. Initial validation of the cybersecurity concept inventory: pilot testing and expert review. In *2019 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–9.

[30] Miranda C Parker, Mark Guzdial, and Shelly Engleman. 2016. Replication, validation, and use of a language independent CS1 knowledge assessment. In *Proceedings of the 2016 ACM conference on international computing education research*. 93–101.

[31] Wolfgang Paul and Jan Vahrenhold. 2013. Hunting high and low: Instruments to detect misconceptions related to algorithms and data structures. In *Proceeding of the 44th ACM technical symposium on Computer science education*. 29–34.

[32] Leo Porter, Saturnino Garcia, Hung-Wei Tseng, and Daniel Zingaro. 2013. Evaluating student understanding of core concepts in computer architecture. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. 279–284.

[33] Leo Porter, Daniel Zingaro, Soohyun Nam Liao, Cynthia Taylor, Kevin C Webb, Cynthia Lee, and Michael Clancy. 2019. BDSI: A validated concept inventory for basic data structures. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*. 111–119.

[34] Seth Poulsen, Geoffrey L Herman, Peter AH Peterson, Enis Golaszewski, Akshita Gorti, Linda Oliva, Travis Scheponik, and Alan T Sherman. 2021. Psychometric evaluation of the cybersecurity concept inventory. *ACM Transactions on Computing Education (TOCE)* 22, 1 (2021), 1–18.

[35] Michael Shindler, Natalia Pinpin, Mia Markovic, Frederick Reiber, Jee Hoon Kim, Giles Pierre Nunez Carlos, Mine Dogucu, Mark Hong, Michael Luu, Brian Anderson, et al. 2022. Student misconceptions of dynamic programming: a replication study. *Computer Science Education* 32, 3 (2022), 288–312.

[36] Andrea Stone, Kirk Allen, Teri Reed Rhoads, Teri J Murphy, Randa L Shehab, and Chaitanya Saha. 2003. The statistics concept inventory: A pilot study. In *33rd Annual Frontiers in Education, 2003. FIE 2003.*, Vol. 1. IEEE, T3D–1.

[37] Cynthia Taylor, Michael Clancy, Kevin C. Webb, Daniel Zingaro, Cynthia Lee, and Leo Porter. 2020. The Practical Details of Building a CS Concept Inventory. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 372–378. https://doi.org/10.1145/3328778.3366903

[38] Kevin C Webb and Cynthia Taylor. 2014. Developing a pre-and post-course concept inventory to gauge operating systems learning. In *Proceedings of the 45th ACM technical symposium on Computer science education*. 103–108.

[39] Benjamin Xie, Greg L Nelson, and Amy J Ko. 2018. An explicit strategy to scaffold novice program tracing. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 344–349.

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