THE UNIVERSITY OF CHICAGO

WHAT DO WE KNOW ABOUT TEACHING ALGORITHM DESIGN?

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES DIVISION
IN CANDIDACY FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY
JONATHAN LIU

CHICAGO, ILLINOIS

TABLE OF CONTENTS

ABSTRACT

Algorithm design is a vital skill developed in most undergraduate CS programs, but few research studies focus on pedagogy related to algorithms coursework. To understand the work that has been done in the area, we present a systematic survey and literature review of CS Education studies. We find a broad sparsity of papers which indicates that many open questions remain about teaching algorithm design. We also note the need for papers using rigorous research methods.

We then synthesize the results of the existing literature to give insights into what the corpus reveals about how we should teach algorithms. Broadly, we find that much of the literature explores implementing well-established practices, such as active learning or automated assessment, in the algorithms classroom. However, there are algorithms-specific results as well: a number of papers find that students may underutilize certain algorithmic design techniques, and studies describe a variety of ways to select algorithms problems that increase student engagement and learning.

The results we present, along with the publicly available set of papers collected, provide a detailed representation of the current corpus of CS Education work related to algorithm design and can orient further research in the area.

CHAPTER 1

INTRODUCTION

The study of algorithms is widely regarded as a critical part of an undergraduate Computer Science (CS) degree [Joint Task Force on Computing Curricula and Society (2013)] and has clear practical applications, especially in software-related careers. Furthermore, algorithm skills often play a large part in job interviews for new graduates attempting to obtain a job after completing their computing degree [Behroozi et al. (2019)]. As such, it is important that students receive meaningful, high-quality instruction on the topic.

This work is also particularly necessary because theoretical computer science courses, like Algorithms, are likely a source of increased inequity. Students in CS already face higher levels of stereotype threat and impostor syndrome than their non-CS peers [Kumar (2012)], and these psychological threats are especially prevalent among women and students of color [Rosenstein et al. (2020)]. At the same time, some of the seminal studies about stereotype threat focus on its detrimental effect on women doing math [Spencer et al. (1999)]. As such, TCS courses, situated right at the intersection of CS and mathematics, are likely to place undue psychological pressure on students traditionally underrepresented in CS, resulting in more exacerbated achievement gaps. For these students especially, it is critical that effective pedagogical practices are developed with the goal of not only promoting success among students but also improving equity and access to the traditionally demographically homogeneous field.

Foundational work must be done in this area to guide future research. In this

work, we present a set of studies towards this goal. The primary contribution of this work, in Chapter 3, is a systematic mapping and review of CS Education literature related to algorithm design education, aiming to understand the current landscape of research and consolidate the published knowledge in the field. In Chapter 4, we discuss the implementation and progress of an ongoing study investigating students' metacognitive and help-seeking behaviors while solving Dynamic Programming problems.

CHAPTER 2

BACKGROUND

2.1 Systematic Literature Reviews

Systematic literature reviews (SLR) are an important part of pushing a research field forward. According to Cooper (1988), as a field expands, a synthesis of existing knowledge pertaining to the topic area becomes increasingly necessary to help researchers stay up to date with current developments, especially in topics related to one's primary specialty. Furthermore, according to Kitchenham and Charters (2007a), an SLR can help researchers know what is and is not known in the field as they plan for future studies, and can provide a framework through which any new discourse on the topic can contribute.

Cooper (1988) also provides a taxonomy of six major characteristics through which SLRs can be classified. We list the six characteristics below, along with the appropriate categories for this review, to help situate our contribution.

- **Focus:** Our review primarily concerns itself with *research methods* and *research outcomes*.
- **Goal:** Our review aims to *synthesize prior literature* and *identify issues central to the field*.
- **Perspective:** Our review takes the role of *neutral representation*, aiming to describe what is discussed by the literature without arguing for a specific point of view.

- **Coverage:** Our review's coverage of papers is *exhaustive with selective citation*. Though we believe our review includes most of the literature on the topic, some of the papers are not cited in this review. That said, a list of all papers reviewed is publicly available. The fully tagged corpus can be found at `https://doi.org/10.7910/DVN/KTR2ZH`.

- **Organization:** Our review organizes papers *conceptually*, both by topic studied and by results.

- **Audience:** Our review is intended for *specialized and general researchers*, as well as *practitioners*. CS Education researchers may use this review to understand the topic and situate new research, whereas practitioners who teach algorithms or related courses may use this review to discover interventions or insights for use in their own courses.

2.2 Literature Reviews in CS Education

SLR papers have appeared regularly in computing education venues in recent years, with reviews being conducted on topics like CS1 [Luxton-Reilly et al. (2018); Becker and Quille (2019)], Parsons problems [Ericson et al. (2022); Du et al. (2020)], undergraduate teaching assistants [Mirza et al. (2019)], meta-cognition in programming [Prather et al. (2020)], use of theory in computing education research [Malmi et al. (2019)], and demographic data reporting [Oleson et al. (2022)]. To our knowledge, there have been no literature reviews seeking to understand the state of algorithms education.

Methodologically, two reviews very similar to ours are by Sheard et al. Sheard et al. (2009), who use Simon's classification scheme to present a detailed quantitative landscape of programming education papers with some important outcomes listed, and by Švábenský et al. Švábenský et al. (2020), who quantitatively extract and present topics, methods, evaluation, impact, and contributors from cybersecurity education papers, analyzing trends of these characteristics without much focus on findings. These SLRs place a larger emphasis on the focus and approach of existing research in the area. We take a similar approach because the relative sparsity of work means that findings are difficult to synthesize but the methods and topics explored by these papers can still be used to guide future work.

Topic-wise, the closest systematic reviews are on algorithm visualizations and their effectiveness [Shaffer et al. (2007, 2010); Hundhausen et al. (2002)], one important subarea of algorithms instruction. While there is some overlap, these reviews cover a largely different set of literature than the literature reviewed in this paper. Both studies cover algorithm visualization of traditional algorithms (e.g. greedy programming). However, algorithm visualization reviews include topics that we do not include, such as using algorithm visualizations to learn $\mathcal{O}(n^2)$ sorting algorithms or basic data structure traversals, as they are typically taught prior to algorithms courses [Joint Task Force on Computing Curricula and Society (2013); Luu et al. (2023)]. In addition, our review covers non-visualization interventions and studies that seek to better understand student experiences in learning algorithms without providing an intervention. In fact, such non-visual areas make up a large proportion of the studies incorporated in this review.

CHAPTER 3

ALGORITHMS EDUCATION LITERATURE REVIEW

3.1 Introduction

Though the design of rudimentary algorithms has been extensively researched in an introductory programming context, most students will take a more in-depth algorithms-focused course later in their degree, as suggested by the ACM curricular guidelines [Joint Task Force on Computing Curricula and Society (2013)]. Research on algorithm design instruction, as taught in upper-level algorithms courses, is critically underrepresented in the literature. As such, many open questions remain about student experiences and difficulties in learning this material, as well as pedagogical practices for teaching algorithm design in an effective manner.

In this chapter, we present a systematic literature mapping and review regarding algorithms education, specifically as found in upper-level college courses, to understand with greater clarity both where knowledge exists and where open questions remain. Our work aims to answer the following research questions:

- **RQ1:** What algorithm topics are currently represented in the literature, and what kinds of studies are being conducted on these topics?
- **RQ2:** What kinds of interventions have been developed to teach algorithm design?
- **RQ3:** What knowledge currently exists in the literature about undergraduate algorithm design courses? What are the major takeaways?

The main contributions of this work are to:

- Characterize the current literature with respect to methods, topic, and authorship,
- Synthesize the existing knowledge about teaching algorithms, and
- Identify opportunities for future research.

3.2 Methods

To investigate our research questions, we conducted a systematic literature review, following guidelines from Kitchenham and Charters (2007b) and Randolph (2009).

3.2.1 Search Strategy

We conducted our search on the ACM Full-Text Collection in the ACM Digital Library (DL). To capture our research questions with an appropriate query, we first identified the set of topics that could be considered appropriate for an algorithms course. Within the "Algorithms and Complexity (AL)" knowledge area proposed by the most recent ACM Curricular Guidelines [Joint Task Force on Computing Curricula and Society (2013)], the sections *AL/Algorithmic Strategies* and *AL/Fundamental Data Structures and Algorithms* relate directly to our research questions. To distinguish between data structures topics and algorithms topics, we removed topics that most schools teach prior to their algorithms course as found by Luu et al. (2023) (e.g. Sequential and binary search algorithms).

For each topic remaining, we designed and validated a query to capture the corpus of literature on the topic. For some topics, this was straightforward: *Greedy algorithms* was captured by the query "greedy". For others, this required more care: for *Reduction: transform-and-conquer*, "reduction" was too common to be a reasonable query, whereas "transform-and-conquer" was too specific, but the query {"algorithm reduction" OR "reduction algorithm"} captures papers relevant to our interests. Included topics and their corresponding queries are shown in Table 3.1.

With this set of queries, we conducted three searches within the ACM Digital Library (DL). First, we searched for all *research articles* from venues sponsored by or in collaboration with SIGCSE (366 unique entries). Papers from SIGCSE TS and ITiCSE between 1970 and 2008 are classified as *articles* in the DL instead because they were published in the SIGCSE Bulletin, so we searched these as well (454 entries). Finally, we searched for research articles in TOCE (56 entries). From this corpus, we removed any retracted papers, ACM Bulletin Member Spotlights, or papers without PDFs in the DL.

We note that this literature search omits venues unaffiliated with the ACM DL. This decision was made in large part due to consistency in search mechanism and capacity constraints. We recognize that venues like the Taylor & Francis *Journal of Computer Science Education*, the Sage *Journal of Educational Computing Research*, and the CCSC regional conferences, among others, provide important contributions to the CS Education discourse as well, and encourage exploration of these venues for work related to our topic. That said, the set of venues included is consistent with other CS Education literature reviews [Heckman et al. (2021)].

3.2.2 Study Selection

We then applied three further exclusion criteria on the remaining 869 papers. First, we aimed to omit papers not relevant to our search, but variation in algorithms course content made this difficult. For example, only around 60% of institutions teach Depth-First Search/Breadth-First Search (DFS/BFS) in their algorithms courses [Luu et al. (2023)]. To ensure that no relevant papers were excluded, we kept any

Table 3.1: Topics and Queries Used

Topic	ACM Digital Library Query
Branch-and-bound	"branch and bound"
Brute-force algorithms	"brute force" AND algorithm
Depth- and breadth-first traversals	"breadth first search" OR "depth first search" OR "breadth first traversal" OR "depth first traversal"
Divide-and-conquer	"divide and conquer"
Dynamic programming	"dynamic programming"
Greedy algorithms	"greedy"
Heuristics	"heuristic algorithms"
Minimum spanning tree	"minimum spanning tree" OR "prim's algorithm" OR "kruskal's algorithm"
Pattern matching and string/text algorithms	"string algorithm" OR "text algorithm" OR "substring matching" OR "regular expression matching" OR "longest common subsequence"
Recursive backtracking	"recursive backtracking"
Reduction: transform-and-conquer	"algorithm reduction" OR "reduction algorithm"
Representations of graphs	"adjacency list" OR "adjacency matrix"
Shortest-path algorithms	"shortest path algorithm" OR "dijkstra's algorithm" OR "floyd's algorithm"
$\mathcal{O}(n \log n)$ Sorting algorithms	"quick sort" OR "quicksort" OR "heap sort" OR "heapsort" OR "merge sort" OR "mergesort"

paper that states involvement in an "algorithms course" as well as any paper explicitly teaching any of our topics (see Table 3.1). For example, a paper about a CS2 (Data Structures) course in general would be omitted, but a paper that describes teaching DFS/BFS in CS2 would be kept.

Papers were also only kept if they presented some form of data from college-level students. Either quantitative or qualitative data was accepted, but papers that describe an intervention or tool but do not evaluate it were omitted.

The exclusion criteria were applied by four of the listed authors, who independently filtered a subset of 20 papers, discussed to reach a consensus, and continued. After the authors filtered the second subset, Fleiss' Kappa statistic [Fleiss (1971)] was used to compute the inter-rater agreement for four raters, and the authors achieved Fleiss' Kappa $\kappa > 0.87$, so the rest of the papers were processed independently. In the end, 94 papers were identified as relevant to our literature review.

3.2.3 Paper Tagging

The authors then developed a set of tags for describing and classifying the relevant papers.

Codebook Development

A Content Analysis protocol [Drisko and Maschi (2016)] was used to develop a codebook with appropriate categories for our review. We began with a draft codebook with deductive codes and descriptions that all coders agreed upon. We then iteratively improved the codebook by repeating the following steps:

1. Each coder independently tags 10 papers according to the codebook.

2. Any disagreements in coding are resolved through discussion.

3. The codebook is adjusted to reflect the new shared understanding of the coders. This may involve adding or removing codes, or adjusting the descriptions.

At the end of the third cycle, no changes were made to the codebook. Fleiss' Kappa statistic was used to calculate reliability with a fixed number of raters on categorical items [Fleiss (1971)], and the authors achieved Fleiss' Kappa $\kappa > 0.80$ inter-rater agreement on this batch, so the remaining papers were tagged independently.

Final Codebook

The finalized codebook's codes were split into four major sections, described here. Within each section, the list of possible tags does not exhaustively consider every possibility for a paper but does capture every paper found in our review. All tags can be seen in Table 3.3.

Topic: Each paper was tagged by the topic or topics from the ACM curricular guidelines it focused on, and an extra *Not topic-specific* tag was added for papers that studied algorithms pedagogy as a whole without focusing on a specific topic. Our codebook specifies that topic tags are reserved for when the *Results* section of the paper explicitly describes the topic, so that whole-course interventions are tagged as *Not topic-specific* rather than with every individual topic taught in the course.

Evaluation: This set of tags aims to describe the data presented by the paper. These tags include *Quantitative Data* and *Qualitative Data* as broad categories, but

also data gathering methods (e.g. *Survey*, *Interview*) and data analysis methods (e.g. *Statistical Tests*, *Thematic Coding*). Note that papers could and often did contain multiple of these tags, and that some tags even imply others. For example, if a paper was tagged with *Thematic Coding*, then it was also tagged with *Qualitative Data*.

Metric: This set of tags aims to determine the student metric measured by the paper. We found all papers presented data on at least one of *Performance*, *Affect*, or *Persistence* (whether students remain enrolled in the course).

Intervention: This set of tags aims to capture information about the intervention developed in the paper, if applicable. We categorized interventions based on the aspect of the course targeted, and introduced a *Tools* tag because some interventions developed a tool without directly affecting the course. These were not mutually exclusive — for example, a tool may be developed to help students during discussion. A full list and description of intervention tags can be found in Table 3.2.

3.2.4 Results Consolidation

To identify major takeaways from the corpus, the papers were split based on whether they presented an intervention. For papers containing interventions, the first and third authors read each paper, recording the results of the paper as well as any features of the intervention explicitly described in the paper as beneficial. These results and features were consolidated and inductively coded by the authors. Papers not containing an intervention were instead inductively coded based on focus. The final codes can be found as the headers in Section 3.4.

Table 3.2: Tagging Scheme to Characterize Interventions

Tag	Description
None	A study that does not change the way students learn the material, and instead investigates the status quo, e.g. misconceptions or student behavior.
Tools	A study that develops and/or evaluates a tool that a student can interact and interface with, especially a software tool, that aids with student learning.
Course Policy	A study that changes course logistics unrelated to content (e.g. late policy or collaboration policy).
Content Presentation	A study that changes how the students are presented non-interactive course material (e.g. lecture).
Discussion/Lab	A study that changes staff-facilitated problem-solving sessions (e.g. labs and group work).
Student Work	A study that changes non-staff-facilitated work done by students (e.g. homework or projects).
Exams	A study that changes how students are evaluated in test settings.

3.3 Corpus Survey

The results of the tagging can be found in Table 3.3. Note that a single paper can be tagged in multiple categories and topics, so it can be represented in multiple rows and columns. The last row contains a summary of the results. The fully tagged corpus can be found at `https://doi.org/10.7910/DVN/KTR2ZH`.

We begin by presenting a broad survey of the methodology used in the corpus. We then drill down to answer more specific questions about topic coverage, changes over time, "rigor" (which we define later), and the venues and authors that publish the most. Findings are accompanied by a discussion about implications and future steps.

Table 3.3: Tags by Topic. Note that some papers cover multiple topics, and contribute to each corresponding row in the table.

		Evaluation								Metric			Intervention						
Topic	Total	Quantitative Data	Survey	Scores/Grades	Statistical Tests	Controlled Trial	Qualitative Data	Interview	Thematic Coding	Performance	Affect	Persistence	None	Tools	Course Policy	Content Presentation	Discussion/Lab	Student Work	Exams
Branch-and-bound	1	1	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0
Brute-force Algorithms	2	2	1	2	2	2	0	0	0	2	1	1	0	2	0	0	1	1	0
Depth- and Breadth-First Traversals	7	6	5	4	3	3	3	1	1	4	5	0	0	3	0	2	1	3	0
Divide-and-Conquer	3	3	1	3	2	1	2	1	0	3	0	0	2	1	0	0	1	0	0
Dynamic Programming	10	10	6	8	5	3	5	2	1	8	5	1	3	4	0	1	1	3	0
Greedy Algorithms	8	8	4	6	0	0	6	2	5	6	4	0	2	3	0	3	2	2	0
Heuristics	3	3	0	3	0	0	2	0	2	3	0	0	1	1	0	0	0	2	0
Minimum Spanning Trees	5	5	4	3	1	1	2	0	0	3	3	1	0	3	0	0	1	4	1
Pattern Matching and String/Text Algorithms	1	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0
Recursive Backtracking	3	3	2	3	1	1	2	0	1	3	2	0	1	2	0	1	1	0	0
Reductions	6	5	3	6	1	0	4	1	2	6	2	0	3	0	1	2	0	2	0
Representations of Graphs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Shortest-path Algorithms	6	6	4	5	2	2	1	0	0	5	4	1	0	6	0	1	1	1	0
$\mathcal{O}(n \log n)$ Sorting Algorithms	10	9	7	6	5	2	5	1	0	6	9	1	1	4	0	2	2	4	0
Total (any topic)	46	43	28	35	15	10	23	6	8	35	27	2	9	20	1	10	9	15	1
Not topic-specific	48	45	27	34	23	7	22	6	7	34	30	2	11	16	11	9	7	16	2
Total	94	88	54	69	38	17	45	12	15	69	57	4	20	36	12	19	16	31	3

3.3.1 Corpus Methodology

Nearly every paper (88/94) meeting our selection criteria presents quantitative data, and 73% (69/94) of papers use student scores and grades. Notably, around 40% (38/94) of the papers run statistical tests, and only around 18% (17/94) run controlled trials. On the other hand, only half of the papers use qualitative data. Thematic coding, found in only 16% (15/94) of papers, was done for interview responses (e.g. Toma and Vahrenhold (2018)), student work (e.g. Velázquez-Iturbide (2012)), and survey responses (e.g. Weber (2023)). Both quantitative and qualitative data are presented in 41% (39/94) of the papers.

In terms of evaluation metrics, 73% (69/94) of papers use student performance, 61% (57/94) use affective measures, and 34% (32/94) of papers use both. Of the four papers measuring persistence [Gárcia-Mateos and Fernández-Alemán (2009); Deb et al. (2017); Lin et al. (2021); Coffey (2013)], all four measure performance as well, and three measure affect, so it may be worthwhile to conduct studies more centered around persistence in the algorithms class.

Most papers (79%, 74/94) present interventions, and around 40% (36/94) center around a tool. For example, five papers present a strategy for implementing active learning in the classroom via interactive question-and-answer tools [Pargas (2006); Anderson et al. (2007); Kurtz et al. (2014); Phan and Hicks (2018); Deb et al. (2018)]. On the other hand, around a fifth (20/94) of the papers surveyed do not develop an intervention. Many of these papers analyze student problem-solving strategies when presented with algorithm design problems, and a number of others study topic-specific misconceptions.

We only find four interventions related to student examinations. Given the broad applicability of algorithms to test questions and the effect that question design and test logistics can have on exam validity, we expect research on this subject to be potentially useful.

3.3.2 Representation of Topics

Of the 94 papers included in our literature review, nearly half focus on a specific topic or set of topics, whereas the other half present results that do not pertain to any particular topic, such as studies about broad problem-solving skills or policy-based interventions.

Most topic-specific papers only focus on one topic, though ten papers focus on two topics, three were tagged with three topics each [Deb et al. (2017); Taylor et al. (2009); Brown et al. (2022)], and one paper was tagged with four (*Branch-and-Bound*, *Greedy Algorithms*, *Heuristics*, *Recursive Backtracking*) [Velázquez-Iturbide (2019)]. However, many areas are still understudied. In this section, we begin by discussing trends in individual topics, and then compare topic-specific papers with those that are not topic-specific.

Topics with More Coverage

At 10 papers, $\mathcal{O}(\mathbf{n \log n})$ **Sorting Algorithms** are the topic most covered in the literature, though we expect that $\mathcal{O}(n^2)$ algorithms are studied far more thoroughly. Most of these papers leverage sorting as a familiar task to develop engaging active-learning interventions for introducing faster sorting algorithms, including through

games [Hakulinen (2011); Hosseini et al. (2019)] and exploratory activities [Rasala et al. (1994); Rebelsky (2005); McCann (2004)], though a couple instead use assessment results to provide insight into common misconceptions [Winters and Payne (2006); Taherkhani et al. (2012)].

Research about the topic of **Dynamic Programming** has taken a relatively wide variety of approaches. Two papers contribute interesting course projects that motivate the use of Dynamic Programming through various problems, one from a centuries-old mathematical text [Pengelley et al. (2006)] and the other from the modern world [Bird and Curran (2006)]. Tools have also been developed to help scaffold different steps of solving DP problems, specifically interpreting the problem constraints [Lin et al. (2021)], visualizing subproblem recurrences [Velázquez-Iturbide et al. (2016); Velázquez-Iturbide and Pérez-Carrasco (2016)], and writing a proof [Xia and Zilles (2023)]. Research has been conducted [Danielsiek et al. (2012); Paul and Vahrenhold (2013)] and reproduced [Zehra et al. (2018)] investigating common roadblocks in solving DP problems as well, finding that students either don't recognize that DP is the correct approach, fail to split the problem into appropriate subproblems, or have trouble identifying the correct recursion of subproblems. Finally, Enström and Kann (2017) find that teaching the DP process as separate steps can increase student self-efficacy .

On the other hand, **Greedy Algorithms** has seen 8 papers but markedly less breadth, with one paper about misconceptions [Velázquez-Iturbide (2019)], one about collective argumentation [Kallia et al. (2022)], and the other six about interactive learning interventions. Though five provide thematic coding of student responses,

none provide statistical tests in their evaluations, limiting our understanding of intervention effectiveness.

Topics with Less Coverage

We found no papers related to **Graph Representations** and one related to **Pattern-Matching and String/Text Algorithms** [Bird and Curran (2006)], though the latter may be partially captured in research about Dynamic Programming and Greedy Algorithms. Despite **Divide-and-Conquer**, **Minimum Spanning Trees (MST)**, and **Shortest-path Algorithms** being taught in more than 80% of algorithms courses [Luu et al. (2023)], the topics are only studied by 3, 5, and 6 papers, respectively. Of the three Divide-and-Conquer papers, two focus on misconception detection in a wide set of topics that includes Divide-and-Conquer [Danielsiek et al. (2012); Paul and Vahrenhold (2013)], while the third involves a tool for learning in the classroom Phan and Hicks (2018), so questions remain about presenting, testing, and assigning work on the topic. Four of the five MST papers relate to the development of student assignments [Deb et al. (2017); Taylor et al. (2009); Erkan (2018); Stasko (1997)], and the other paper studies exam questions [Gaber et al. (2023)], leaving the presentation of the material or common student misconceptions unexplored. Finally, all six Shortest-path papers are tool-based interventions, leaving plenty of room for exploration of misconceptions or pedagogical strategies.

Papers Without Specific Topics

Within non-topic-specific papers (half of our set), there is considerable variation in the extent to which the knowledge extracted applies specifically to algorithms courses. For example, some papers test interventions in an algorithms course but could have reasonably tested them in any CS course (e.g. [Belleville et al. (2020); Pargas (2006)]), some studies include algorithms courses alongside other CS courses in their data set (e.g. [Deb et al. (2018); Bennedssen and Caspersen (2008)]), and some studies are topical to algorithms in particular (e.g. [Danielsiek et al. (2017); Collberg et al. (2004)]).

For the most part, these non-specific studies mirror the topic-specific papers in terms of evaluation and intervention methods used. The most notable difference is in *Course Policy*, where all but one paper focuses on the course as a whole rather than individual topics. This is unsurprising, as we expect course policy changes to affect all topics in a course equally, though Crescenzi et al. (2013) study a sizeable modification to course structure targeted at the course's NP-completeness unit.

That said, the studies are still notably different in their aims. For example, while most topic-specific papers without interventions focus on identifying misconceptions for the topic, papers without specific topics instead focus on broader measurements like skill development [Ginat (2001)], learning outcomes [Bennedssen and Caspersen (2008)], and even student affect [Danielsiek et al. (2017)]. Likewise, while most content-specific tools presented are related to algorithm visualization, papers without specific topics feature not only visualization tools (e.g. Naps et al. (2000); Lee and Rößling (2011)) but also tools for facilitating improvements in grading [Gárcia-

Mateos and Fernández-Alemán (2009); Coore and Fokum (2019)], lectures [Pargas (2006); Kurtz et al. (2014)], and assignments [Zeller (2000); Marshall et al. (2006)].

3.3.3 Topics over Time

In Figure 3.1, we display the distribution of topics covered by year. Sorting Algorithms dominate the topic-specific literature early on, representing the only topic in 1994-1998 and 5 of 12 between 1994-2008. However, we also see it become more infrequent in later years despite the general steadiness of papers. Even so, in every period, at least two-thirds of the papers within our study relate to either Sorting, DFS/BFS, Greedy Algorithms, or Dynamic Programming.

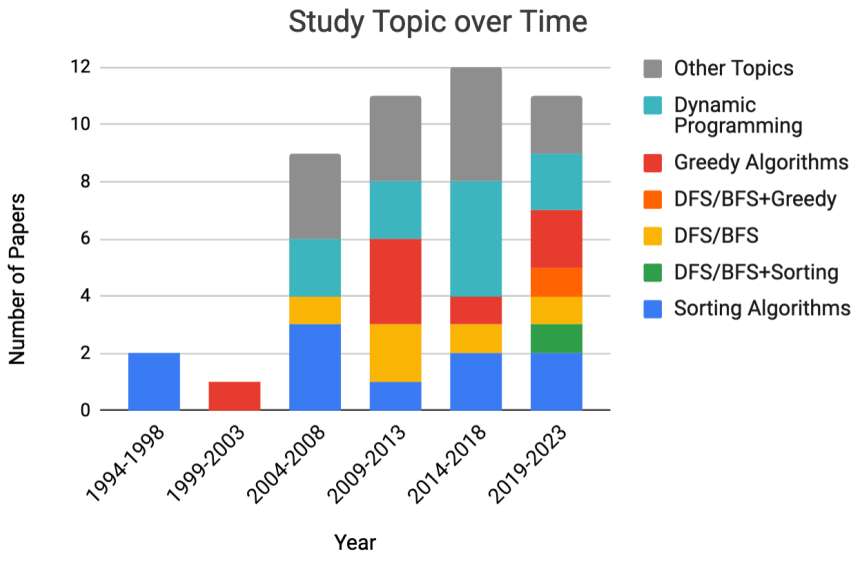

Figure 3.1: Topic of topic-specific papers, by year

3.3.4 Rigor over Time

Scientific rigor is by nature tough to quantify, as the best practices for any paper depend on the specific research questions. For papers with quantitative data, the use of *statistical tests* or *controlled trials* serves as our proxy for a metric of rigor [Sanders et al. (2019)]. Similarly, for papers presenting qualitative data, we use *thematic coding*. Note that the papers with no evaluation or student data were already filtered out of our review. Figure 3.2 shows the percentage of each type of study, presented in 5-year bands.

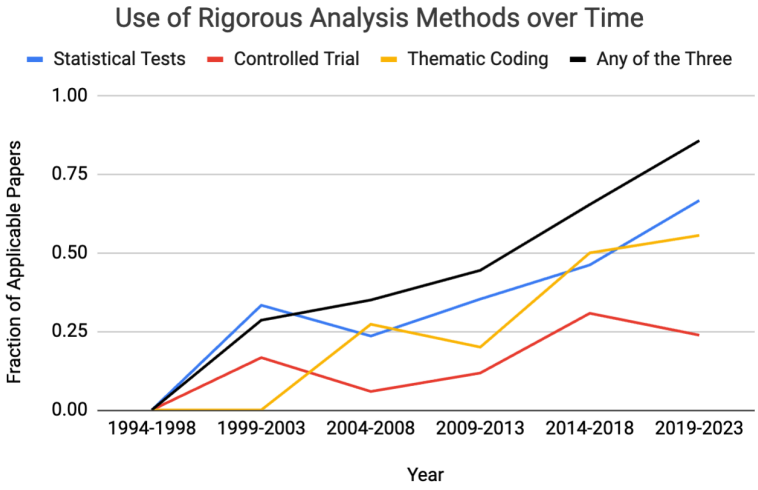

Figure 3.2: Percentage of papers with statistical tests (quantitative), controlled trials (quantitative), thematic coding (qualitative), and at least one of the above

We see no "rigorous" studies initially, but there is an encouraging upward trend in the usage of rigorous procedures in algorithms papers as the field matures. In the most recent period, 2019-2023, 56% of qualitative papers use thematic coding, and two thirds of quantitative papers use either statistical tests or controlled tests.

3.3.5 Where and Whom are Studies Coming From?

Our search consisted of all publications sponsored by or in collaboration with SIGCSE, as well as TOCE. The distributions of papers per conference, split by year, can be found in Figure 3.3.

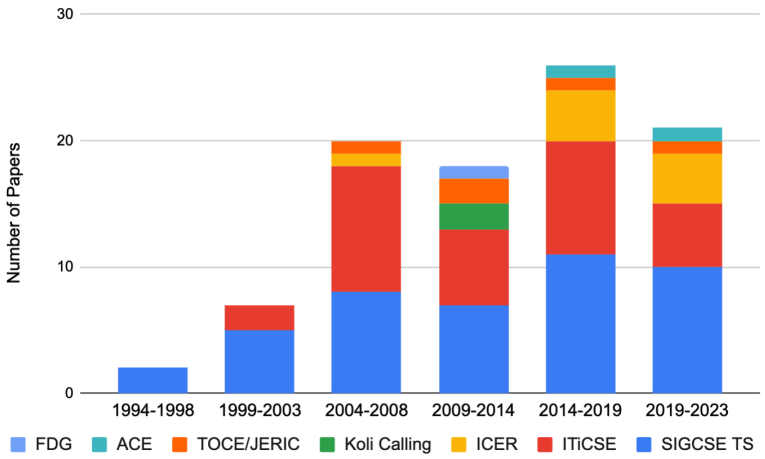

Figure 3.3: Papers per venue, by year

As expected, SIGCSE's largest conferences, the Technical Symposium and ITiCSE, account for a vast majority of the papers (44 and 33, respectively) in our literature review, with ICER contributing more recently. We found five relevant papers from TOCE (and its previous moniker JERIC) as well.

We also seek to understand whether much of the work is accomplished by just a few groups, or whether it is widespread in the community. The tally of papers per author can be found in Table 3.4.

We can see that the most active groups are two pairs of collaborators. Velázquez-Iturbide / Pérez-Carrasco have studied optimization algorithms and developed vi-

Table 3.4: Number of Papers per Author. Asterisks indicate frequent collaborators.

Author	Number of Publications
J. Ángel Velázquez-Iturbide*	7
Jan Vahrenhold**	5
David Ginat	4
Antonio Pérez-Carrasco*	4
Michal Armoni	3
Holger Danielsiek**	3
[12 authors]	2
[183 authors]	1

sualization tools to assist with a variety of algorithm paradigms (e.g. Velázquez-Iturbide (2012); Velázquez-Iturbide et al. (2016); Velázquez-Iturbide (2019); Velázquez-Iturbide and Pérez-Carrasco (2016)) and Vahrenhold / Danielsiek have explored instruments for measuring student affect and success in algorithms courses (e.g. Danielsiek et al. (2012, 2017); Paul and Vahrenhold (2013); Toma and Vahrenhold (2018)). In addition, David Ginat has written four papers discussing metacognitive strategies for algorithmic problem-solving (e.g. Ginat (2001); Ginat and Blau (2017)), and Michal Armoni has conducted a series of studies on reductive thinking [Armoni (2009)]. The vast majority of authors publish only one or two papers (195 authors), limiting the depth of contributions that might be made.

3.4 Corpus Takeaways

In this section, we go beyond the numbers to provide insights into the major takeaways from the corpus as a whole. Subsections are broadly sorted in order of decreasing specificity to algorithms courses.

3.4.1 Student Approaches to Algorithm Design Problems

Though the skill of algorithm design is somewhat nebulous, there is general consensus that it is closely tied to abstraction ability. Bennedssen and Caspersen (2008) examine 10 courses mandatory for the CS degree and find that abstraction ability correlates with performance in only the algorithms course. Think-alouds have been run to identify varied extents to which students utilize abstraction to solve problems [Ginat and Blau (2017); Izu et al. (2017)].

More specific algorithm design skills have been studied as well, with the general finding that students under-utilize them even after having taken an algorithms course. In problems where recursion [Ginat (2004)] and reduction [Armoni (2009)] would have been helpful, students understand the concepts but underestimate their utility in general problem-solving contexts. On the other hand, students demonstrate an overreliance on intuitive greedy approaches, even in the face of counterexamples [Ginat (2003)].

At the topic level, a limited number of papers explore common misconceptions in algorithms topics. Danielsiek et al. (2012) use both student work and thinkalouds to cover a wide variety of topics, but other studies have focused on more specific topics like dynamic programming [Zehra et al. (2018)] and optimization algorithms [Velázquez-Iturbide (2019)].

3.4.2 Problem Selection

We find a recurring theme that intentional problem selection can be leveraged to motivate students. The broad applicability of algorithms lends itself nicely to problems

tied to the real world, which can give students both familiarity with the problem setting and motivation to solve it. For example, linking shortest path algorithms to GPS navigation is a setting in which multiple papers demonstrate that students enjoy and engage more with the real-world map [Holdsworth and Lui (2009); Teresco et al. (2018)]. Mehta et al. (2012) presented bipartite matching in the setting of forming balanced class project groups, resulting in both more successful groupings and an increased understanding of bipartite matching. Problems can also tie into the real world through the researcher: Pengelley et al. (2006) presented a Dynamic Programming problem posed by a mathematician's letter from 1838, and Wirth and Bertolacci (2006) presented a problem from the instructor's research, both finding that the humanized context motivated students.

Problems with a variety of solutions can also be particularly helpful. In multiple studies, students indicate that writing both efficient and inefficient solutions to problems and comparing their runtimes on real inputs provided more motivation for designing efficient algorithms than theoretical runtime analyses [McCann (2004); Coffey (2013)]. Similarly, studies have designed projects asking students to compare BFS against DFS [Erkan and Gochev (2008)], Prim's algorithm against Kruskal's algorithm [Erkan (2018)], and several different sorting [Rasala et al. (1994)] and matching [Lucas (2015)] algorithms, finding in all cases that students describe the comparisons as valuable and insightful. In each of these projects, the comparison is facilitated in part by a visualization of either the evaluation or output of the algorithms.

3.4.3 Algorithm Visualization

In our data set, we identified 15 studies about algorithms visualizations. Broadly, students state that they appreciate visualizations, though few studies experimentally measure learning outcomes. Of these papers, three found that introducing visualizations improves student learning [Velázquez-Iturbide (2013); Deb et al. (2017); Farghally et al. (2017)], though only one uses statistical tests. On the other hand, Jarc et al. (2000) find that introducing visualizations provides no statistically significant learning outcomes, and Velázquez-Iturbide and Pérez-Carrasco (2016) find that students are more efficient but do not perform better. Taylor et al. (2009) find that student learning significantly improves when the visualization forces students to take an active, predictive role. For a more thorough review on algorithms visualizations that is not specific to algorithms courses, we recommend the work by Hundhausen et al. (2002).

3.4.4 Coding Practice

Though around a quarter of algorithms courses have no programming component [Luu et al. (2023)], research has found that algorithmic coding assignments increase student engagement [Gárcia-Mateos and Fernández-Alemán (2009)] and helps students develop their algorithmic design [Izu and Alexander (2018)] and runtime analysis [Coore and Fokum (2019)] skills. Online coding platforms allow for a public leaderboard of coding problem scores, which some studies have used in an attempt to motivate students [Coore and Fokum (2019); Färnqvist and Heintz (2016)] with mixed results: Coore and Fokum (2019) observed that the strongest students worked

hard to maintain their ranking, but other students were demotivated when the top scorers had yet to solve a problem.

3.4.5 Assessment Policy

The use of algorithmic coding problems permits automated assessment, which students find fair and constructive [Färnqvist and Heintz (2016); Weber (2023)], though additional human interpretation of automated feedback may be helpful [Leite and Blanco (2020)]. Weber (2023) finds promise in standards-based grading for algorithms courses, along with allowing students to re-submit work as they continue to master course concepts. While students did continue to improve on past assignments throughout the course, some took advantage of the chance to procrastinate work until the end of the term.

Two papers investigate the utility of two-stage quizzes (consisting of an individual and a group phase), finding that team quizzes increase students' perceived and measured learning gains [Belleville et al. (2020); Ham and Myers (2021)]. Gaber et al. (2023) make the case for repeating previously-seen problems on exams, showing that these questions are still considerably challenging for students yet perceived as fair.

3.4.6 Active Learning

Research on algorithms-based active learning exercises in lecture corroborates findings in other settings that students enjoy exploring a problem space with their peers [Belleville et al. (2020); VanDeGrift (2017); Toma and Vahrenhold (2018)], feel an increased sense of engagement in class [Anderson et al. (2007); Gehringer

and Miller (2009)], and report learning both course material and collaboration skills [VanDeGrift (2017)]. Additionally, incorporating in-class interaction into historically lecture-based courses led to higher instructor satisfaction in some cases [Hosseini and Perweiler (2019); Bouchez-Tichadou (2018)].

Most studies incorporate active learning by having students work through problems in class, whether exploring a problem instance [Anderson et al. (2007); Kurtz et al. (2014)], coding an algorithmic solution [Phan and Hicks (2018)], or performing algorithm analysis [Pargas (2006); Deibel (2005)]. Some of these activities were equipped with auto-grading as well, which Deb et al. (2018) find improved student retention of course material. To dissuade a "guess-and-check" approach while using an auto-grader, Kurtz et al. (2014) capped grades based on the number of submissions.

Game-based exercises have also shown promising results, with research finding that these activities can drive participation from students who are typically silent [Hosseini and Perweiler (2019); Hosseini et al. (2019)]. Some of the algorithms-based games developed include a sorting algorithm design contest [Hosseini et al. (2019)], a matching game in which students choose algorithms for given criteria, and a draw-and-guess game in which students illustrate course content [Hakulinen (2011)].

However, despite the success of interactive in-class activities, facilitating productive group work may require further attention: in a study on communication behaviors during a collaborative algorithmic problem solving session, Kallia et al. (2022) observed that first-year undergraduates struggled to build off of each others' ideas while MSc students were overly dismissive of others' ideas. Two papers find success using a variety of intentional group-formation strategies to encourage produc-

tive collaboration, making groups based on learning style, prior student knowledge, or mixed instructor/student input [Deibel (2005); Mehta et al. (2012)].

3.4.7 Psychological Factors

Little is known about psychological factors in algorithms courses, but a series of papers by Danielsiek, Toma, and Vahrenhold develop and validate an algorithms-specific instrument to measure self-efficacy [Danielsiek et al. (2017)] and use it along with other instruments to measure the cognitive load and perceived benefit of different collaborative lab assignments [Toma and Vahrenhold (2018)]. Their work finds that collaborative labs are perceived as effective and inclusive, and demonstrate the potential for using measurements of various emotional responses to guide intervention design.

3.5 Limitations and Future Work

Though we designed our search query to capture papers related to algorithm design, some papers may have been missed. First, if a paper does not use our search terms, our query may not capture it. Furthermore, our search was limited to venues sponsored by or in collaboration with SIGCSE in the ACM DL, so research published outside of these venues was not included (e.g. the Taylor & Francis journal *Computer Science Education*), though we expect to have captured most of the work in the area. (See Section 3.2.1.)

The variability in course offerings across institutions also may have resulted in some omitted papers. Topics that we consider to be relevant to algorithm design are

sometimes taught in prerequisite courses like CS2 [Luu et al. (2023)], but we may not be able to tell based on the paper. As such, knowledge about teaching these topics in other courses is missed if the paper does not explicitly mention our topic of interest.

Furthermore, while in this review we were able to cover all topics from the ACM Curricular Guidelines sections *AL/Algorithmic Strategies* and *AL/Fundamental Data Structures and Algorithms* which are usually covered in an algorithms course, we were not able to cover topics listed in the related sections *AL/Basic Automata Computability and Complexity* about computing theory and *AL/Basic Analysis* about algorithmic analysis. These topics are also required learning for CS majors according to the ACM Curricular Guidelines, though they are less frequently covered in algorithms courses [Luu et al. (2023)]. Future reviews could use these as topics of focus.

As with any qualitative coding endeavor, we also note that misinterpretations of papers in the review are possible. However, we hope that the content analysis procedure, with many rounds of codebook refinement and discussions about potentially different interpretations of the codebook, mitigated the likelihood of these outcomes.

CHAPTER 4

DYNAMIC PROGRAMMING THINK-ALOUD INTERVIEWS

4.1 Introduction

Dynamic Programming (DP) is a quintessential algorithms topic, covered in more than 80% of college upper-level Algorithms courses [Luu et al. (2023)]. As discussed in the prior chapter, it has seen slightly more research focus than other algorithms topics, at least in part due to its general perception as one of the most conceptually challenging topics in the course [Enström and Kann (2017)]. A summary of the variety of work done on this topic can be found in Section 3.3.2.

As described, little work has been done investigating metacognitive interventions for teaching DP. Prior work by Danielsiek et al. (2012), Paul and Vahrenhold (2013), and Zehra et al. (2018) explore common roadblocks experienced by students independently solving DP problems, and Enström and Kann (2017) describe teaching the last step of DP first to boost student self-efficacy with the topic. In this chapter, we present an ongoing project that extends their work by conducting an investigation examining student metacognitive approaches to DP. We conduct think-aloud interviews where students receive on-demand semi-structured guidance as they work through the problems. This guidance is focused on metacognitive approaches to solving the problems, designed to answer the following research questions:

- **RQ1:** What common incomplete understandings or misconceptions lead to students being unable to solve Dynamic Programming problems?

- **RQ2:** What metacognitive approaches do students develop while learning to solve Dynamic Programming problems?
- **RQ3:** What obstacles prompt students to request further guidance while solving Dynamic Programming problems?
- **RQ4:** To what extent does structured metacognitive guidance benefit students solving Dynamic Programming problems?

This work is still ongoing: interviews have been conducted and fully transcribed, and data analysis is in its very early stages. As such, in this chapter we describe the study's motivation and design, as well as the planned methodology for data analysis, but no results will be presented.

4.2 Background

4.2.1 Metacognition

Metacognition, somewhat colloquially described as "thinking about thinking" [Miller et al. (1970)], refers to a student's self-regulatory knowledge and mechanisms related to their own learning, though the scope of this and its overlap with *self-regulation* can vary [Dinsmore et al. (2008)]. In our context, we operationalize the concept to refer to a student's ability to evaluate the effectiveness of their own learning/problem-solving strategies and reuse effective ones. The concept is well-established even within the field of CS education, demonstrated in part by the review conducted by Prather et al. (2020) about metacognition in programming education. Metacognition has

been studied to a limited extent within algorithms as well - see Section 3.4.1.

4.2.2 Dynamic Programming Problem-Solving Steps

One popular method for developing a student's metacognitive problem-solving skills is in-process scaffolding, in which students are explicitly provided with a series of problem-solving steps to help them self-orient and self-reflect as they think through solving a problem (e.g., Prather et al. (2019); Loksa et al. (2016)). Loksa et al. (2016) presents the following six steps for solving open-ended programming problems:

1. Reinterpret the problem prompt,
2. Search for analogous problems,
3. Search for solutions,
4. Evaluate a potential solution,
5. Implement a solution,
6. Evaluate the implemented solution.

Though these steps do not translate directly to algorithm design, the popular algorithms textbook by Cormen et al. (2022) provides a set of steps for tackling Dynamic Programming problems:

1. Characterize the structure of an optimal solution,
2. Recursively define the value of an optimal solution,

3. Compute the value of an optimal solution in a bottom-up fashion.

These steps are validated to an extent by the work done by Danielsiek et al. (2012) and Zehra et al. (2018), who identify these steps as common places where students falter in solving DP problems. Within our study, we take inspiration from both of these lists in both orienting and guiding students as they solve DP problems.

4.3 Methods

4.3.1 Study Population and Recruitment

We conducted interviews in Winter 2024 after receiving IRB approval. Participants were recruited from our institution's Algorithms course, through announcements made by the course's instructors on behalf of the researchers. In these announcements, students were informed of an opportunity to participate in a study where they would receive both practice and guidance about solving Dynamic Programming problems. Students were also informed that participation was entirely voluntary and that instructors would not be informed of student participation.

Interviews were conducted in two rounds: one for the two weeks between the release and due date of the DP homework assignment, and the other for the two school days prior to the course final. 20 interviews were conducted in the first round, and 11 interviews were conducted in the second round. Participants were allowed to participate in both rounds if desired, and 7 of the second-round interviewees were also interviewed in the first round. We especially wanted to collect data from returning participants in order to measure how student approaches develop as they

gain experience with the material.

4.3.2 Interview Protocol

Interviews were conducted with a think-aloud model, where participants were presented with a series of prepared Dynamic Programming problems and asked to narrate their thought process to the best of their ability as they worked through the problems. Participants were informed that the interviewer could provide guidance at any time, but only upon request. This delineation was made both to allow the participant to work as independently as possible and to understand what types of help are requested by the participant. Each interview lasted approximately one hour.

Two forms of data were collected. First, a transcript was created from a voice recording of the interview. Second, participants were provided with scratch paper for the participant to work on, which was scanned at the end of the interview.

Dynamic Programming problems were selected to ensure that a wide variety of solution types were covered. Specifically, we selected both 1D and 2D problems, varied the data type both being processed and being stored in the DP array, and picked a problem with a non-standard subproblem and extraction method. Students solved between 1-3 problems in each session, and problems were presented to students in increasing difficulty. Students who were being interviewed for a second time received problems from a new set of similar difficulty. The first problem, Coin Row, is the same as the one used in prior DP misconceptions research Danielsiek et al. (2012); Zehra et al. (2018), though the others differ. The list of problems can be found in the appendix, Section A.1.

During the last few minutes of each interview, participants were also asked a short series of questions about their experience solving algorithms problems. These questions were also changed for second-time interview participants. The following questions were asked for a participant's first interview:

- How did it feel to work through those questions?
- Is there anything you'd like us to know about the process of solving algorithms that you use that may not have been represented while you were solving the problems?
- What do you think has had the largest effect on the way you approach these problems? (if asked for examples, a class, a professor, a textbook, a video.)
- Please describe your experience with Algorithms outside of UChicago. Do you work on problems related to Algorithms for interview practice, research, or any other extracurricular activities?

For a returning participant, we instead asked the following questions:

- How did it feel to work through those questions, compared to last time?
- What has guided your learning process for Dynamic Programming over the last few weeks? What has been helpful? Unhelpful?
- How has your comfort level with Dynamic Programming changed over the last few weeks?

4.3.3 Provided Guidance

The guidance provided by interviewers was semi-structured to focus on metacognitive guidance if possible. When students requested help, interviewers asked follow-up questions if necessary to differentiate between four levels of help, inspired by a framework presented by Franklin et al. (2013):

1. **Validation:** Student needs reassurance that they are on the right track.
2. **What:** Student needs a keyword/pointer about the next step.
3. **How:** Student still needs some guidance even if given a keyword.
4. **Reteach:** Student needs entire lesson to be retaught.

The guidance provided by the interviewer was determined by the level indicated. If the request was on level 1, they were simply provided reassurance. If the request was on level 2, the participant was presented with a list of four problem-solving steps, essentially a combination of the steps provided by Loksa et al. (2016) and Cormen et al. (2022) (see Section 4.2.2):

- Reinterpret the problem prompt,
- Pick a subproblem,
- Write a recurrence relation,
- Compute the value of an optimal solution in a bottom-up fashion.

Participants were then simply asked to identify which step they were on. For level 3 requests, participants were provided help both with the steps above and through the lens of a different popular metacognitive strategy: creating a worked example. In particular, the participant was instructed to create a small worked example, and were asked a set of questions that helped them use the worked example to complete the step they were on. Finally, for level 4 requests, the interviewer also provided more problem-specific and participant-specific guidance as necessary.

4.3.4 Planned Analysis

Two researchers will independently code all participant interview transcriptions and scanned student work. The researchers will analyze the interviews, coding for the following:

- Common incomplete understandings of Dynamic Programming,
- Types of help requested,
- Student utilization of problem-solving strategies, and
- Common sources of information relied on by students outside of the interview.

Coding will be done inductively. The researchers will meet both during and after the coding process to discuss any emergent themes or findings.

CHAPTER 5

FUTURE WORK

As discussed, the sparsity of research on algorithms education suggests many areas for future work. Section 3.3.2 discusses potential areas of exploration at a more fine-grained level.

At a broader level, our review finds that research in the area relies heavily on quantitative data, especially measuring student scores or grades, but relatively few studies use statistical tests and even fewer run controlled trials. Though we acknowledge the difficulty of running controlled trials in an educational setting, more rigorous approaches to this data would be valuable to better understand the effectiveness of various interventions.

At the same time, this reliance on quantitative data is accompanied by a dearth of qualitative data, especially about psychological factors like sense of belonging or self-efficacy. Rigorous explorations of student experiences in these classes would be incredibly useful for not only determining the extent to which the mathematical content of algorithms courses affects psychological factors but also orienting future research around mitigating impostor syndrome, stereotype threat, or other discovered barriers.

CHAPTER 6

AUTHOR'S CONTRIBUTION

The literature review presented in this Master's paper is work that was performed by a collaborative, multi-institutional team, on which I took a leadership role. I led weekly meetings for the project, providing insight into the direction and scope of the literature review. I developed the initial version of the codebook and updated it in accordance with the team's updated decisions and understandings. I tagged more than half of the papers in both the study selection (869 papers) and paper tagging (94 papers) phases of the review. Data compilation, analysis, and visualization were all done by me as well. During our survey of paper results, I read every paper in the corpus, and worked with one collaborator to group the results by major themes. Sections 3.3.4, 3.3.5, and 3.3.6 were written by this collaborator.

The think-aloud interviews are also work being performed with one collaborator.

CHAPTER 7

CONCLUSION

In this thesis, we present a pair of studies aimed at increasing our understanding of teaching upper-level undergraduate algorithms. In an effort to synthesize research related to algorithm design education, we categorized all papers in the ACM Digital Library containing search terms for algorithm design topics covered in algorithms courses, guided by the ACM Curricular Guidelines and surveys of algorithms course instructors [Joint Task Force on Computing Curricula and Society (2013); Luu et al. (2023)]. We found that algorithms research is a growing field, with more rigorous methods being applied over time. Studies have generally agreed that active learning interventions and interesting problem-selection can contribute to both learning and motivation, corroborating research in the broader CS Education space. However, even among the most studied areas in algorithm design, there are still large gaps in the literature. For some topics, there is adequate literature on student misconceptions, but concrete instructional interventions are lacking. In others, researchers have created interventions, but have not yet rigorously tested their efficacy using controlled trials. Exams and Psychological Factors stand out as important areas with very little research at all. The second study we present is ongoing work aimed at resolving one of those gaps - there is adequate literature about where students tend to struggle in the process of solving Dynamic Programming problems, but no studies have developed metacognitive instructional interventions for guiding them through it. We hope this review can similarly inform other research directions and help guide the future of algorithms education.

REFERENCES

Richard Anderson, Ruth Anderson, K. M. Davis, Natalie Linnell, Craig Prince, and Valentin Razmov. 2007. Supporting active learning and example based instruction with classroom technology. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (Covington, Kentucky, USA) *(SIGCSE '07)*. Association for Computing Machinery, New York, NY, USA, 69–73. https://doi.org/10.1145/1227310.1227338

Michal Armoni. 2009. Reduction in CS: A (Mostly) Quantitative Analysis of Reductive Solutions to Algorithmic Problems. *J. Educ. Resour. Comput.* 8, 4, Article 11 (jan 2009), 30 pages.

Brett A. Becker and Keith Quille. 2019. 50 Years of CS1 at SIGCSE: A Review of the Evolution of Introductory Programming Education Research. 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, 338–344. https://doi.org/10.1145/3287324.3287432

Mahnaz Behroozi, Chris Parnin, and Titus Barik. 2019. Hiring is broken: What do developers say about technical interviews?. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, IEEE Computer Society, Los Alamitos, CA, USA, 1–9.

Patrice Belleville, Steven A. Wolfman, Susanne Bradley, and Cinda Heeren. 2020. Inverted Two-Stage Exams for Prospective Learning: Using an Initial Group Stage to Incentivize Anticipation of Transfer. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. ACM, New York, NY, USA, 720–738.

Jens Bennedssen and Michael E. Caspersen. 2008. Abstraction Ability as an Indicator of Success for Learning Computing Science?. In *Proceedings of the Fourth International Workshop on Computing Education Research* (Sydney, Australia) *(ICER '08)*. ACM, New York, NY, USA, 15–26.

Steven Bird and James R. Curran. 2006. Building a Search Engine to Drive Problem-Based Learning. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy) *(ITICSE '06)*. ACM, New York, NY, USA, 153–157.

Florent Bouchez-Tichadou. 2018. Problem solving to teach advanced algorithms in heterogeneous groups. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018)*. Association for Computing Machinery, New York, NY, USA, 200–205. https://doi.org/10.1145/3197091.3197147

Noelle Brown, Koriann South, and Eliane S. Wiese. 2022. The Shortest Path to Ethics in AI: An Integrated Assignment Where Human Concerns Guide Technical Decisions. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1* (Lugano and Virtual Event, Switzerland) *(ICER '22)*. ACM, New York, NY, USA, 344–355.

John W. Coffey. 2013. Integrating Theoretical and Empirical Computer Science in a Data Structures Course. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. ACM, New York, NY, USA, 23–28.

Christian Collberg, Stephen G. Kobourov, and Suzanne Westbrook. 2004. AlgoVista: An Algorithmic Search Tool in an Educational Setting. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (Norfolk, Virginia, USA) *(SIGCSE '04)*. ACM, New York, NY, USA, 462–466.

Harris M. Cooper. 1988. Organizing knowledge syntheses: A taxonomy of literature reviews. *Knowledge in Society* (1988). https://doi.org/10.1007/BF03177550

Daniel Coore and Daniel Fokum. 2019. Facilitating Course Assessment with a Competitive Programming Platform. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. ACM, New York, NY, USA, 449–455.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2022. *Introduction to Algorithms* (4 ed.). The MIT Press, Cambridge, Massachusetts.

Pierluigi Crescenzi, Emma Enström, and Viggo Kann. 2013. From Theory to Practice: NP-Completeness for Every CS Student. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education* (Canterbury, England, UK) *(ITiCSE '13)*. ACM, New York, NY, USA, 16–21.

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 (Raleigh, North Carolina, USA) *(SIGCSE '12)*. ACM, New York, NY, USA, 21–26.

Holger Danielsiek, Laura Toma, and Jan Vahrenhold. 2017. An Instrument to Assess Self-Efficacy in Introductory Algorithms Courses. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (Tacoma, Washington, USA) *(ICER '17)*. ACM, New York, NY, USA, 217–225.

Debzani Deb, Muztaba Fuad, James Etim, and Clay Gloster. 2018. MRS: Automated Assessment of Interactive Classroom Exercises. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. ACM, New York, NY, USA, 290–295.

Debzani Deb, Mohammad Muztaba Fuad, and Mallek Kanan. 2017. Creating Engaging Exercises With Mobile Response System (MRS). In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. ACM, New York, NY, USA, 147–152.

Katherine Deibel. 2005. Team formation methods for increasing interaction during in-class group work. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Caparica, Portugal) *(ITiCSE '05)*. Association for Computing Machinery, New York, NY, USA, 291–295. `https://doi.org/10.1145/1067445.1067525`

Daniel L Dinsmore, Patricia A Alexander, and Sandra M Loughlin. 2008. Focusing the conceptual lens on metacognition, self-regulation, and self-regulated learning. *Educational psychology review* 20 (2008), 391–409.

James W. Drisko and Tina Maschi. 2016. *Content Analysis*. Oxford University Press.

Yuemeng Du, Andrew Luxton-Reilly, and Paul Denny. 2020. A Review of Research on Parsons Problems. In *Proceedings of the Twenty-Second Australasian Computing Education Conference* (Melbourne, VIC, Australia) *(ACE'20)*. Association for Computing Machinery, New York, NY, USA, 195–202. `https://doi.org/10.1145/3373165.3373187`

Emma Enström and Viggo Kann. 2017. Iteratively Intervening with the "Most Difficult" Topics of an Algorithms and Complexity Course. *ACM Trans. Comput. Educ.* 17, 1, Article 4 (jan 2017), 38 pages.

Barbara J. Ericson, Paul Denny, James Prather, Rodrigo Duran, Arto Hellas, Juho Leinonen, Craig S. Miller, Briana B. Morrison, Janice L. Pearce, and Susan H. Rodger. 2022. Parsons Problems and Beyond: Systematic Literature Review and Empirical Study Designs. In *Proceedings of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education* (Dublin, Ireland) *(ITiCSE-WGR '22)*. Association for Computing Machinery, New York, NY, USA, 191–234. https://doi.org/10.1145/3571785.3574127

Ali Erkan. 2018. The Educational Insights and Opportunities Afforded by the Nuances of Prim's and Kruskal's MST Algorithms. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018)*. ACM, New York, NY, USA, 129–134.

Ali Erkan and Diyan Gochev. 2008. An Image Background Detection Project for a Visual Exploration of DFS and BFS. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '08)*. ACM, New York, NY, USA, 483–487.

Mohammed F. Farghally, Kyu Han Koh, Hossameldin Shahin, and Clifford A. Shaffer. 2017. Evaluating the Effectiveness of Algorithm Analysis Visualizations. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 201–206. https://doi.org/10.1145/3017680.3017698

Tommy Färnqvist and Fredrik Heintz. 2016. Competition and Feedback through Automated Assessment in a Data Structures and Algorithms Course. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (Arequipa, Peru) *(ITiCSE '16)*. Association for Computing Machinery, New York, NY, USA, 130–135. https://doi.org/10.1145/2899415.2899454

J. L. Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological Bulletin* 76 (1971). Issue 5. https://doi.org/10.1037/h0031619

Diana Franklin, Phillip Conrad, Bryce Boe, Katy Nilsen, Charlotte Hill, Michelle Len, Greg Dreschler, Gerardo Aldana, Paulo Almeida-Tanaka, Brynn Kiefer, Chelsea Laird, Felicia Lopez, Christine Pham, Jessica Suarez, and Robert Waite. 2013. Assessment of computer science learning in a scratch-based outreach program. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing

Machinery, New York, NY, USA, 371–376. https://doi.org/10.1145/2445196.2445304

Iris Gaber, Amir Kirsh, and David Statter. 2023. Studied Questions in Data Structures and Algorithms Assessments. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. ACM, New York, NY, USA, 250–256.

Ginés Gárcia-Mateos and José Luis Fernández-Alemán. 2009. A Course on Algorithms and Data Structures Using On-Line Judging. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education* (Paris, France) *(ITiCSE '09)*. ACM, New York, NY, USA, 45–49.

Edward F. Gehringer and Carolyn S. Miller. 2009. Student-generated active-learning exercises. *SIGCSE Bull.* 41, 1 (mar 2009), 81–85. https://doi.org/10.1145/1539024.1508897

David Ginat. 2001. Metacognitive Awareness Utilized for Learning Control Elements in Algorithmic Problem Solving. In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education* (Canterbury, United Kingdom) *(ITiCSE '01)*. ACM, New York, NY, USA, 81–84.

David Ginat. 2003. The greedy trap and learning from mistakes. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education* (Reno, Navada, USA) *(SIGCSE '03)*. Association for Computing Machinery, New York, NY, USA, 11–15. https://doi.org/10.1145/611892.611920

David Ginat. 2004. Do senior CS students capitalize on recursion?. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Leeds, United Kingdom) *(ITiCSE '04)*. Association for Computing Machinery, New York, NY, USA, 82–86. https://doi.org/10.1145/1007996.1008020

David Ginat and Yoav Blau. 2017. Multiple Levels of Abstraction in Algorithmic Problem Solving. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. ACM, New York, NY, USA, 237–242.

Lasse Hakulinen. 2011. Using Serious Games in Computer Science Education. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '11)*. ACM, New York, NY, USA, 83–88.

Yeajin Ham and Brandon Myers. 2021. Learning from Team Quizzes in CS2. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 362–368. https://doi.org/10.1145/3408877.3432412

Sarah Heckman, Jeffrey C. Carver, Mark Sherriff, and Ahmed Al-zubidy. 2021. A Systematic Literature Review of Empiricism and Norms of Reporting in Computing Education Research Literature. *ACM Trans. Comput. Educ.* 22, 1, Article 3 (oct 2021), 46 pages. https://doi.org/10.1145/3470652

Jason J Holdsworth and Siu Man Lui. 2009. GPS-Enabled Mobiles for Learning Shortest Paths: A Pilot Study. In *Proceedings of the 4th International Conference on Foundations of Digital Games* (Orlando, Florida) *(FDG '09)*. ACM, New York, NY, USA, 86–90.

Hadi Hosseini, Maxwell Hartt, and Mehrnaz Mostafapour. 2019. Learning IS Child's Play: Game-Based Learning in Computer Science Education. *ACM Trans. Comput. Educ.* 19, 3, Article 22 (jan 2019), 18 pages.

Hadi Hosseini and Laurel Perweiler. 2019. Are You Game?. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. ACM, New York, NY, USA, 866–872.

Christopher D Hundhausen, Sarah A Douglas, and John T Stasko. 2002. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages & Computing* 13, 3 (2002), 259–290.

Cruz Izu and Brad Alexander. 2018. Using unstructured practice plus reflection to develop programming/problem-solving fluency. In *Proceedings of the 20th Australasian Computing Education Conference* (Brisbane, Queensland, Australia) *(ACE '18)*. Association for Computing Machinery, New York, NY, USA, 25–34. https://doi.org/10.1145/3160489.3160496

Cruz Izu, Cheryl Pope, and Amali Weerasinghe. 2017. On the Ability to Reason About Program Behaviour: A Think-Aloud Study. In *Proceedings of the 2017*

ACM Conference on Innovation and Technology in Computer Science Education (Bologna, Italy) *(ITiCSE '17)*. ACM, New York, NY, USA, 305–310.

Duane J. Jarc, Michael B. Feldman, and Rachelle S. Heller. 2000. Assessing the benefits of interactive prediction using Web-based algorithm animation courseware. In *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education* (Austin, Texas, USA) *(SIGCSE '00)*. Association for Computing Machinery, New York, NY, USA, 377–381. https://doi.org/10.1145/330908.331889

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*. ACM, New York, NY, USA.

Maria Kallia, Quintin Cutts, and Nicola Looker. 2022. When Rhetorical Logic Meets Programming: Collective Argumentative Reasoning in Problem-Solving in Programming. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1* (Lugano and Virtual Event, Switzerland) *(ICER '22)*. ACM, New York, NY, USA, 120–134.

Barbara Kitchenham and Stuart Charters. 2007a. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Technical Report EBSE-2007-01. Software Engineering Group, Keele University and Department of Computer Science, University of Durham, Salt Lake City, UT.

Barbara Kitchenham and Stuart Charters. 2007b. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Technical Report EBSE-2007-01. Software Engineering Group, Keele University and Department of Computer Science, University of Durham, Salt Lake City, UT.

Amruth N. Kumar. 2012. A study of stereotype threat in computer science. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (Haifa, Israel) *(ITiCSE '12)*. Association for Computing Machinery, New York, NY, USA, 273–278. https://doi.org/10.1145/2325296.2325361

Barry L. Kurtz, James B. Fenwick, Rahman Tashakkori, Ahmad Esmail, and Stephen R. Tate. 2014. Active Learning during Lecture Using Tablets. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA) *(SIGCSE '14)*. ACM, New York, NY, USA, 121–126.

Ming-Han Lee and Guido Rößling. 2011. Toward Replicating Handmade Algorithm Visualization Behaviors in a Digital Environment: A Pre-Study. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* (Darmstadt, Germany) *(ITiCSE '11)*. ACM, New York, NY, USA, 198–202.

Abe Leite and Saúl A. Blanco. 2020. Effects of Human vs. Automatic Feedback on Students' Understanding of AI Concepts and Programming Style. 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, 44–50. https://doi.org/10.1145/3328778.3366921

Shu Lin, Na Meng, Dennis Kafura, and Wenxin Li. 2021. PDL: Scaffolding Problem Solving in Programming Courses. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (Virtual Event, Germany) *(ITiCSE '21)*. ACM, New York, NY, USA, 185–191.

Dastyni Loksa, Amy J. Ko, Will Jernigan, Alannah Oleson, Christopher J. Mendez, and Margaret M. Burnett. 2016. Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 1449–1461. https://doi.org/10.1145/2858036.2858252

Joan M. Lucas. 2015. Illustrating the Interaction of Algorithms and Data Structures Using the Matching Problem. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Kansas City, Missouri, USA) *(SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 247–252. https://doi.org/10.1145/2676723.2677212

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, New York, NY, USA, 284–290.

Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A Becker, Michail Giannakos, Amruth N Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard,

et al. 2018. Introductory programming: a systematic literature review. In *Proceedings companion of the 23rd annual ACM conference on innovation and technology in computer science education*. Association for Computing Machinery, New York, NY, USA, 55–106.

Lauri Malmi, Judy Sheard, Päivi Kinnunen, Simon, and Jane Sinclair. 2019. Computing Education Theories: What Are They and How Are They Used?. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. ACM, New York, NY, USA, 187–197.

Byron B. Marshall, Hsinchun Chen, Rao Shen, and Edward A. Fox. 2006. Moving Digital Libraries into the Student Learning Space: The GetSmart Experience. *J. Educ. Resour. Comput.* 6, 1 (mar 2006), 2–es.

Lester I. McCann. 2004. Contemplate Sorting with Columnsort. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (Norfolk, Virginia, USA) *(SIGCSE '04)*. ACM, New York, NY, USA, 166–169.

Dinesh Mehta, Tina Kouri, and Irene Polycarpou. 2012. Forming project groups while learning about matching and network flows in algorithms. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (Haifa, Israel) *(ITiCSE '12)*. Association for Computing Machinery, New York, NY, USA, 40–45. https://doi.org/10.1145/2325296.2325310

Patricia H. Miller, Frank S. Kessel, and John H. Flavell. 1970. Thinking about People Thinking about People Thinking about...: A Study of Social Cognitive Development. *Child Development* 41, 3 (1970), 613–623. http://www.jstor.org/stable/1127211

Diba Mirza, Phillip T. Conrad, Christian Lloyd, Ziad Matni, and Arthur Gatin. 2019. Undergraduate Teaching Assistants in Computer Science: A Systematic Literature Review. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. Association for Computing Machinery, New York, NY, USA, 31–40. https://doi.org/10.1145/3291279.3339422

Thomas L. Naps, James R. Eagan, and Laura L. Norton. 2000. JHAVÉ—an Environment to Actively Engage Students in Web-Based Algorithm Visualizations. In *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education* (Austin, Texas, USA) *(SIGCSE '00)*. ACM, New York, NY, USA, 109–113.

Alannah Oleson, Benjamin Xie, Jean Salac, Jayne Everson, F. Megumi Kivuva, and Amy J. Ko. 2022. A Decade of Demographics in Computing Education Research: A Critical Review of Trends in Collection, Reporting, and Use. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1* (Lugano and Virtual Event, Switzerland) *(ICER '22)*. ACM, New York, NY, USA, 323–343.

Roy P. Pargas. 2006. Reducing Lecture and Increasing Student Activity in Large Computer Science Courses. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy) *(ITICSE '06)*. ACM, New York, NY, USA, 3–7.

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* (Denver, Colorado, USA) *(SIGCSE '13)*. ACM, New York, NY, USA, 29–34.

David Pengelley, Inna Pivkina, Desh Ranjan, and Karen Villaverde. 2006. A Project in Algorithms Based on a Primary Historical Source about Catalan Numbers. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (Houston, Texas, USA) *(SIGCSE '06)*. ACM, New York, NY, USA, 318–322.

Vinhthuy Phan and Eric Hicks. 2018. Code4Brownies: An Active Learning Solution for Teaching Programming and Problem Solving in the Classroom. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018)*. ACM, New York, NY, USA, 153–158.

James Prather, Brett A Becker, Michelle Craig, Paul Denny, Dastyni Loksa, and Lauren Margulieux. 2020. What do we think we think we are doing? Metacognition and self-regulation in programming. In *Proceedings of the 2020 ACM conference on international computing education research*. Association for Computing Machinery, New York, NY, USA, 2–13.

James Prather, Raymond Pettit, Brett A. Becker, Paul Denny, Dastyni Loksa, Alani Peters, Zachary Albrecht, and Krista Masci. 2019. First Things First: Providing Metacognitive Scaffolding for Interpreting Problem Prompts. 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, 531–537. https://doi.org/10.1145/3287324.3287374

Justus Randolph. 2009. A Guide to Writing the Dissertation Literature Review. *Practical Assessment, Research, and Evaluation* 14 (2009). Issue 1.

Richard Rasala, Viera K. Proulx, and Harriet J. Fell. 1994. From animation to analysis in introductory computer science. In *Proceedings of the Twenty-Fifth SIGCSE Symposium on Computer Science Education* (Phoenix, Arizona, USA) *(SIGCSE '94)*. Association for Computing Machinery, New York, NY, USA, 61–65. https://doi.org/10.1145/191029.191057

Samuel A. Rebelsky. 2005. The New Science Students in Too Much, Too Soon an Abbreviated, Accelerated, Constructivist, Collaborative, Introductory Experience in CS. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* (St. Louis, Missouri, USA) *(SIGCSE '05)*. ACM, New York, NY, USA, 312–316.

Adam Rosenstein, Aishma Raghu, and Leo Porter. 2020. Identifying the Prevalence of the Impostor Phenomenon Among Computer Science Students. 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, 30–36. https://doi.org/10.1145/3328778.3366815

Kate Sanders, Judy Sheard, Brett A. Becker, Anna Eckerdal, Sally Hamouda, and Simon. 2019. Inferential Statistics in Computing Education Research: A Methodological Review. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. Association for Computing Machinery, New York, NY, USA, 177–185. https://doi.org/10.1145/3291279.3339408

Clifford A. Shaffer, Matthew Cooper, and Stephen H. Edwards. 2007. Algorithm Visualization: A Report on the State of the Field. *SIGCSE Bull.* 39, 1 (mar 2007), 150–154.

Clifford A. Shaffer, Matthew L. Cooper, Alexander Joel D. Alon, Monika Akbar, Michael Stewart, Sean Ponce, and Stephen H. Edwards. 2010. Algorithm Visualization: The State of the Field. *ACM Trans. Comput. Educ.* 10, 3, Article 9 (aug 2010), 22 pages.

Judy Sheard, S. Simon, Margaret Hamilton, and Jan Lönnberg. 2009. Analysis of research into the teaching and learning of programming. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop* (Berkeley,

CA, USA) *(ICER '09)*. Association for Computing Machinery, New York, NY, USA, 93–104. https://doi.org/10.1145/1584322.1584334

Steven J. Spencer, Claude M. Steele, and Diane M. Quinn. 1999. Stereotype Threat and Women's Math Performance. *Journal of Experimental Social Psychology* 35, 1 (1999), 4–28. https://doi.org/10.1006/jesp.1998.1373

John T. Stasko. 1997. Using Student-Built Algorithm Animations as Learning Aids. In *Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education* (San Jose, California, USA) *(SIGCSE '97)*. ACM, New York, NY, USA, 25–29.

Ahmad Taherkhani, Ari Korhonen, and Lauri Malmi. 2012. Automatic Recognition of Students' Sorting Algorithm Implementations in a Data Structures and Algorithms Course. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '12)*. ACM, New York, NY, USA, 83–92.

David Scot Taylor, Andrei F. Lurie, Cay S. Horstmenn, Menko B. Johnson, Sean K. Sharma, and Edward C. Yin. 2009. Predictive vs. Passive Animation Learning Tools. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (Chattanooga, TN, USA) *(SIGCSE '09)*. ACM, New York, NY, USA, 494–498.

James D. Teresco, Razieh Fathi, Lukasz Ziarek, MariaRose Bamundo, Arjol Pengu, and Clarice F. Tarbay. 2018. Map-Based Algorithm Visualization with METAL Highway Data. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. ACM, New York, NY, USA, 550–555.

Laura Toma and Jan Vahrenhold. 2018. Self-Efficacy, Cognitive Load, and Emotional Reactions in Collaborative Algorithms Labs - A Case Study. In *Proceedings of the 2018 ACM Conference on International Computing Education Research* (Espoo, Finland) *(ICER '18)*. ACM, New York, NY, USA, 1–10.

Tammy VanDeGrift. 2017. POGIL Activities in Data Structures: What do Students Value?. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 597–602. https://doi.org/10.1145/3017680.3017697

J. Ángel Velázquez-Iturbide. 2012. Refinement of an Experimental Approach Tocomputer-Based, Active Learning of Greedy Algorithms. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (Haifa, Israel) *(ITiCSE '12)*. ACM, New York, NY, USA, 46–51.

J. Ángel Velázquez-Iturbide. 2013. An Experimental Method for the Active Learning of Greedy Algorithms. *ACM Trans. Comput. Educ.* 13, 4, Article 18 (nov 2013), 23 pages. https://doi.org/10.1145/2534972

J. Ángel Velázquez-Iturbide. 2019. Students' Misconceptions of Optimization Algorithms. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE '19)*. ACM, New York, NY, USA, 464–470.

J. Ángel Velázquez-Iturbide, Isidoro Hernán-Losada, and Antonio Pérez-Carrasco. 2016. A "Multiple Executions" Technique of Visualization. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (Arequipa, Peru) *(ITiCSE '16)*. ACM, New York, NY, USA, 59–64.

J. Ángel Velázquez-Iturbide and Antonio Pérez-Carrasco. 2016. Systematic Development of Dynamic Programming Algorithms Assisted by Interactive Visualization. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (Arequipa, Peru) *(ITiCSE '16)*. ACM, New York, NY, USA, 71–76.

Valdemar Švábenský, Jan Vykopal, and Pavel Čeleda. 2020. What Are Cybersecurity Education Papers About? A Systematic Literature Review of SIGCSE and ITiCSE Conferences. 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, 2–8. https://doi.org/10.1145/3328778.3366816

Robbie Weber. 2023. Using Alternative Grading in a Non-Major Algorithms Course. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. ACM, New York, NY, USA, 638–644.

Titus Winters and Tom Payne. 2006. Closing the Loop on Test Creation: A Question Assessment Mechanism for Instructors. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (Houston, Texas, USA) *(SIGCSE '06)*. ACM, New York, NY, USA, 169–170.

Anthony Wirth and Michael Bertolacci. 2006. New algorithms research for first year students. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy) *(ITICSE '06)*. Association for Computing Machinery, New York, NY, USA, 128–132. https://doi.org/10.1145/1140124.1140160

Jason Xia and Craig Zilles. 2023. Using Context-Free Grammars to Scaffold and Automate Feedback in Precise Mathematical Writing. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. ACM, New York, NY, USA, 479–485.

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* (Baltimore, Maryland, USA) *(SIGCSE '18)*. ACM, New York, NY, USA, 556–561.

Andreas Zeller. 2000. Making Students Read and Review Code. In *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSEconference on Innovation and Technology in Computer Science Education* (Helsinki, Finland) *(ITiCSE '00)*. ACM, New York, NY, USA, 89–92.

CHAPTER A

APPENDIX

A.1 Dynamic Programming Interview Problems

Below are the problems presented to students participating in the Dynamic Programming interviews. Problems are split by whether it was the participant's first or second interview, and within each interview listed in order of increasing difficulty. See Section 4.3.2.

A.1.1 First Interview

- **Q1: Coin Row.** There are n coins in a row whose values are positive integers $c_1, ..., c_n$. You may pick up coins from this row, but you can not pick up two adjacent coins. Design a Dynamic Programming algorithm that finds the largest sum of values you can pick up.

- **Q2: Gift Packing.** You have a box of size V, and gifts of positive integer size $p_1, ..., p_n$. You want to give your friend a box full of gifts. Design a Dynamic Programming algorithm to determine whether there exists a subset of the presents that has total size exactly equal to the size of the box.

- **Q3: Text Reconstruction.** You've intercepted a secret message that consists of the letters $m_1, ..., m_n$, but there are no spaces or punctuation so you're not sure whether it was supposed to be a sequence of words. For any string of letters s, you can call `is_valid`(s) to determine whether the string is a valid word

in constant time. Design a Dynamic Programming algorithm that determines whether the message can be split into a sequence of valid words.

A.1.2 Second Interview

- **Q1: Vacation Planning.** You're planning a big trip, but the place you're visiting is known for its erratic weather. For each of the n days that you could visit, your enjoyment on day i would be some integer w_i (not necessarily positive). Find the sequence of consecutive days to visit that maximizes the sum of your enjoyment.

- **Q2: Roadtrip Planning.** You've decided to make the long drive to your vacation destination. Along the highway, you will pass through n gas stations. At gas station i, you can pay p_i for each mile's worth of gas, and your tank can hold enough gas to travel C miles. The distance between station $i-1$ and station i is given by d_i. You start at station 0 with 0 gallons of gas. Determine the minimum amount you need to pay to successfully reach gas station n.

- **Q3: Lazy Meetings.** At your job, there are n meetings scheduled today. Meeting i starts at time a_i, ends at time b_i, and costs you e_i energy to attend. You have been asked to attend every meeting, but some of the meetings overlap with each other, so you will need to skip some. You may only skip a meeting if you attend a meeting that overlaps with it. Design a Dynamic Programming algorithm that finds the least amount of energy you need to spend to get through your meetings today.