A Survey of Undergraduate Theory of Computation Curricula in the United States

Ryan E. Dougherty*
ryan.dougherty@westpoint.edu
United States Military Academy
West Point, New York, USA

Tim Randolph†
trandolph@g.hmc.edu
Harvey Mudd College
Claremont, California, USA

Tzu-Yi Chen
tzuyi.chen@pomona.edu
Pomona College
Claremont, California, USA

Jeff Erickson
jeffe@illinois.edu
University of Illinois
Urbana-Champaign
Urbana, Illinois, USA

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

Dennis Komm
dennis.komm@inf.ethz.ch
ETH Zürich
Zürich, Switzerland

Jonathan Liu
jonliu@uchicago.edu
University of Chicago
Chicago, Illinois, USA

Timothy Ng
timng@uchicago.edu
University of Chicago
Chicago, Illinois, USA

Ana Smaranda Sandu
smaranda.sandu@wellesley.edu
Wellesley College
Wellesley, Massachusetts, USA

Michael Shindler
mikes@uci.edu
UC Irvine
Irvine, California, USA

Edward Talmage
elt006@bucknell.edu
Bucknell University
Lewisburg, Pennsylvania, USA

Thomas Zeume
thomas.zeume@rub.de
Ruhr University Bochum
Bochum, Germany

Abstract

Theory of computation (ToC), the subfield of theoretical computer science concerned with automata, formal languages, grammars, computability, and the foundations of complexity theory, among other topics, is a staple of undergraduate computer science programs. Nevertheless ToC pedagogy is severely understudied from the perspective of computing education research (CER).

We surveyed institutions of higher education in the United States that awarded at least one bachelor's degree in computer science during the 2022-2023 academic year in order to learn about their ToC curricula. We asked questions designed to determine:

- what topics are taught in ToC courses,
- where ToC courses appear in the curriculum,
- whether and in what contexts ToC courses are required,
- how ToC courses are organized, including student composition, course staff demographics, and class format, and
- which pedagogical tools are most commonly deployed in ToC classrooms.

We received responses from faculty members representing a diverse set of 166 institutions. Our major findings include (1) a list of the most frequently covered ToC topics indexed by rate of coverage and position within the curriculum, (2) an overview of the most common attributes of ToC courses and classrooms (course staff, instruction format, section size, etc.,), and (3) that very few courses use ToC-specific digital teaching tools.

CCS Concepts

• **Social and professional topics** → **Computing education**; • **Theory of computation**;

Keywords

Theory of computation, CS course design, CS pedagogy, technical CS course

ACM Reference Format:
Ryan E. Dougherty, Tim Randolph, Tzu-Yi Chen, Jeff Erickson, Matthew Ferland, Dennis Komm, Jonathan Liu, Timothy Ng, Ana Smaranda Sandu, Michael Shindler, Edward Talmage, and Thomas Zeume. 2024. A Survey of Undergraduate Theory of Computation Curricula in the United States . In *2024 Working Group Reports on 1st ACM Virtual Global Computing Education Conference (SIGCSE Virtual–WGR 2024), December 5–8, 2024, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3708550.3730559

1 Introduction

Theory of computation (ToC) provides a rigorous, formal foundation for undergraduate courses across the discipline of computer science (CS). As such, many undergraduate CS programs require their majors to demonstrate proficiency in ToC. In this report, we present the results of a national survey of colleges and universities focused on ToC curricula.

*Working group co-leader.
†Working group co-leader.

SIGCSE Virtual–WGR 2024, December 5–8, 2024, Virtual Event, USA

ACM ISBN 979-8-4007-1408-5/2024/12
https://doi.org/10.1145/3708550.3730559

As this is the first major survey on the subject, we do not claim to provide a complete picture of ToC teaching. However, we hope to provide a useful baseline against which college educators can compare their ToC curricula and provide a foundation for future collaborations among ToC educators and researchers.

Our survey attempts to address the following three research questions. In the context of institutions in the United States that offer bachelor's degrees in CS or closely related fields of study, we asked:

- **RQ1.** What specific concepts and topics do faculty teach in "ToC courses"?
- **RQ2.** What are the most common administrative and classroom features of "ToC courses," (number and demographics of course staff, format of instruction, section size, etc.)?
- **RQ3.** What digital and other pedagogical tools do faculty use when teaching "ToC courses"?

The first difficulty in answering these questions is reaching a satisfactory definition of a "ToC course." There is no single list of the topics that comprise "theory of computation," and the use of the term is not clearly distinguished from "theory of computing," "theoretical computer science," and other near-synonyms. Courses labeled "theory of computation" may include material on formal languages, grammars, regular expressions, automata theory, Boolean circuits, computability, complexity theory, and related material from fields such as algorithms, programming languages, and natural language processing, among many others. Among the 12 members of our working group, all of whom teach or have taught some version of a ToC course, we reached consensus on a list of "ToC topics" inspired by the "Computational Models and Formal Languages" knowledge unit of the 2023 ACM Computer Science Curriculum [19]. This list of topics effectively defines the scope of ToC for the purposes of our survey. (We take no position on the "correct" scope of ToC, but explain our usage and the focus of the survey in more detail in Section 3.3.)

We prompted respondents to identify a small set of courses in which students first experienced clusters of these topics in significant detail, and treated these as our representative "ToC courses." We then asked survey respondents a variety of questions about the ToC courses at their institutions, including questions concerning

- how ToC courses fit into the progression of the CS major, including their prerequisites;
- how many ToC courses are offered, how frequently these courses are offered, and whether they are required;
- ToC course size, course staff, and format of instruction;
- role and professional focus of the primary instructor; and
- what pedagogical tools are used in ToC courses.

We directed our questions to senior faculty members at some 1,058 U.S. colleges and universities that offer bachelor's degrees in computer science and received responses from 166. Section 3 includes information about our criteria for including institutions and selecting institutional representatives to take the survey.

The remainder of this report is organized as follows. Section 2 covers related work, including similar survey research and existing computing education research (CER) on ToC curricula and pedagogy. Section 3 details our research methods, including information about survey scope, respondents, data collection, survey structure, and limitations. Section 4 presents the results of our survey. Section 5 discusses the results in context, summarizes the implications of our results for our three research questions, and offers some speculative conclusions. Section 6 concludes the report with open questions and future work prompted by this research.

2 Related Work

To the best of our knowledge this is the first survey of ToC courses from the perspective of CER. Most similar in spirit to our survey is that of Luu et al. [20], who conducted a survey of algorithms courses at undergraduate institutions. Algorithms and ToC are related but distinct sub-areas of CS: to make a rough distinction using the terminology of the ACM/IEEE/AAAI CS2023 Curricular Guidelines, ToC centers on the "Computational Models and Formal Languages" knowledge unit, while Algorithms centers on the "Algorithmic Strategies" knowledge unit [19].

Existing literature at the intersection of ToC and CER mainly concerns course design, classroom experiments, visualization of theory concepts and auto-grading systems. In the subsections below we provide a brief and noncomprehensive summary of existing work in this area. A full survey of computing education research in ToC would be a useful complement to the current work.

2.1 ToC Course Design

Several authors have explored alternative designs for ToC courses and for courses outside of computer science that incorporate ToC topics. These are immediately relevant to our **RQ2** (and perhaps **RQ1**) although they may not be representative of typical ToC experiences. For example, Berque et al. introduced a split-screen approach for teaching ToC in which a question was posed to students on an upper screen and students drew a diagram underneath [4]. East [14] developed a framework for integrating a theory-based perspective throughout a CS curriculum. Del Vado Vírseda went further by exploring the pedagogical benefits of introducing ToC material into other subject areas, such as economics and the social sciences [10]. The same author experimented with teaching undecidability results from computability theory in physics and mathematics courses in order to increase student interest in and future success in learning ToC [8, 9].

Blumenthal [5] performed a curricular survey of computability courses to determine the extent to which computer science majors are exposed to Turing Machines and other equally powerful models of computation. Based on course descriptions, syllabi, and course materials, he concluded that a majority of CS majors are *not* exposed to a definition of computability involving Turing Machines,[1] and very few are exposed to alternative models, which informs **RQ1**. He proposed three changes to the ACM CS Curricular 2013 report based on his results: (1) making the Chomsky Hierarchy, Turing machines, Church-Turing thesis, and Uncomputability required; (2) modifying the formal computation topic to include Turing machines and some other equivalent formal model; and (3) changing the name of the "Algorithms and Complexity" knowledge area to "Algorithmic Foundations".

[1] This appears to conflict with our survey results: a majority of respondents to this survey indicated that their majors were exposed to Turing machines in at least one required course (Table 1). One possible explanation for this discrepancy is sampling bias (see Section 3.4).

2.2 ToC Classroom Experiments

ToC classrooms have inspired a wide variety of classroom experiments; we cite a small sample below.

Lack of engagement in ToC courses has been a common challenge for educators. Sigman observed that engaging students in ToC is "notoriously difficult" and employed a discovery learning technique to boost engagement in a ToC course [27]. Pillay [23], with similar motivation to Sigman, conducted a study to understand points at which students struggled in ToC and attributed student difficulties to immature problem-solving skills and issues with generalizing specific observations to abstract theorems such as the Pumping Lemma. Analyzing and attempting to remedy curricular features that students often find difficult requires a baseline understanding of the common curricular features of ToC classes (**RQ2**) and topics taught (**RQ1**).

Others have improved engagement by incorporating presentation skills and otherwise encouraging student agency. Dougherty [12, 13] developed a scaffolded "mock conference" experience in his ToC class; additionally, he implemented a "backwards" ordering of concepts in a ToC course [11]. Randolph [24] implemented pedagogical strategies of participatory governance in the context of an intermediate ToC class. Experimental teaching approaches both challenge existing administrative and classroom features of ToC courses (**RQ2**) and explore the potential of innovative pedagogical strategies (**RQ3**).

Also relevant to **RQ3** are the attempts of some authors to leverage programming and implementation, more traditional pedagogical tools in computer science, to teach ToC. For example, Morazán has designed an introduction to formal languages and automata theory that emphasizes programming implementations [22]. Krone [18], in an older work, had students write programs to understand the Church-Turing thesis more effectively.

2.3 ToC Tools: Visualizations and Auto-Grading Systems

Theory of computation introduces students to several complex discrete mathematical objects, notably automata, and students encountering these topics for the first time struggle to build intuition for how they work. Several authors have addressed this problem by building tools for visualizing ToC concepts, and their efforts are relevant to our **RQ3**. For example, Mohammed et al. [21] developed visualizations and auto-graded exercises for a ToC course and analyzed student performance. Bennett-Manke et al. [3] (see also a follow-up work due to Baker et al. [2]) developed an automatic visualizer for automata simulation that connects the formal definition of an automaton with its state diagram.

In addition to visualization tools, ToC instructors have developed ToC-specific auto-grading tools to evaluate student work more efficiently. Alur et al. [1] first introduced a method for automatically grading finite automata questions at-scale. Robson et al. [25] built a tool for creating finite automata questions that can be auto-graded; similar work was performed by Erickson et al. [15], again with finite automata. Schmellenkamp et al. [26] built a tool for auto-graded exercises on formal languages and computational reductions.

3 Research Methods

In this section, we summarize our target survey population and the criteria for survey participation we used to generate a list of institutions approximating the target. We then describe our data collection procedure and the design of the survey instrument itself. Finally, we briefly discuss the limitations of our survey data.

3.1 Survey Scope and Criteria for Participation

Our intended survey population is those colleges and universities in the U.S. that teach ToC and/or closely related topics in CS. As a proxy for this, we surveyed institutions that offer bachelor's degrees in CS or a closely related field. This required generating a list of institutions and collecting the contact information of respondents who could provide reliable information about their institution's CS program.

To create the initial list of colleges and universities we used data from the *Integrated Postsecondary Education Data System* (IPEDS). This data is provided by the National Center for Education Statistics (NCES), the agency within the U.S. Department of Education in charge of collecting and publishing statistics about education in the U.S. IPEDS contains most of the information the NCES publishes about higher education, and virtually every institution that can be considered a college or university participates in IPEDS data collection. The institutions participating can be roughly divided into three categories:

- Institutions that have *Program Participation Agreements* (PPAs). Every institution with a PPA is required to provide institutional data to the NCES for publishing in IPEDS. Every institution that directly participates in any of the Title IV Federal Financial Student Aid programs, such as the Pell Grant or Work Study programs, must have a PPA. Programs that fail to report on time are fined, so these institutions have a nearly 100% response rate.
- Satellite campuses, such as the many Pennsylvania State University campuses, which do not have a PPA of their own, but are "under" an institution with a PPA, can elect to either report individually or report in aggregate under their parent institution. This is notable, as Carnegie Classification is not given to satellite campuses without their own PPA, making the list of IPEDs participants broader than the Carnegie Classification list.
- A relatively small number of non-Title IV participating institutions also participate in IPEDS.

We worked with data from the 2022–2023 academic year, the most recent available at the time of our survey. This data includes approximately 6,100 Title IV participating institutions and 300 other institutions, for a total of 6,400. We then reduced this list to the set of institutions that offer a bachelor's degree in some field of computing using IPEDS file "C2023_A," which records information about each degree granted by every institution in the 2022–2023 academic year. For our study, we required that an institution have a degree with the following characteristics:

- The degree had to have a 2 digit CIP code of "11" ("Computer and Information Sciences and Support Services"). This category includes Computer Science, Informatics, Software Engineering, and several other computing degrees.

- The degree had to have an award level code of "05" which corresponds to a bachelor's degree.

After applying this process, we were left with 1,451 institutions. For each of these, we gathered the following from the IPEDS database:

- Institution name,
- Institution nickname(s),
- Institution city and state, and
- Institution homepage URL.

These 1,451 institutions made up our preliminary list of target institutions. For each institution on the preliminary list, we manually gathered contact email addresses according to the following protocol: First, we attempted to discover the institutional email address of the chair of the department offering the major in CS. If we failed to identify the email of a department chair, we attempted to identify the contact information of a faculty member teaching theory of computation. Finally, if we could not identify faculty teaching a ToC course or courses, we attempted to identify the contact information of any senior faculty member, ideally a tenured faculty member at the rank of full professor, non-emeritus. The primary sources for this information were public-facing department web pages, academic directories, course catalogs, and public-facing faculty web pages. We omitted institutions for which we could not find contact information as well as permanently closed institutions and institutions no longer accepting enrollments.

3.2 Data Collection

In total, we recovered contact information for 1,058 institutions (72.9%[2] of the preliminary list). Delivery failed for 29 contact addresses, leaving a total of 1,029 institutions. The most common reasons that we failed to recover contact information for an institution were a very limited or nonexistent web presence, a web presence without publicly listed faculty or faculty contact information, or the lack of any program with a theoretical component (for example, a standalone program in IT Systems Management).

We emailed each institution with a short description of our survey and a link to the survey itself, which was hosted using Google Forms. We instructed recipients to take the survey themselves or pass it along to the member of their department best qualified to answer questions about how their institution teaches ToC. We did not offer compensation or other incentives to respondents. We sent initial emails on the 23rd and 24th of October 2024 and reminder emails on the 5th of November 2024. Among the 1,029 institutions we contacted, we received survey responses from 166 (16.1%). Since many institutions on our preliminary contact list did not appear to offer ToC of any kind in their curriculum,[3] we believe that the effective response rate among institutions teaching ToC may have been much higher than this figure suggests.

3.3 Survey Structure

To organize our list of "theory topics," we grouped topics into three broad categories:

(1) **Automata Theory:** Topics including formal language classes, grammars, and machine models other than Turing machines.
(2) **Computability Theory:** Topics including Turing machines, decidability and undecidability, and formal models equivalent in power to Turing machines.
(3) **Basic Complexity Theory:** Topics such as time and space complexity, complexity classes, and polynomial-time reductions.

Within each category we compiled a list of specific topics that seemed likely to appear in an undergraduate CS curriculum. We intended our list to include the breadth of topics often covered in a first course on theory of computation and to avoid topics primarily covered in algorithms courses and adjacent "Theory B" topics such as programming language semantics and formal logic. Our topic list was influenced by the desire to avoid duplicating existing work, particularly the survey of algorithms courses due to Luu et al. [20]. We also consulted the "Algorithmic Foundations: AL-Models: Computational Models and Formal Languages" list of knowledge units in the ACM/IEEE/AAAI CS2023 Curricular Guidelines [19].[4] The final list of theory topics was achieved by consensus resolution of the 12 instructors in our working group, all of whom teach ToC. (We do not claim any authoritative status for our topic list, and believe excluded topics would be excellent inclusions in a follow-up survey.) Table 1 presents the entire list of topics, grouped according to our three categories.

Our survey consisted of four components: basic information about the respondent's institution, questions about theory of computation topics, questions about theory of computation courses, and questions about course logistics. In order to learn the demographics of the institutions represented in our survey data, we asked questions about the category of the respondent's institution, its size, and the size of the CS major (or nearest equivalent). In the next section, we asked questions about where our theory topics appeared in the curriculum. We also provided an opportunity for respondents to list topics omitted from our list. In the third section, we asked about the courses in which students typically first encounter each topic category (Automata Theory, Computability Theory, and Basic Complexity Theory) in "significant depth". We then asked several follow-up questions about the "ToC courses" respondents named. These questions included information about ToC prerequisites, whether ToC courses are required for CS majors, course size, and use of teaching assistants (TAs). We also asked about the faculty who teach ToC courses and the digital and other pedagogical tools used in ToC classes. Finally, we provided an open-ended prompt so that respondents could include additional information about the survey or any of their responses if desired.

3.4 Survey Limitations

Among institutions in the U.S. that confer bachelor's degrees in CS, our respondents represent the subset of institutions that

(1) have publicly available contact information,
(2) responded to an email invitation during a 4-week period in October/November 2024,

[2]Throughout the paper, we write out percentages to the first decimal place. This does not indicate statistical significance.
[3]On the basis of the publicly-available course catalogs, faculty profiles, and major programs reviewed while compiling the contact list.
[4]This includes nearly all of the topics listed in the 2013 and 2008 ACM Curricular Guidelines [6, 16] as well.

(3) had a faculty member knowledgeable about the institution's (theory of computation) curriculum, and
(4) chose to devote time to respond to our survey.

Each of these items may have introduced selection bias into our data. We were able to contact most (1,029 of 1,451) institutions eligible according to our selection criteria. Those that we were unable to contact were disproportionally smaller, newer, and for-profit institutions, as well as satellite campuses of larger universities. It seems reasonable to assume that respondents able to complete a survey concerned with theory of computation curricula were drawn disproportionately from institutions that regularly teach ToC.

We believe that our sample captures the breadth of ways in which institutions in the United States teach ToC to undergraduates, but did not adjust the data to represent institution or program types proportionally. In particular, our survey treats each *institution* as a single respondent, but does not weight by CS program size. Thus our data may over-represent the frequency of curricular features common at smaller institutions relative to how often the median *student* encounters these curricular features.

3.4.1 Peculiarities of Institutions Teaching ToC. We explicitly restricted our sample of institutions to those that offer bachelors' degrees in CS (see Section 3). This excludes some institutions of higher education teaching CS (for instance, those offering 2-year degrees), including some that may offer theory of computation according to our definition. Moreover, we recognize that the set of schools that offer programs in CS is not the same as the set of schools that offer significant course material in *theoretical* CS, and the set of schools that teach theoretical CS (construed broadly to include algorithms and the theoretical aspects of other subfields such as programming languages and data structures) is a superset of those that teach *theory of computation* as we define it.

Certain programs that fall under the broader heading of Computer and Information Systems are focused on preparing students for specific roles and work environments (for example, programs in IT management, cybersecurity, or systems administration). Although we did not collect data on this point, we suspect that programs in this category are less likely to offer coursework in theory of computation and in theoretical CS more broadly. Because our invitation to participate explicitly requested input from faculty familiar with theory of computation, we expect that our sample is somewhat representative of programs that offer ToC. As such, it may underrepresent programs that focus on job training and overrepresent programs that offer broader curricula based in the liberal arts (this accords with the plurality of respondents in our sample who self-identified as representing a Small Liberal Arts College). Resolving this speculation is beyond the scope of our survey, and we leave it as an open question.

If the set of institutions teaching ToC is unusual, our data on ToC courses might reflect peculiarities of these institutions in addition to features specific to ToC courses. In particular, we observed that sections in ToC courses were relatively small (typically under 30 students) and only rarely taught by adjunct faculty. Both of these statistics might reflect peculiarities of the institutions at which ToC is taught rather than ways in which ToC classes differ from other classes at the same institution.

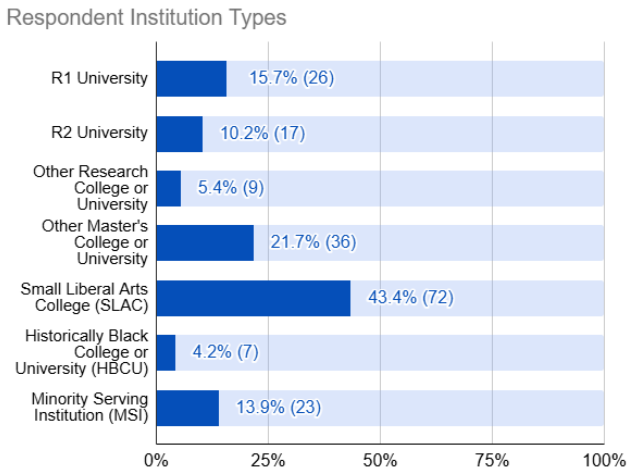

Figure 1: Institution types self-identified by respondents. Respondents could select multiple institution types.

4 Results

4.1 Respondent Demographics

We gave survey respondents the ability to self-identify their institution in several ways, with the option of selecting all descriptors that applied from a list of institution types. Among survey respondents, 72 (43.4%) identified their institution as a Small Liberal Arts College (SLAC). 26 (15.7%) identified their institution as an R1 university (institution awarding doctoral degrees with "very high research activity"), while 17 (10.2%) selected R2 university (institution awarding doctoral degrees with "high research activity") and 36 (21.7%) selected master's college or university. 23 respondents (13.9%) identified their institution as a Minority Serving Institution (MSI), while 7 (4.2%) indicated that their institution was a Historically Black College or University (HBCU). Figure 1 summarizes the frequencies with which survey respondents identified different institution types.

Survey respondents also answered whether their institution offered at least one major or major-equivalent program for undergraduates in CS, and 98.8% responded "Yes" (the final two respondents identified the name of their program that most closely approximated "Computer Science" as "Computer and Information Science" and "Software Engineering", respectively).

We divided institutions into four size categories according to the ranges established by the Carnegie Classification of Institutions of Higher Education: *very small four-year* (0-999), *small four-year* (1000-2999), *medium four-year* (3000-9999), and *large four-year* (10000+). To simplify the prompt, we asked respondents to approximate the "number of undergraduates" at their institutions instead of the more technical "number of full-time equivalents". Out of the respondents, 15 (9.0%) approximated between 0-999 undergraduates at their institutions. Out of these, 7 (46.7%) reported between 6-15 computer science majors per year, and 3 (20.0%) reported between 0-5 majors. Notably, 2 of them reported over 100 majors per year.

76 (45.8%) approximated their institution having between 1000-2999 undergraduates. Out of these, 35 (32.9%) approximated having

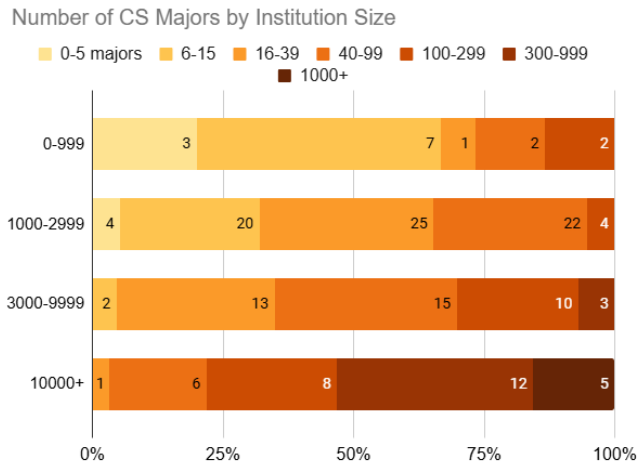

Figure 2: CS majors by institution size (number of responses, % axis).

between 16-39 majors, and 22 (29%) approximated having between 40-99 majors. 43 (25.9%) approximated having between 3000-9999 undergraduates at their institutions. Out of these, 15 (19.7%) approximated having between 40-99 majors and 16 (21.0%) approximated having between 16-39 majors. Finally, the remaining 32 (19.3%) approximated having over 10000 undergraduates. 12 (37.5%) of these approximated having between 300-999 majors, and 5 (15.6%) have over 1000 majors. Figure 2 summarizes, for each school-size (e.g. 0-999 students), the number of CS majors in the school, as estimated by respondents.

Limitations on survey respondents' time and our own resources made it unrealistic to determine precise enrollment data for each of the institutions represented in our survey. Thus our data on institution and major size represents respondents' best estimates and not official enrollments and major counts. Nevertheless we believe this data is sufficient to gain a picture of the size of the institutions and programs in our sample and to underscore the large number of students impacted by ToC curricula.

4.2 Theory of Computation in the Curriculum

For each topic in our list of "ToC topics", we asked respondents to identify the status of the course or courses in which they "primarily appear". Respondents were asked to specify one of the following five options:

- In a course required for at least one CS program.
- In a non-required (elective) course offered regularly.
- In a non-required (elective) course offered occasionally or sometimes.
- In a non-required (elective) course offered rarely or not at all.
- Unsure.

We did not attach numerical thresholds for terms like "occasionally" and "rarely", and instructed respondents to select the prompt that best described the situation at their institution in their opinion, even if the match was not exact. The complete response data is presented in Table 1.

The most common required topic on our list was "proof techniques:" 150 (90.4%) of institutions reported that they cover proof techniques in a course required for the CS major. No other topics were covered in required courses by more than 75% of institutions. Several topics were covered in a required course by at least 60% of institutions: these were finite automata, regular expressions, regular languages, context-free grammars, context-free languages, and complexity classes.

After introducing respondents to our topic list, we gave them the opportunity to list "closely related subtopics not included on the list above that you think should be included." We intended this question to elicit any closely related topics often taught in conjunction with the ToC topics on our list. Respondents offered the following supplemental topics (Chomsky Normal Form, Quantum Computing, and NP-Completeness were listed twice):

- Backus Normal Form
- Boehm-Jacopini theorem
- CKY algorithm
- Cantor's diagonalization [argument]
- Chomsky Normal Form (2)
- Closure properties
- Countable and uncountable sets
- DFA minimization
- Decision problems
- Deterministic PDA
- Extended CFG
- Formal languages
- Incompressible methods for proofs
- Kleene's recursion theorem
- Kolmogorov Complexity
- Lambda calculus
- Logic inference
- NP-Completeness (2)
- Myhill-Nerode [Theorem]
- P vs NP conjecture
- Quantum computing (2)
- RAM machines
- Russell's paradox
- Space complexity
- Space complexity hierarchy
- Time complexity hierarchy
- Traversal of maps, searching and complexity
- Turing completeness
- Universal Turing Machines
- Universal computation

We include all answers for completeness, although several of these topics appear to be closely related or equivalent to each other or topics on our original list.

4.3 Primary Theory of Computation Courses

In order to address our research questions concerning the administrative make-up of and pedagogical tools used in ToC courses (RQ2 and RQ3), we first needed respondents to identify a "ToC course" or courses. This presented a challenge: our chosen topics might appear

Table 1: Where theory of computation topics are covered in the curriculum

Topics	Required	Elective-regular	Elective-occasional	Rarely	Unsure/No answer
Proof techniques	150 (90.4%)	9 (5.4%)	2 (1.2%)	4 (2.4%)	1 (0.6%)
Topics in automata theory					
Deterministic/nondeterministic finite automata	111 (66.9%)	30 (18.1%)	10 (6.0%)	13 (7.8%)	2 (1.2%)
Pushdown automata (PDAs)	82 (49.4%)	37 (22.3%)	17 (10.2%)	27 (16.3%)	3 (1.8%)
Linear bounded automata (LBAs)	28 (16.9%)	15 (9.0%)	12 (7.2%)	92 (55.4%)	19 (11.4%)
Other automata	38 (22.9%)	21 (12.7%)	13 (7.8%)	70 (42.2%)	24 (14.5%)
Regular expressions	121 (72.9%)	27 (16.3%)	10 (6.0%)	5 (3.0%)	3 (1.8%)
Regular languages	111 (66.9%)	28 (16.9%)	14 (8.4%)	10 (6.0%)	3 (1.8%)
Proving that a language is not regular	84 (50.6%)	37 (22.3%)	16 (9.6%)	19 (11.4%)	10 (6.0%)
Context-free grammars	104 (62.7%)	29 (17.5%)	16 (9.6%)	14 (8.4%)	3 (1.8%)
Context-free languages	100 (60.2%)	29 (17.5%)	16 (9.6%)	16 (9.6%)	5 (3.0%)
Proving that a language is not context-free	70 (42.2%)	34 (20.5%)	18 (10.8%)	35 (21.1%)	9 (5.4%)
Context-sensitive languages	35 (21.1%)	19 (11.4%)	14 (8.4%)	80 (48.2%)	18 (10.8%)
Context-sensitive grammars	34 (20.5%)	18 (10.8%)	15 (9.0%)	81 (48.8%)	18 (10.8%)
Topics in computability theory					
(nondeterministic) Turing machines	93 (56.0%)	37 (22.3%)	13 (7.8%)	22 (13.3%)	1 (0.6%)
Recursive languages	89 (53.6%)	32 (19.3%)	11 (6.6%)	28 (16.9%)	6 (3.6%)
Recursively enumerable languages	76 (45.8%)	33 (19.9%)	13 (7.8%)	32 (19.3%)	12 (7.2%)
Decidability and undecidability	88 (53.0%)	35 (21.1%)	14 (8.4%)	25 (15.1%)	4 (2.4%)
Mapping reductions	67 (40.4%)	32 (19.3%)	10 (6.0%)	42 (25.3%)	15 (9.0%)
Rice's theorem	36 (21.7%)	28 (16.9%)	13 (7.8%)	66 (39.8%)	23 (13.9%)
Chomsky hierarchy	69 (41.6%)	25 (15.1%)	12 (7.2%)	44 (26.5%)	16 (9.6%)
Lambda calculus	24 (14.5%)	25 (15.1%)	20 (12.0%)	72 (43.4%)	25 (15.1%)
General recursive functions	58 (34.9%)	22 (13.3%)	8 (4.8%)	62 (37.3%)	16 (9.6%)
Church-Turing thesis	83 (50.0%)	35 (21.1%)	8 (4.8%)	31 (18.7%)	9 (5.4%)
Topics in basic complexity theory					
Turing machine time complexity	71 (42.8%)	36 (21.7%)	10 (6.0%)	38 (22.9%)	11 (6.6%)
Complexity classes (P, NP, EXP, etc.)	121 (72.9%)	24 (14.5%)	5 (3.0%)	13 (7.8%)	3 (1.8%)
Turing machine space complexity	38 (22.9%)	30 (18.1%)	16 (9.6%)	60 (36.1%)	22 (13.3%)
Polynomial-time reductions	91 (54.8%)	24 (14.5%)	4 (2.4%)	35 (21.1%)	12 (7.2%)
Cook-Levin theorem	51 (30.7%)	25 (15.1%)	11 (6.6%)	55 (33.1%)	24 (14.5%)

in one or many courses within a curriculum, in different orders and to different degrees. For each of our three topic categories (automata theory, computability theory, and basic complexity theory), we asked respondents:

> List the name and course number of the course at your institution in which *most students* who encounter [topic category] first encounter it *at significant depth*. [emphasis added]

We then asked follow-up questions about the courses that respondents identified. We intentionally left some ambiguity in the phrasing of this question (specifically, the qualifiers "most students" and "at significant depth") to allow the experienced faculty members responding to our survey to select the most appropriate course from multiple possibilities. Additionally, we prompted respondents to identify if students first encountered multiple topic categories in the same course (for example, if students first encountered many of the topics in our list in a single "Theory of Computation" course).

After respondents identified 1-3 "ToC courses" in which students first encountered each topic category in significant depth, we asked whether each topic on our list appeared in at least one listed "ToC course."

4.3.1 Automata Theory. 149 respondents (89.8%) identified at least one course offered at their institution in which topics in automata theory were covered. 17 responses (10.2%) did not identify such a course.

Figure 3 summarizes, for each topic in the automata theory category of our topic list, whether this topic was covered in at least one of the ToC courses previously identified by the respondent.[5]

[5]Because we did not differentiate between ToC courses for this question, this would include an automata theory topic covered in the course in which students first encountered complexity theory as "covered", even if this was not the same course in which students first encountered automata theory. We chose to group all 1-3 identified ToC courses together in this way because we did not expect distinctions between them to be consistent between respondents.

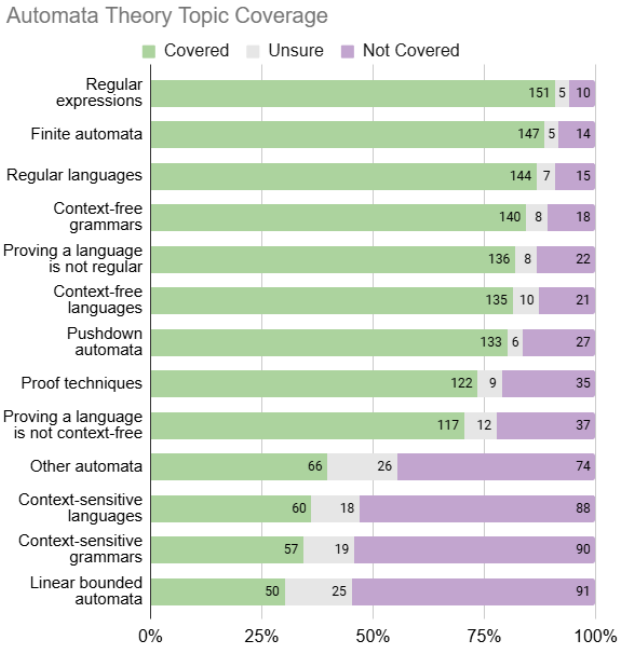

Figure 3: Topics in automata theory covered in at least one identified ToC course; unsure; not covered.

More than 87% of the responses indicated that regular expressions, finite automata, and regular languages (three closely related topics) were covered in at least one "ToC course". Similarly, more than 80% of responses indicated that context-free grammars, context-free languages, and pushdown automata were covered. Fewer than 40% of responses reported that topics related to other automata and language classes were covered in at least one "ToC course".

4.3.2 Computability Theory. 142 respondents (88.5%) identified at least one course offered at their institution in which topics in computability theory were covered. 24 responses (11.5%) did not.

Figure 4 summarizes the coverage of computability theory topics in identified ToC courses. The topics most frequently identified as covered in at least one ToC course were Turing machines (88.5%) and undecidability (80.7%). At least two-thirds of the responses also included recursive (71.1%) and recursively enumerable (67.5%) languages and the Church–Turing thesis (71.7%). Fewer than 40% of responses included other models of computation equivalent to Turing machines.

4.3.3 Basic Complexity Theory. 150 respondents (90.4%) identified at least one course offered at their institution in which topics in basic complexity theory were covered. 16 responses (9.6%) did not.

Figure 5 summarizes the coverage of basic complexity theory topics in one of the identified ToC courses. Complexity classes (84.3%) was the only topic that more than 75% of respondents identified as covered in one of the identified courses. At least 70% of responses also included polynomial-time reductions (71.1%) and

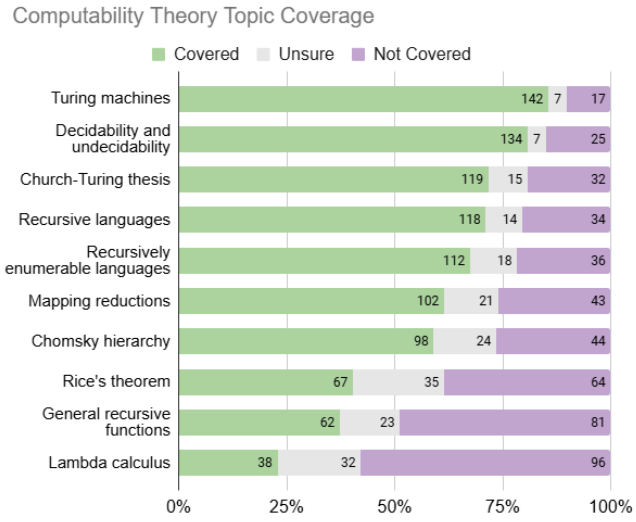

Figure 4: Topics in computability theory covered in at least one identified ToC course; unsure; not covered.

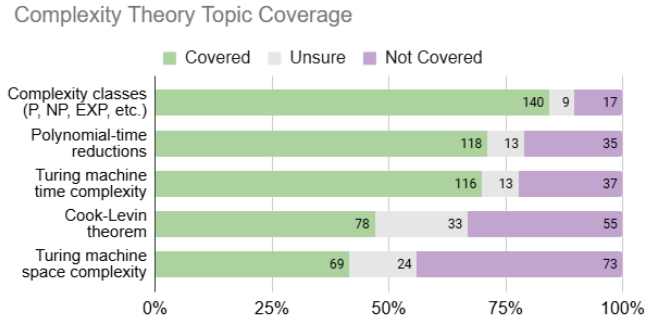

Figure 5: Topics in basic complexity theory covered in at least one identified ToC course; unsure; not covered.

Turing machine time complexity (69.9%). Both the Cook–Levin theorem (47.0%) and Turing machine space complexity (41.6%) were included in fewer than half of the responses.

4.4 Course Logistics

In the previous subsection, we summarized the primary ToC courses identified by respondents: those in which most students at their institution first encounter automata theory, computability theory and complexity theory in significant depth. Because the division between topic categories is somewhat arbitrary and our intention in this survey is to capture the diversity of ToC courses, in the following subsection we summarize data *per course*, counting multiple courses described by the same respondent separately in summary data.

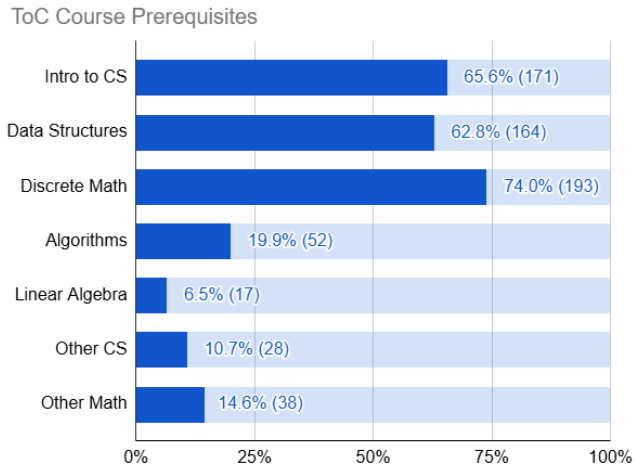

Figure 6: Prerequisite courses listed for all identified ToC courses.

4.4.1 Prerequisite Courses. Of the 261 ToC courses listed by respondents, we recorded the number that required each of the following course options as prerequisites:

- Discrete Mathematics or similar course (193, 74.0%)
- Introduction to Computer Science / Programming or similar course (171, 65.5%)
- Data Structures or similar course (164, 62.8%)
- Algorithm Design & Analysis or similar course (52, 20.0%)
- Linear Algebra or similar course (17, 6.5%)
- Other computer science course(s) (28, 10.7%)
- Other math course(s) (38, 14.6%)

The most common prerequisite course was Discrete Mathematics, which was required for nearly three quarters of all ToC courses listed by respondents. It was followed by Introduction to Computer Science / Programming and Data Structures, respectively. Other prerequisites were much less common. Respondents wrote in the following additional prerequisite courses:

- Discrete Structures (2)
- Computer Architecture
- Theory of Computation

4.4.2 Section Size. Respondents reported section sizes for 252 ToC courses (9 respondents did not indicate a section size for one or more previously mentioned courses.) The reported section sizes broke down as follows:

- 9 or fewer students (26, 10.3%)
- 10-19 (63, 25.0%)
- 20-29 (90, 35.7%)
- 30-59 (46, 18.2%)
- 60-119 (18, 7.1%)
- 120-249 (7, 2.8%)
- 250 or more (2, 0.8%)

The majority of responses (60.7%) indicated section sizes between 10-29 students, with a plurality of responses indicating 20-29 students. Most (89.3%) classes had under 60 students in total.

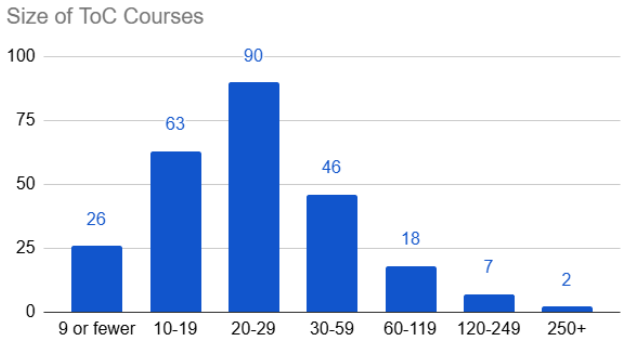

Figure 7: Section size listed for all identified ToC Courses.

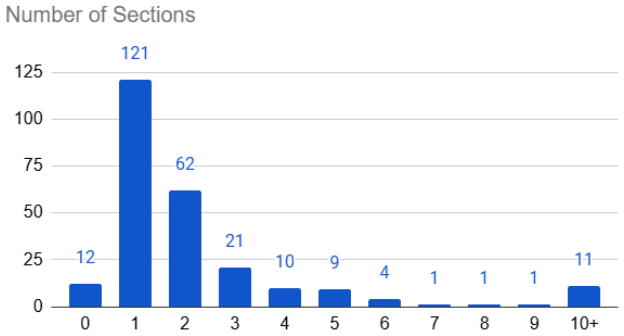

Figure 8: Approximate number of sections offered annually for all identified ToC Courses.

Almost half of responses (121, 47.8%) indicated that their institution offered approximately one section of a certain ToC course per year. Significant minorities indicated approximately two (62, 25.4%) and approximately three (21, 8.3%) sections per year. Figure 8 summarizes the number of sections offered.

4.4.3 Format of Instruction. The vast majority of the instruction in the theory courses we surveyed was done via lectures, with a significant portion of those also including a problem solving session. We allowed respondents to select multiple options.

Respondents indicated that 227 courses out of 253 for which we recieved answers (89.7%) were taught in the lecture format (either as a "Lecture" or a "Lecture with Problem Session", or both). Figure 9 summarizes the number of sections offered for each type of course format.

4.4.4 Use of Student Helpers . Of the ToC courses listed by respondents, only about a third of the sections employed student helpers (teaching assistants, instructional assistants, or other student workers in similar roles). Figure 10 summarizes the number of TAs per section of ToC courses reported by respondents.

4.5 Instructor Demographics

In addition to data regarding specific ToC courses, we also asked respondents to answer several questions regarding the instructors

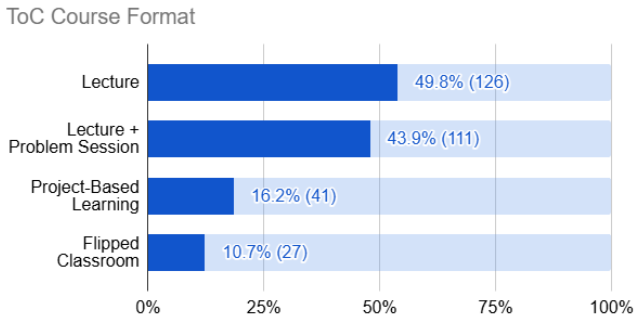

Figure 9: Course format of the ToC courses reported by respondents. Respondents could select multiple options for a single course.

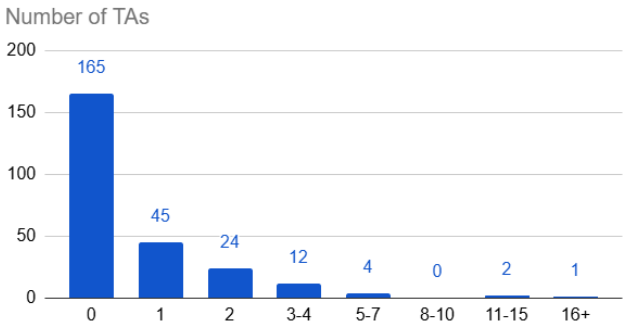

Figure 10: Number of TAs per section in the ToC courses reported by respondents.

teaching ToC courses. Because different instructors might teach different ToC courses during different periods of time, we made no attempt to link these questions to the specific ToC courses considered in previous questions. Instead, we asked less specifically about the (approximate) fraction of theory-focused courses taught by faculty members of different types.

We first asked respondents about the professional roles of instructors teaching ToC courses (Figure 11).[6] The most common instructor category was permanent teaching-focused CS faculty: 88 respondents (63.3% of respondents who answered this subquestion, 53.0% of total) reported that 'most' or 'all or nearly all' of their ToC classes are taught by faculty members in this category. 51 respondents (42.5% of answered, 30.7% of total) reported that 'most' or 'all or nearly all' of their ToC classes are taught by permanent research-focused CS faculty. (Some respondents did not interpret these responses as mutually exclusive: 21 (12.7% of total) fell into *both* categories.) In contrast, only 11 respondents reported that 'most' or 'all or nearly all' of their ToC classes were taught by instructors in *any* of the 'Adjunct CS faculty', 'Non-CS faculty', or 'Other' categories. The strong bias towards permanent members

[6]Some ambiguity was introduced by the fact that different numbers of respondents chose to leave each subquestion blank; this might indicate a response similar to "Few or none", "Unsure", or something else. We provide percentages calculated with and without those who chose to skip the subquestion.

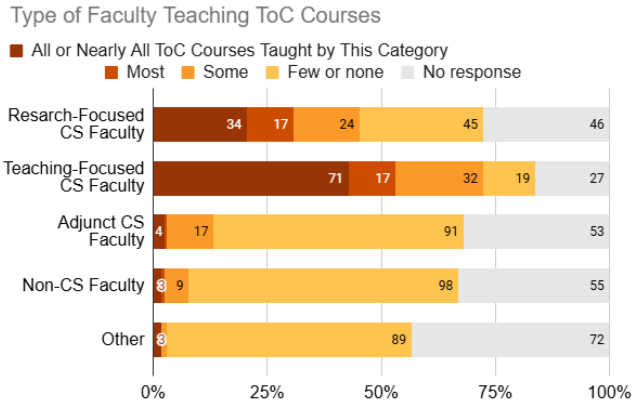

Figure 11: Number of institutions at which "all or nearly all", "most", "some", or "few or none" ToC courses are taught by faculty in different roles.

of a CS faculty may reflect the pool of instructors at institutions that teach ToC (and perhaps have large, well-established CS departments) rather than a trend specific to ToC courses. Still, it is somewhat surprising to see so few adjunct faculty represented in our sample.

Moreover, a significant number of institutions in our sample reported that ToC courses were taught by faculty members with a research focus in ToC or algorithm design. 39 respondents (32.0% of answered, 22.9% of total) reported that "all or nearly all" ToC courses at their institution were taught by faculty members with a primary research interest in ToC or algorithm design. A similar number (38 respondents, 19.2% of answered, 23.5% of total) reported that "all or nearly all" ToC courses at their institution were taught by faculty members with a primary research interest in another area of CS. Finally, a substantial portion of respondents reported some instructors in both categories: 61 respondents (36.7% of total) reported that at their institution at least 'some' ToC instructors' research focuses on ToC or algorithm design *and* at least 'some' ToC instructors' research focuses on another area of CS, reflecting institutions at which a mix of theorists and non-theorists teach ToC courses.

4.6 Tool Use in ToC courses

In a final survey section we asked respondents which digital tools they or their colleagues use when teaching ToC courses at their institution. We asked questions about two tool categories: (a) clickers and polling platforms and (b) ToC-specific digital tools (such as tools for drawing finite state automata).

The following digital tools were reported as used in at least one theory-focused course by at least one respondent:

- Clickers (8)
- Kahoot! (7)
- Poll Everywhere (6)
- Mentimeter (3)
- Socrative (3)

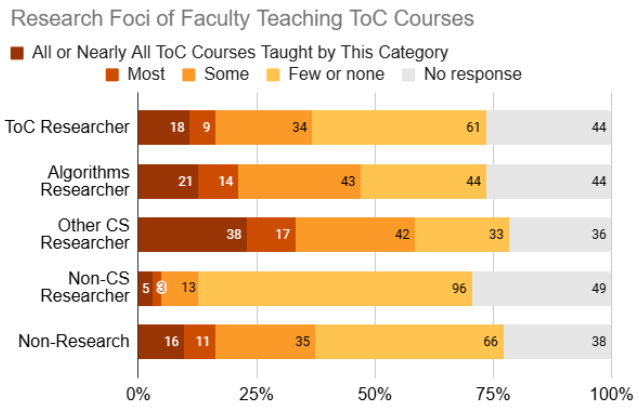

Figure 12: Number of institutions at which "all or nearly all", "most", "some", or "few or none" ToC courses are taught by faculty with different research foci.

In short, very few respondents reported using the polling methods we included in the survey.

A significant minority of respondents (49 participants, 29.5%) reported using at least one of the ToC-specific digital tools we prompted. At least one respondent reported using the following ToC-specific tools in at least one theory-focused course at their institution:

- JFLAP (40)
- FSM Designer (9)
- AutomataTutor (4)
- State Machine Cat (3)

With the exception of the popular software package JFLAP, the ToC-specific tools we polled were used in very few classrooms.

5 Discussion and Discoveries

In this section, we summarize insights gained from and trends observed in our data. In keeping with the exploratory nature of our survey, we emphasize that our conclusions are preliminary: we aim to establish a variety of plausible hypotheses rather than to provide final answers.

5.1 The Theory of Computation "Core"

The most universal theory topic (perhaps a prerequisite topic) in our topic list was "proof techniques", which over 90% of respondents listed as present in a required course. While "proof techniques" was unique in its ubiquity, respondents listed several other topics in our topic list as both commonly required in CS curricula and usually present in the primary ToC courses they identified. By topic category, these were the following:

5.1.1 Core Automata Theory Topics. Topics in the automata category fall into three rough groups: those present in the vast majority of curricula, those present in most curricula, and those present in a significant minority of curricula.

There were five automata topics that had at least 60% of respondents list as "required":

(1) Regular expressions (72.9%)
(2) Deterministic/nondeterministic Finite Automata (66.9%)
(3) Regular languages (66.9%)
(4) Context-free grammars (62.7%)
(5) Context-free languages (60.2%)

If we add the number of respondents who said that a regularly offered non-required (elective) course covered each of these topics, we can safely conclude that at least 80% of our respondents regularly cover these topics in their curriculum (Table 1). This aligns with the subsequent finding that more than 80% of respondents reported that at least one identified ToC course covers these topics (Figure 3). Around half of respondents listed an additional three topics as "required":

(1) Proving that a language is not regular (50.6%)
(2) Pushdown automata (49.4%)
(3) Proving that a language is not context-free (42.2%)

For each of these topics, a majority of respondents indicated that they offer it regularly in the curriculum. These eight topics are exactly those concerned specifically with (1) regular languages and (2) context-free languages. The remainder of topics in the automata category are present in the curricula of many institutions, but only a distinct minority of respondents offer them regularly.

A significant fraction of courses in which students first encountered automata theory were courses concerned with discrete mathematics, programming languages or compilers (see Section 5.3.1). We speculate that these courses might be less likely to cover nonregularity and pushdown automata.

5.1.2 Core Computability Theory Topics. None of our listed computability theory topics had more than 60% of respondents indicate that their curriculum requires the topic. However, several are commonly required:

(1) Deterministic or Nondeterministic Turing Machines (56.0%)
(2) Recursive Languages (53.6%)
(3) Decidability and Undecidability (53.0%)
(4) Church-Turing Thesis (50.0%)
(5) Recursively Enumerable Languages (45.8%)
(6) Chomsky hierarchy (41.6%)
(7) Mapping reductions (41.6%)

Including regularly-offered elective courses, each of these topics appeared regularly in the curricula of institutions represented by between 56.6% and 78.3% of 166 respondents, a clear majority. No other topic in the computability category appeared regularly in the curricula for more than half of respondents, or appeared in a named ToC course for more than half of respondents.

Our data on computability theory coverage aligns with several observations due to Blumenthal [5]. Michael Sipser's *Introduction to the Theory of Computation* is the most popular textbook used in ToC classrooms [5], and it relegates Rice's Theorem to an exercise [28]. This might help to explain why Rice's theorem appeared regularly in the curriculum of only 64 institutions in our sample (38.6%). Our results also agree with Blumenthal's observation [5] that despite covering the Church–Turing thesis, ToC courses tend not to cover any models that are equivalent in power to Turing machines. We asked about two such models (general recursive functions and

lambda calculus) and neither was covered by greater than 40% of respondents.

5.1.3 Core Complexity Theory Topics. The majority of institutions (72.9%) reported covering complexity classes (P, NP, EXP, etc.) as a part of a required course. This was the only listed topic outside of the automata theory category to be required to a similar extent as the five most often required automata theory topics. Most institutions also reported covering polynomial-time reductions and Turing machine time complexity in at least some regularly offered required or elective course.

As discussed in Section 5.3, many respondents expect their students to encounter basic complexity theory in an algorithms course. This may explain the relative lack of coverage of the Cook–Levin Theorem despite coverage of complexity classes and polynomial-time reductions.[7]

5.2 The "Typical" Theory Course

For reasons previously discussed (see Section 3.4), we cannot assume our respondents represent "typical" CS programs. However, we can summarize the common characteristics of the ToC courses described by our respondents, which may be more representative of those institutions specifically offering ToC. Within our sample, a "typical ToC course":

(1) requires a course in Discrete Mathematics, and is more likely than not to require Introduction to Computer Science or Programming and Data Structures courses as well.
(2) enrolls between 10 and 29 students.
(3) is taught primarily via lecture, perhaps with sessions devoted to group problem-solving.
(4) is slightly more likely to be taught from the whiteboard or blackboard than via a slide deck, although both are common.
(5) is taught by a permanent faculty member primarily focused on teaching or research. This faculty member is very likely to be a computer scientist, and is roughly equally likely to conduct research in either Algorithms or ToC, or in another area of computer science.
(6) employs 2 or fewer teaching assistants, and may have none.
(7) is unlikely to use digital tools.

The above description is intended as a high-level summary to provide intuition about our results. It generalizes from plurality answers, ignores correlations between answers and may not describe any particular course. Nevertheless, it conveys the general trends present in our data.

5.3 Course Naming Trends

In general, our survey focused on quantifiable metrics and did not collect qualitative data about the ToC courses in our sample. However, we did collect the *names* of ToC courses. While slight differences in the names of courses may not allow us to infer any specific differences in course content, *major* differences in course title (e.g., "Theory of Computation" vs. "Algorithm Design") indicate significant differences in the focus of a course.

5.3.1 Names of Courses Covering Automata Theory Topics. Of the 149 responses that identified a course in which students encountered topics in automata theory, about 66 responses (44.2%) contained the phrase "theory of computation" or "theory of computing" in the course title. This is in contrast with "theoretical computer science", which appears in only four course titles, and "computability", which appears in only six course titles. This points to an interesting distinction between the three phrases.

The next most frequent term in course titles is "Language" (34 responses, 22.8%). There are roughly two kinds of courses that contain the term "Language". The first are traditional ToC courses, with titles like "Languages and Automata" or "Introduction to Formal Languages". There were 12 responses (8.0%) that identified courses with the term "formal languages" in the title and 22 responses 14.8%), the most frequent term, identified courses with the term "automata" in the course title. Note that these counts are not exclusive—some courses contain both, e.g. "Formal Languages and Automata Theory".

The second group of courses are not traditional ToC courses, but are courses on programming languages. There were 13 responses (8.7%) that identified courses that contain "Programming Languages" in the title. These are typically "programming language theory" courses, though there are also 5 courses (3.3%) that contain the term "compiler". This means approximately one tenth of responses identify a programming language theory or implementation course as the primary course through which their students encounter automata theory.

One more small group of respondents (5 responses, 3.3%) identified their courses contain the term "discrete"–these are discrete math courses. 3 responses identified courses that contained the term "mathematical foundations", though it is unclear whether these courses are discrete math courses or ToC courses.

5.3.2 Names of Courses Covering Computability Theory Topics. For the primary course on computability topics, most responses identified the same course as the primary course on automata theory. Of the 32 responses that identified a different course, most contained the phrase "Theory of Computing", "Computability", or "Complexity". We note that some of the identified course titles and numbers were repeated from the question identifying the primary automata theory course.

5.3.3 Names of Courses Covering Basic Complexity Theory Topics. 69 respondents indicated that the same course primarily covering topics in the basic complexity theory category was the same as that covering at least one of automata theory or computability theory. However, approximately one-third of responses (57 responses, 34.3%) identified a course with a title that contains the term "algorithm" as the course where students encounter topics in basic complexity theory. We suspect that many of these courses are focused on the design and analysis of algorithms and centered on the "Algorithmic Strategies" knowledge unit of the 2023 ACM/IEEE-CS/AAAI Curriculum [19]. This seems to agree with the survey

[7] Recall that the Cook–Levin Theorem states that the Boolean satisfiability problem is NP-complete, and provides the "first" NP-complete problem to be used in subsequent reductions. The proof of the Cook–Levin theorem proceeds by construction of nondeterministic Turing machines. While this proof is often presented in courses dedicated to complexity theory, in an algorithms course, it is not likely that Turing machines are covered. As a result, in many algorithms texts [7, 17], the "first" NP-complete problem is presented as the circuit satisfiability problem rather than satisfiability, avoiding the need to introduce Turing machines.

by Luu el al. [20], which found that about half of identified algorithms courses cover basic complexity topics, like P vs. NP and NP-completeness.

Across all responses for the three identified courses, very few course titles contained the phrase "Complexity". This suggests that, if students experience complexity theory in any significant detail in their CS curriculum, they are unlikely to first encounter it in a course devoted specifically to complexity theory.

5.4 Use of Digital Teaching Tools is Rare

Curiosity about the use of digital tools in the classroom was a a significant motive for the design of our survey. As educators in theory of computation, we were curious to learn what tools, especially ToC-specific tools, were in use by our peers. However, our data suggests that few ToC classes use digital tools. About a quarter of respondents reported using general-purpose classroom tools, such as clickers and autograders, and about a quarter of respondents reported using ToC-specific tools. The landscape of ToC-specific tools was fragmented: there is only one tool (JFLAP) which more than 10% of survey respondents use. These negative results suggest natural follow-up questions, including why most instructors do not use digital tools, whether there is a perceived need for such tools, and, if so, what specific features could be useful.

6 Open Questions & Future Work

Given the relative lack of computing education research on theory of computation, we conceived this project as a preliminary survey. We have attempted to provide productive insights, generate hypotheses to be tested, and emphasize gaps in our understanding for future research to fill. As such, we close with a short list of open questions:

(1) **Open Question 1:** What are the demographics of institutions of higher education that offer theory of computation (or, more broadly, theoretical CS), and how do these institutions compare to other institutions teaching CS?
(2) **Open Question 2:** In this survey, we omitted several important subtopics in introductory CS, including formal logic and programming language semantics. We also omitted advanced topics in algorithms and complexity theory. What is the role of other theoretical CS topics in the curriculum?
(3) **Open Question 3:** This survey indicates that relatively few ToC courses use ToC-specific tools, and many of those that do use in-house tools developed specifically for the course. It is possible that ToC courses are small because of a lack of available tools, or vice versa. What is the cause and effect?

Other adjacent topics for future research include opinions of students and faculty about theoretical CS, student performance in ToC courses of different types, and meta-analysis of the body of ToC-focused computing education research, among many others.

Data Availability

The data in this manuscript represents survey responses submitted by 11 November 2024. In subsequent weeks we received additional responses that expanded our data set slightly but did not materially change the conclusions presented here.

A copy of the full survey data with identifying information removed can be freely accessed here: https://github.com/twrand/toc-survey-2025 .

Acknowledgments

The authors would like to thank Jacqueline Whalley and Juho Leinonen for their editorial guidance and helpful feedback on an earlier draft of this manuscript. We would also like to thank several reviewers for their deep engagement with and thoughtful feedback on our work, which greatly improved the final product even though some suggestions could not be implemented due to time limitations.

The opinions in this work are solely of the authors, and do not necessarily reflect those of the U.S. Army, U.S. Army Research Labs, the U.S. Military Academy, or the Department of Defense.

Thomas Zeume is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 448468041.

References

[1] Rajeev Alur, Loris D'Antoni, Sumit Gulwani, Dileep Kini, and Mahesh Viswanathan. 2013. Automated Grading of DFA Constructions. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, Francesca Rossi (Ed.). IJCAI/AAAI, 1976–1982. http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6759

[2] Lillian Baker, Sierra Zoe Bennett-Manke, Sebastian Neumann, Ian Njuguna, and Ryan E. Dougherty. 2025. TheoryViz: A Visualizer Tool for Theory of Computing Concepts. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 1373–1374. https://doi.org/10.1145/3641555.3705268

[3] Sierra Zoe Bennett-Manke, Sebastian Neumann, and Ryan E. Dougherty. 2024. Finite State Machine with Input and Process Render. In *Proceedings of the 2024 on ACM Virtual Global Computing Education Conference V. 2* (Virtual Event, NC, USA) *(SIGCSE Virtual 2024)*. Association for Computing Machinery, New York, NY, USA, 299–300. https://doi.org/10.1145/3649409.3691079

[4] Dave Berque, David K Johnson, and Larry Jovanovic. 2001. Teaching theory of computation using pen-based computers and an electronic whiteboard. *ACM SIGCSE Bulletin* 33, 3 (2001), 169–172.

[5] Richard Blumenthal. 2022. Teach More, Not Less Computability Theory in CS202X: A Case for Teaching Multiple Representations of the Church-Turing Thesis. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 675–681. https://doi.org/10.1145/3478431.3499309

[6] Lillian Cassel, Alan Clements, Gordon Davies, Mark Guzdial, Renée McCauley, Andrew McGettrick, Bob Sloan, Larry Snyder, Paul Tymann, and Bruce W. Weide. 2008. *Computer Science Curriculum 2008: An Interim Revision of CS 2001*. Technical Report. New York, NY, USA.

[7] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.

[8] Rafael del Vado Vírseda. 2021. Learning from the Impossible: Introducing Theoretical Computer Science in CS Mathematics Courses. 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, 952–958. https://doi.org/10.1145/3408877.3432475

[9] Rafael del Vado Vírseda. 2023. Theoretical Computer Science Education from Impossibility and Undecidability Problems in Physics. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 270–276. https://doi.org/10.1145/3545945.3569742

[10] Rafael del Vado Vírseda. 2024. Introducing Theoretical Computer Science Education in Social Sciences and Economics Degrees. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1620–1621. https://doi.org/10.1145/3626253.3635481

[11] Ryan E. Dougherty. 2024. Designing Theory of Computing Backwards. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1628–1629. https://doi.org/10.1145/3626253.3635479

[12] Ryan E. Dougherty. 2024. Experiences Using Research Processes in an Undergraduate Theory of Computing Course. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE*

2024). Association for Computing Machinery, New York, NY, USA, 310–316. https://doi.org/10.1145/3626252.3630849

[13] Ryan E. Dougherty. 2025. Scaffolding Mock Conference Projects in Theory of Computing Courses. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 1441–1442. https://doi.org/10.1145/3641555.3705243

[14] J. Philip East. 2006. On Models of and for Teaching: Toward Theory-Based Computing Education. In *Proceedings of the Second International Workshop on Computing Education Research* (Canterbury, United Kingdom) *(ICER '06)*. Association for Computing Machinery, New York, NY, USA, 41–50. https://doi.org/10.1145/1151588.1151596

[15] Jeff Erickson, Jason Xia, Eliot Wong Robson, Tue Do, Aidan Tzur Glickman, Zhuofan Jia, Eric Jin, Jiwon Lee, Patrick Lin, Steven Pan, et al. 2023. Auto-graded scaffolding exercises for theoretical computer science. In *ASEE Annual Conference and Exposition, Conference Proceedings*.

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

[17] Jon Kleinberg and Eva Tardos. 2005. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., USA.

[18] Joan Krone. 1992. Student Designed Machines for a Theory of Computation Course. *SIGCSE Bull.* 24, 3 (sep 1992), 51–52. https://doi.org/10.1145/142040.142075

[19] Amruth N. Kumar, Rajendra K. Raj, Sherif G. Aly, Monica D. Anderson, Brett A. Becker, Richard L. Blumenthal, Eric Eaton, Susan L. Epstein, Michael Goldweber, Pankaj Jalote, Douglas Lea, Michael Oudshoorn, Marcelo Pias, Susan Reiser, Christian Servin, Rahul Simha, Titus Winters, and Qiao Xiang. 2024. *Computer Science Curricula 2023*. Association for Computing Machinery, New York, NY, USA.

[20] 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 US. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 284–290.

[21] Mostafa Mohammed, Clifford A. Shaffer, and Susan H. Rodger. 2021. Teaching Formal Languages with Visualizations and Auto-Graded Exercises. 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, 569–575. https://doi.org/10.1145/3408877.3432398

[22] Marco T Morazán. 2023. *Programming-Based Formal Languages and Automata Theory: Design, Implement, Validate, and Prove*. Springer Nature.

[23] Nelishia Pillay. 2010. Learning Difficulties Experienced by Students in a Course on Formal Languages and Automata Theory. *SIGCSE Bull.* 41, 4 (jan 2010), 48–52. https://doi.org/10.1145/1709424.1709444

[24] Tim Randolph. 2024. Participatory Governance in the Computer Science Theory Classroom. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 1091–1097.

[25] Eliot Wong Robson, Sam Ruggerio, and Jeff Erickson. 2024. FSM Builder: A Tool for Writing Autograded Finite Automata Questions. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 269–275. https://doi.org/10.1145/3649217.3653599

[26] Marko Schmellenkamp, Fabian Vehlken, and Thomas Zeume. 2024. Teaching Formal Foundations of Computer Science with Iltis. *Bulletin of EATCS* 142, 1 (2024).

[27] Scott Sigman. 2007. Engaging Students in Formal Language Theory and Theory of Computation. 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, 450–453. https://doi.org/10.1145/1227310.1227463

[28] Michael Sipser. 2013. *Introduction to the Theory of Computation* (3 ed.). Course Technology.