Earning a CS Minor: A Not-So-Minor Feat. A Survey of Accessibility and Structure of 120 Computer Science Minors

Albert Lionelle
Khoury College of Computer Sciences
Northeastern University
Boston, MA, USA
a.lionelle@northeastern.edu

Anya Amin
Center for Inclusive Computing
Northeastern University
Boston, MA, USA
a.amin@northeastern.edu

Megan Giordano
Center for Inclusive Computing
Northeastern University
Boston, MA, USA
m.giordano@northeastern.edu

Catherine Gill
Center for Inclusive Computing
Northeastern University
Boston, MA, USA
c.gill@northeastern.edu

Abstract

As computer science education continues to evolve, minors in Computer Science represent an important but understudied pathway for students to develop computational thinking skills and interdisciplinary expertise. This study examines 104 Computer Science minor programs from the top 120 universities by graduation rates to understand their current structure, requirements, and accessibility.

Our analysis reveals significant challenges in Computer Science minor accessibility; with the actual credits to obtain a minor being significantly higher than advertised credits, imposed entry requirements, hidden requirements, unrealistic math requirements, and the majority of minors requiring more than two years to complete. An analysis of open elective credits for science majors (Biology, Chemistry, Math, Physics, Psychology) shows that most cannot complete a Computer Science minor without exceeding standard degree requirements, despite these fields increasingly requiring computational skills.

In addition to the minor in Computer Science, we present information about other minors offered by the 120 Computer Science departments to better understand the landscape of minors offered by Computer Science departments overall, including interdisciplinary minors such as Bioinformatics or Human Computer Interaction.

We provide recommendations for improving minor accessibility including: reducing prerequisite complexity, eliminating hidden requirements, reconsidering entry barriers, and developing interdisciplinary pathways. These findings call for a systematic reevaluation of Computer Science minors to better serve students seeking computational literacy across disciplines.

CCS Concepts

• **Social and professional topics → Computing education**; **Computing literacy**; **Model curricula**.

This work is licensed under a Creative Commons Attribution 4.0 International License.
SIGCSE TS 2026, St. Louis, MO, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2256-1/26/02
https://doi.org/10.1145/3770762.3772553

Keywords

Computing education, Curriculum design, BPC, Computing Minors

ACM Reference Format:
Albert Lionelle, Anya Amin, Megan Giordano, and Catherine Gill. 2026. Earning a CS Minor: A Not-So-Minor Feat. A Survey of Accessibility and Structure of 120 Computer Science Minors. In *Proceedings of the 57th ACM Technical Symposium on Computer Science Education V.1 (SIGCSE TS 2026), February 18–21, 2026, St. Louis, MO, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770762.3772553

1 Introduction

University educators know that students benefit greatly from being exposed to diverse perspectives and building interdisciplinary skill sets. One way for students to build such skills is by minoring in a field outside their chosen major.

Most research on Computer Science (CS) degree programs is focused on the structure of the undergraduate major, and there is little research on the CS *minor*. Anecdotally, we hear that many CS minor degree plans haven't been updated in years, and that these programs have low enrollments. Nonetheless, this does not mean that the minor should continue to be largely ignored.

Indeed, we assert that minors are both an important route into the major and a key way for students to build their interdisciplinary skills. Some institutions insist on the value of the minor; for example, liberal arts institutions often require their students to pursue both a major and a minor [31]. However, given the lack of research on the CS minor, we know little about student experiences and outcomes and therefore lack insight into how universities might enhance their CS minors to align better with students' needs and interests.

The work in this paper seeks to fill some of this void, in particular by investigating the feasibility of the CS minor. We set out to understand whether the CS minor is an accessible degree and whether there may be structural barriers that are limiting the minor's popularity. To investigate this question, we examined the CS minors at the 120 schools with the highest completion of computing majors (as defined by the Integrated Postsecondary Education Data System (IPEDS) [1] categories CIP 11.01 and 11.07). Using public facing websites, we then built program maps for 104 of the CS minors at these 120 institutions (the remaining 16 either did not have a CS

[1] https://nces.ed.gov/ipeds/

minor, or the publicly available data was incomplete). A program map is similar to a degree map in that it lays out the courses a student needs to complete, on a semester-per-semester basis, in order to achieve the degree. To assess the accessibility of achieving the CS minor at each institution, we then compared the advertised number of credits against: 1) the actual number of credits needed to earn the minor per the program map (i.e., to assess the discrepancy between what an institution says and what is actually true); 2) the typical number of credits for a minor at the institution (i.e., to see how it compares to a schools' overarching approach to minors); and 3) the open elective credits for other science majors (i.e., to gauge the accessibility of the minor to STEM majors).

We note that, at the 120 institutions, our analysis included tracking minors that are related to, but not necessarily titled Computer Science - such as minors in Data Science, Bioinformatics, and Software Engineering. This allowed us to see the minors offered by Computer Science departments overall, including interdisciplinary minors and those in specialized fields.

Overall, minors are a powerful tool for students to explore Computer Science and gain a foundation in computational thinking. This research examines whether minor pathways are viable for students and whether accessibility barriers exist - problems that Computer Science departments could set out to address. As such, based on the analysis, we present recommendations for institutions on how to improve their CS minor(s) along with suggestions for future work within the field.

In Section 2, we first review existing research on minors and define curricular complexity measures used in our analysis. In Section 3, we describe our experimental methodology, how we built our data set, and the assumptions used when developing the 104 minor plans. In Section 4 we present our results, including threats to their validity. In Section 5 and Section 6, we discuss the takeaways for computing departments and our overarching conclusions, respectively.

2 Related Work

There is a long history in the computer science education research community of studying and revising curricular guidelines. In addition to CS education researchers, there are two professional organizations, ABET-CAC [12] and ACM [3] that provide curricular recommendations and guidelines. However, neither ABET-CAC nor ACM-2023 [28] discuss in depth the curricular structure of the degree requirements nor do they detail requirements for a minor.

The SIGCSE Committee on Computing Education in Liberal Arts Colleges emphasizes the necessity for flexibility in curricular guidelines [23, 39]. The committee underscores that liberal arts institutions, which encourage students to engage with diverse perspectives and integrate skills across multiple disciplines, cannot rely on a singular pathway toward Computer Science degree completion. They advocate for adaptable, condensed curricular structures featuring thematic pathways. Interdisciplinary initiatives prove fundamental, manifesting either through interdisciplinary degrees or minors. It is worth noting that a recent study demonstrated the benefits of flexible and shorter curricular pathways along with thematic pathways at a more typical R1 institution. Ganesan et al. [16] reported retention rates increasing from 67% to 98% along with improvements in attracting students to their program by reducing curricular complexity.

The focus on interdisciplinary programs is not limited to liberal arts colleges. Indeed, some universities have taken the lead in how they visualize CS for the current generation of students [29], and it was proposed as far back as 1976 that CS should have interdisciplinary studies [2]. Much of the work has been focused on interdisciplinary introductory Computer Science courses [4, 5, 13, 14, 18, 32, 40]. Northeastern University (NU) has 49 combined majors across 29 distinct interdisciplinary fields. As of Fall 2021, 44.6% of majors were combined majors, and 46% of combined major students identified as women [8]. NU also offers a "Meaningful Minor" in Computer Science which allows the fifth (and final) required course to be an additional upper level computing elective or an interdisciplinary course from a "pick list" [1]. Computer Science at the University of Illinois Urbana-Champaign has a long history of interdisciplinary programs. Mathematics & Computer Science was established in 1968, Statistics and Computer Science in 1988, and an additional 15 blended programs were introduced in 2013. For these two schools and others who introduce what are frequently termed 'CS + X', interdisciplinary programs often blend a strong CS core with core requirements from another discipline program [6].

Although there hasn't been much research on minors in Computer Science outside of an initial proposal in 1979 for an interdisciplinary minor that combines other STEM programs [30], there is evidence that interdisciplinary minors are needed and can be a means to improve representation. Kuri et al. present a Bioinformatics minor with 60% of the students enrolled in the Bioinformatics minor identifying as women [26, 27].

While it is well known that there is an increase in Computer Science majors since 2017, it is also reported that there is an overall increase in the demand for minors in both Computer Science and Data Science [34, 35]. Liberal arts institutions particularly embrace minors. Allegheny College requires all students to complete both a major and a minor, creating 999+ combinations [31]. This approach addresses the career patterns of Gen Z students, who are expected to be the most educated generation, but will frequently change jobs and fields, requiring continuous skill development [36]. Minors offer students the opportunity to build interdisciplinary skills, strengthen adaptability, and foster analytical and critical thinking. Importantly, Stock identified that 54% of students choose a minor in their second year and 30% in their third year [38], indicating that minors should be designed to ensure completion is possible in two years.

The growing field of Curricular Analytics [19–22, 37] focuses on the structure of curricular design and how that design can create unintended systemic barriers. Recent work in Computer Science [15, 16, 25, 33] shows that curricular complexity can be a tool to compare curricular designs. While Lionelle et al. and Jiang et al. demonstrate curricular complexity can be a barrier for women and transfer students respectively, Ganesan et al. uses curricular complexity as a means to compare program changes pre and post curricular revamp [16]. Jarratt at al. look at structural complexity and find that majors that are more homogeneous, those with "locked" requirements behind longer curricular chains, are more likely to lose students as they progress over the years ("an out curriculum") while heterogeneous majors (more flexible) often show an intake of students throughout the years ("an up curriculum").

STEM programs tend to be more aligned with "out" majors, losing more students to other majors [24]. Grosz et al., looked at community college certificates and found that certificates with more flexibility show higher enrollments and success [17].

Heileman et al. [20] defines four measures to help compare curricular designs which we use in this paper to compare minor curricular designs. **Delay Factor** measures the length of a prerequisite chain. This allows for comparison of the minimum length of programs. **Blocking Factor** measures the number of future courses that a course blocks by being a prerequisite. This is an indicator of how students' progress in a degree is blocked if they need to retake a course. **Structural Complexity** is a linear combination of blocking and delay factor, assigning a complexity score to each course. The sum of all course complexities is the program structure complexity. **Centrality** measures the total number of prerequisite chains that go through the course. This indicates how central a course is to the program, whether intended or not. Looking at centrality, it is possible to see bottlenecks that have a large influence on one or more pathways in the program.

3 Methodology

Using the Integrated Postsecondary Education Data System (IPEDS) for 2021-2022, we pulled a list of all Computer Science Programs (CIP Codes: 11.01 and 11.07) that produced over 100 graduates in that academic year. We then took the top 120 programs based on the number of degree completions. From this list, researchers manually evaluated public facing websites and minor degree plans. If the minor requirements weren't listed publicly, or there were discrepancies in the information found (such as conflicting requirements listed), or the website was too difficult to navigate (such as requirements not being listed), it was removed from the dataset. Curricular maps were generated for minors in the remaining 104 programs.

3.1 Degree Curricular Maps

Using the information for each university, we built a degree plan for a sample student with the following assumed characteristics:

- Student is evaluating if they should minor in Computer Science and has no prior computing experience.
- Student is "Calculus ready".
- Student chooses the shortest path towards completion with the least number of prerequisites. (Many plans would have options for students to pick upper division courses from a list, so in these cases we would select the courses that required the fewest prerequisites.)

While there are universities that say that close to 50% of their students are not calculus ready [16], the calc-ready assumption ensured we had the "quickest path towards completion" for the minor (we discuss this in the section on threats to validity). Every sample plan was then uploaded to Curricular Analytics[2] to generate *structural complexity*, *course centrality*, and *delay factor*.

3.2 Credit Counts and Normalization

For each plan, we collected *Advertised Credits* (as found on the institution's website), and the *Actual Credits*, namely the number of credits needed to actually complete the minor, including hidden requirements. For example, the list of minor requirements might include Discrete Math but leave out Calculus even if Calculus is a prerequisite for Discrete Math. Fifty-five (52.88%) of the plans had hidden requirements.

We also searched each university's documentation on how many credits are required for a typical minor. The *Typical Credits* expected for a minor acts as a baseline for normalization (see below).

In addition to the number of credits needed for the minor, we looked at *Open Credits* for Biology, Chemistry, Math, Physics, and Psychology. For most 120-credit undergraduate programs, 50-60 credits are general education requirements and 30-40 credits are major requirements, leaving 20-40 credits as electives or "open credits" [11]. Comparing *Open Credits* to *Actual Credits* helps determine if a minor in Computer Science is possible without adding additional semesters or credits beyond the major degree. We focused on science degrees due to the growing requirement for programming skills in all STEM job listings [10]. To estimate *Open Credits*, we used a publicly available web-based LLM and randomly evaluated the results to ensure the LLM's accuracy. In every evaluation, it returned correct answers within 1-4 credits. This gives us confidence the *Open Credit* estimation was close enough to determine if a Computer Science minor could be completed without adding additional courses (and thus extra semesters) beyond the *Open Credits* available within a degree.

While most college degrees entail 120 credits, with 3-4 credits/course, some schools follow very different systems, including 1 credit per course or 12-16 credits per course. To normalize the data relative to the university, we divided *Advertised* or *Actual Credits* by *Typical Credits*. This balances every university against its expected credit count for a minor. For example: if the Typical Credits found for University One are 15 credits, but the Advertised Credits are 16, and the Typical Credits for University Two are 18 credits with the Advertised Credits being 21 credits, then it would be:

$$UniversityOneAdvertisedCredits = \frac{16}{15} = 1.07$$

$$UniversityTwoAdvertisedCredits = \frac{21}{18} = 1.17$$

If the Actual Credits are 24 for University One, and 47 for University Two, then:

$$UniversityOneActualCredits = \frac{24}{15} = 1.60$$

$$UniversityTwoActualCredits = \frac{47}{18} = 2.61$$

Since this is normalized against the university norm, this allows us to compare universities regardless of number of credits per course.

3.3 Additional Minors

Additionally, we collected data on any other minors offered by the Computer Science department, such as Data Science, Bioinformatics, and Software Engineering. Fifty-nine of the 120 universities had minors offered by the Computer Science department, but were not titled Computer Science. A category was created if there were at least four different universities that offered a minor with the same (or nominally different) title. The following categories emerged: Systems/Networks, IT, Robotics, Engineering (example: Minor in Scientific Engineering and Computing), Bioinfomatics, Data Science, Cybersecurity, AI/Machine Learning, Computing/Applied

[2]https://curricularanalytics.org

Table 1: Overview statistics of minors in computer science.

Requirement	Total (n=104)	Percent
Calculus Required	52	50.00%
Data Structures Required	73	70.19%
Discrete Math Required	55	52.88%
Entry Requirements Exist	73	70.19%
Hidden Math Requirement	55	52.88%

Computing, and Software Development. Anything else was placed in an 'Other' category.[3]

4 Results

The analysis led to several findings. First, it revealed that, while seventy-nine (75.96%) minors had math requirements, in fifty-five of these programs (52.88%) the math requirements were hidden; meaning they were not included in the advertised curricula but were prerequisites to classes that were included. Digging in deeper, we found that fifty-two programs (50.00%) required Calculus I. This is not surprising given that 96% of Computer Science bachelor's degrees require Calculus, usually as a prerequisite for Discrete Math [9]. That said, this finding raises the important question as to whether the inclusion of Calculus in the minor is intentional or whether it might result from unintentional structural overlap and dependencies with the major. Fifty-five (52.88%) of the minors studied explicitly required Discrete Math, typically as a prerequisite (whether explicit or hidden) for more advanced courses. Finally, seventy-three programs (70.19%) required Data Structures. For all courses, we were conservative in classifying courses as non-equivalent when names were ambiguous.

A second finding is that seventy-three programs (70.19%) imposed entry requirements, including prerequisite courses (often CS 1 and Calculus I) and GPA minimums that typically mirrored the admission standards of the major degrees. These requirements create systemic barriers for students, [7] potentially limiting minor accessibility. Summarized results of the 104 minor analysis are shown in Table 1.

A third major finding is that *nearly all* of the programs studied required additional credits beyond the advertised amounts. Figure 1 plots normalized advertised credits against actual requirements. For Advertised Credits, the first quartile was 1.0 (where 1.0 indicates exact alignment with university-typical minor credits), median 1.06, and third quartile 1.27. For Actual Credits, the first quartile was 1.17, median 1.46, and third quartile 1.78 (where 2.0 would indicate double the typical minor credits). This suggests Computer Science departments advertise minors that closely align with institutional norms but that Actual Credits reveal a different reality. A paired t-test confirms significant difference between Advertised and Actual credits ($t = -6.8569$, $p < 0.00001$), indicating Computer Science minors commonly require nearly *double* the credits recommended by the institution.

[3]Data is available at https://tinyurl.com/4ssb7vbw including links to the minors found and curricular maps for every minor.

The next finding relates to our interest in examining whether the minor could fit within the standard degree elective allowances for other STEM majors (i.e., a standard 120-credit degree with 40 core and 50 major credits leaves 30 "open elective" credits available for minoring). For this portion of the analysis, we looked at Biology, Chemistry, Math, Physics, and Psychology degrees because these majors typically have high major credit requirements but increasingly include requirements related to computational proficiency. Ideally, a Computer Science minor should be attainable within the available "open elective" credits for these other STEM majors. The results, however, tell a different story.

Figure 2 plots the difference between actual credits needed for a CS minor and open credits available in the 5 selected science programs. A value of zero would indicate the number of actual credits required for a given university's CS minor was the exact number of estimated open elective credits available for that science program. A negative value indicates that the Computer Science minor required more credits than the student had available, which has time and money implications for students.

The only science program with a positive median score was Psychology at 0.33, meaning students could earn the CS minor and still have 1-2 courses for other electives. Biology, Chemistry, Math, and Physics had respective median scores of -0.43, -0.53, -0.18, -0.50, meaning students in those programs would require 2-3 additional courses beyond the typical degree at the university. Even if these programs had an overlap in credits with the minor (e.g., they required the same Calculus course), it would still mean students had to know they were going to minor very early in their post-secondary experience and that they could never take an elective outside of the minor, let alone balance courses to avoid overloading semesters with too much technical content.

Next, we examined the curricular structure of the minors, analyzing both the structural complexity and delay factors of the degrees. Figure 3 shows the histogram of structural complexity across programs. We would estimate a minor with a four-course rigid prerequisite chain and additional courses to have a complexity of around forty. About one third (n=36) of the 104 minors were

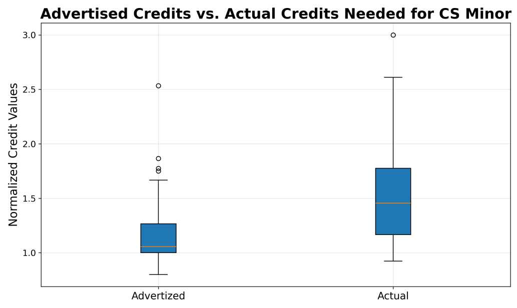

Figure 1: Advertised Credits (Degree Plan) vs. Actual Credits Required.

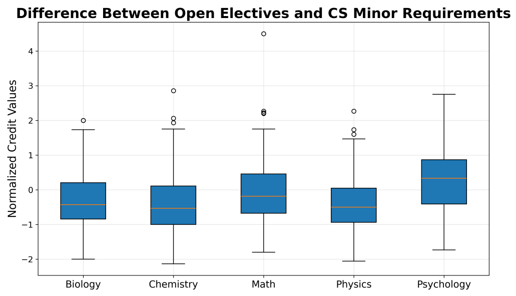

Figure 2: Difference Between Open Elective Credits for Science Degrees and Computer Science Minor.

Table 2: High centrality courses grouped by similar names, count of minors with the course as highest centrality, and percent of minors that had that course.

Course Category	Count	Percentage
Data Structures and Algorithms	57	54.8%
Computer Science 1	18	17.3%
Object-Oriented Programming	13	12.5%
Discrete Mathematics/Structures	12	11.5%
Intermediate/Advanced Programming	10	9.6%
Computer Systems/Organization	8	7.7%
Software Development/Engineering	8	7.7%
Algorithms	6	5.8%
Programming Languages	4	3.8%
Systems Programming	4	3.8%

in the 36-51.50 range and another third (n=37) schools had a complexity greater than 51.50. High structural complexity means there is little room for “error” (e.g., needing to repeat a course or not taking the next course in the sequence at a particular time). In sum, high structural complexity is a way to measure systemic barriers to a degree, highlighting rigidity that makes it hard for students to pursue the degree and unattractive to them to try to do so.

A key measure within curricular complexity is delay factor, which measures the longest prerequisite chains and, therefore, the shortest amount of time needed to earn the minor. For example, CS 1 $\longrightarrow$ CS 2 $\longrightarrow$ CS 3 $\longrightarrow$ Software Design creates a delay factor of four. Figure 4 shows a histogram of the delay factors for the 104 minors in the sample. We see that 75% of the minors investigated had a delay factor of 4 or more, meaning that a student would need two or more years to complete the degree. Given that most students don’t even decide on minoring until late into their second or third year, this delay factor makes minors unattractive and even unattainable for the majority of students (Without even taking into account entry requirements!).

A second measure within curricular complexity is course centrality, which identifies the course/pre-requisite that unlocks the most number of future courses. Table 2 shows the top ten courses to appear across all 104 degree programs. We see that Data Structures and Algorithms has the highest centrality in 54.8% of the programs (this doesn’t mean other programs don’t require Data Structures, just that the course doesn’t have the same prerequisite structure). At these schools, if a student fails Data Structures and needs to repeat, their progress in the degree is functionally blocked, possibly motivating them to drop the minor.

Overall, our analysis indicates the following: that CS minors are likely to: 1) contain "hidden" requirements, 2) be larger than what is advertised and what can be reasonably accomplished by other STEM majors with open electives; 3) have rigid curricular structures, including a high risk of delay due to specific courses such as Data Structures and Algorithms.

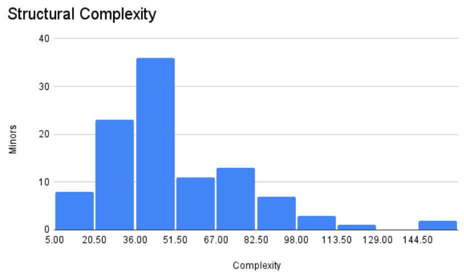

Figure 3: Structural Complexity Across Programs.

Table 3: Additional minors often hosted by the Computer Science program.

Minor	Count	Percent
Other	19	15.83%
Data Science	17	14.17%
IT	15	12.50%
Bioinformatics	10	8.33%
Cybersecurity	10	8.33%
Applied Computing	8	6.67%
Engineering	7	5.83%
Systems/Networks	6	5.00%
AI / Machine Learning	5	4.17%
Robotics	5	4.17%
Software Development	5	4.17%

However, a minor in Computer Science isn’t the only minor departments might offer. In Table 3, we show the variety of minors offered by the CS departments in our data set that did not have “Computer Science” in the title. Data Science had the highest count at 17, followed by Information Technology (15), Bioinformatics (10) and Cybersecurity (10). In the “other” category we capture a range of minors such as: Human Centered Computing, Digital Design, Game Design, and Web Design (3 or fewer). We found that if a school offered one minor outside of Computer Science, they tended to offer multiple; we found 107 “additional” minors even though only 59 of the 120 schools offered these.

Threats To Validity: Three methodological assumptions may limit our findings. First, we assumed students were calculus-ready and would choose the shortest completion path. This may not reflect student reality, particularly for non-STEM majors. Since degree programs state they assume students are calculus-ready, this assumption aligns with institutional expectations and reduces complexity estimates, but potentially understates barriers for students.

Second, assuming that students select the quickest graduation path probably also underestimates program complexity. Many minors offer specialized tracks (e.g., machine learning) requiring additional prerequisites. Our shortest-path approach biased results toward lower complexity and more uniform structures.

Third, our analysis of science majors’ ability to complete CS minors did not account for credit “double-dipping” opportunities where courses satisfy both major and minor requirements. While this could reduce actual credit burden, the substantial variation in university policies and major requirements made systematic analysis impractical.

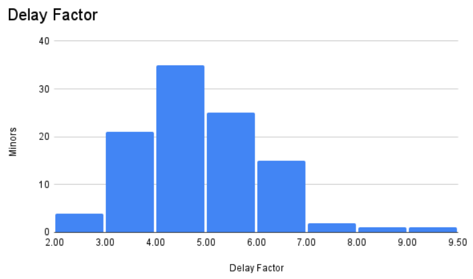

Figure 4: Delay Factor for Programs Indicates the Shortest Time to Complete the Minor.

5 Recommendations

In analyzing and comparing 104 CS minor degree plans, we arrive at a number of insights and recommendations.

Prerequisites Drive Programs: In many of the minors evaluated, prerequisites - hidden and explicit - had significant impact on the program structure and total number of credits. Overall, the presence of these prerequisites limits access to the minor as they functionally force students to decide to pursue the minor very early on in their time at university (though most students tend to make this decision later on). Instead, a best practice would be to ensure that prerequisites are aligned with the content that is truly intended and needed for the minor. Rethinking the prerequisites in this way would allow more flexibility in the timing of minor declaration for students who are exploring and/or looking to complement their major. As demonstrated by Ganesan et al. [16], removing prerequisite barriers has positive benefits for the major degree as well.

Calculus Requirements: The placement of Calculus in CS degrees has come under scrutiny of late. The 2023 ACM computing curricula guidelines encourage that Calculus be delayed [28] such that it not act as a barrier to students looking to try out computing. Other sources echo this recommendation [9, 33]. Not surprisingly, we are confronted with the placement of Calculus in the CS minor as well. To be clear, the point is not *whether* Calculus is needed for a Computer Science minor, but rather *when* the course should be taken by the student. By including Calculus as a prerequisite to most CS minor programs, the course serves as a barrier for students exploring Computer Science from other disciplines, many of which do not require Calculus in their degree plans.

Be Honest With The Requirements: The majority of programs studied had additional course requirements beyond what was publicly advertised. This is obviously very confusing and likely frustrating for students. Arguably, a top criterion for a student in determining whether they should pursue a minor is whether they can do so without adding time to graduation. It is important for every Computer Science program to reevaluate how their minor is presented and ensure that what is being shared publicly is accurate.

Entry Requirements: We found that 70.19% of all minors have entry requirements, which could be intimidating for students looking to try out or explore Computer Science. While there is an argument that a minor should match the major entry requirements to prevent students from finding a work around into the major, we challenge that if they can be successful in the CS course work, then maybe the entry requirement is not needed. It would benefit universities to reconsider the goals of the entry requirements, evaluate if they are really needed for the minor, and ensure that there are feasible pathways to allow students to discover CS. A minor today may become a major tomorrow, so building viable pathways is beneficial for both the student and the CS program itself.

Interdisciplinary Minors / Pathways: The analysis of science open electives highlights the difficulty even science majors have in pursuing a CS minor. This is problematic since graduates of these majors increasingly need Computer Science for their jobs. With the rise of LLMs, this is even more important as AI literacy will become essential for these majors (and, arguably, all majors). With the exception of Data Science, there were very few "additional" minors that targeted other science majors. For example, out of the 107 minors in this "additional" category, only three were Human Computer Interaction, which is a natural area at the juncture of Computer Science and Psychology (the most popular degree on many campuses). We encourage CS departments to partner with other departments (both science and humanities) to promote interdisciplinary majors and minors - or, at least, to facilitate feasible pathways into the minor to promote exploration and diversity of thought. Such partnerships should also include intentional opportunities for early discovery of Computer Science (e.g. CS as part of the general education requirements [32]), which can be effective strategies for attracting students to a minor program.

6 Conclusions and Future Work

This survey of Computer Science minors has resulted in many interesting, individual insights and one overarching finding above all, namely that, in general, it is very difficult for students to minor in Computer Science. This is because current minor degree structures often have restrictive entry requirements, hidden prerequisite chains, and credit loads that exceed institutional norms. As such, students seeking to minor in CS would need to make this choice very early on in their college career and may, even then, experience difficulties completing the degree on time.

Given the paucity of research on minors to date, the results presented are meant to be a starting point for Computer Science programs. We hope that the analysis motivates CS leaders to reevaluate their minor(s) and ask whether the structures they have put in place are aligned with the intent and goals of the degrees. Minors can serve many goals (including being a pathway to the major). It is our belief that the CS minor matters because it is a structured way for students to grow their interdisciplinary skill set - something that is becoming even more important in the age of AI. In sum, as computational literacy becomes increasingly valuable across fields, ensuring these pathways remain accessible to diverse student populations seeking to integrate these skills with their primary academic interests becomes correspondingly important.

Finally, we close by emphasizing that future research should address the limitations identified in this study. The absence of enrollment data disaggregated by demographics limited our understanding of how accessibility impacts enrollment in CS minors. Programs implementing improvements would benefit from tracking outcomes disaggregated by demographics. We encourage others to expand upon previous work that looks at interdisciplinary minors (e.g. Bioinformatics or HCI) or minors in specialized areas (e.g. AI/ML or Cybersecurity) that promote CS and that can be important attraction mechanisms for students from populations historically underrepresented in computing. Finally, our research did not uncover any curricular overlaps across the minors beyond CS 1 and CS 2, which means that, as it currently stands, there is little consensus on what a minor in Computer Science even is. The field would be well served by developing guidelines defining core competencies for Computer Science minors that are distinct and do not just "borrow" from major requirements.

Acknowledgment: This project was partially funded by the CIC at Northeastern University. We would like to thank Carter Ithier and Heather Lionelle for early reviews.

References

[1] [n. d.]. Computer Science, Minor Northeastern University Academic Catalog. https://catalog.northeastern.edu/undergraduate/computer-information-science/computer-science/minor/#programrequirementstext

[2] William W. Agresti. 1976. Computer science as an interdisciplinary study. In *Proceedings of the sixth SIGCSE technical symposium on Computer science education - SIGCSE '76*, Vol. 8. ACM Press, New York, New York, USA, 12–14. doi:10.1145/800144.804747

[3] Association for Computing Machinery. 2024. Advancing Computing as a Science & Profession. http://www.acm.org/

[4] Valerie Barr. 2016. Disciplinary thinking, computational doing. *ACM Inroads* 7, 2 (5 2016), 48–57. doi:10.1145/2891414

[5] BradyAlyce, CutterPamela, SchultzKelly, Alyce Brady, Pamela Cutter, and Kelly Schultz. 2004. Benefits of a CS0 course in liberal arts colleges. *Journal of Computing Sciences in Colleges* 20, 1 (10 2004), 90–97. doi:10.5555/1040231.1040243

[6] Carla Brodley, Valerie Barr, Elsa Guter, Mark Guzdial, Ran Libeskind-Hadas, and Bill Manaris. 2024. ACM 2023: CS + X-Challenges and Opportunities in Developing Interdisciplinary-Computing Curricula. *ACM Inroads* 15, 3 (8 2024), 42–50. doi:10.1145

[7] Carla E. Brodley. 2022. Why universities must resist GPA-based enrollment caps in the face of surging enrollments. *Commun. ACM* 65, 8 (8 2022), 20–22. doi:10.1145/3544547

[8] Carla E. Brodley, Benjamin J. Hescott, Jessica Biron, Ali Ressing, Melissa Peiken, Sarah Maravetz, and Alan Mislove. 2022. Broadening Participation in Computing via Ubiquitous Combined Majors (CS+X). *SIGCSE 2022 - Proceedings of the 53rd ACM Technical Symposium on Computer Science Education* 1 (2 2022), 544–550. doi:10.1145/3478431.3499352

[9] Carla E. Brodley, Mckenna Quam, and Mark Weiss. 2024. An Analysis of the Math Requirements of 199 CS BS/ BA Degrees at 158 U.S. Universities. *Commun. ACM* 67, 8 (7 2024), 122–131. doi:10.1145/3661482

[10] Burning Glass Technologies. 2016. *Beyond Point and Click the Expanding Demand for Coding Skills*. Technical Report June. 11 pages. https://www.burning-glass.com/research-project/coding-skills/

[11] Genevieve Carlton. 2023. Do You Need a Minor in College? Pros and Cons | BestColleges. https://www.bestcolleges.com/blog/do-you-need-college-minor/

[12] Computing Accreditation Commission, ABET. 2024. Accreditation Criteria and Supporting Documents. https://www.abet.org/accreditation/accreditation-criteria/cac-criteria/

[13] David Cowden, April O'Neill, Erik Opavsky, Dilan Ustek, and Henry M. Walker. 2012. A C-based introductory course using robots. *SIGCSE'12 - Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (2012), 27–31. doi:10.1145/2157136.2157150

[14] Zachary Dodds, Ran Libeskind-Hadas, and Eliot Bush. 2010. When CS 1 is biology 1: Crossdisciplinary collaboration as CS context. *ITiCSE'10 - Proceedings of the 2010 ACM SIGCSE Annual Conference on Innovation and Technology in Computer Science Education* (2010), 219–223. doi:10.1145/1822090.1822152

[15] Stefanie Colino Dube and Albert Lionelle. 2025. Does ABET Accreditation Influence the Representation of Women in CS Programs ?. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2 (SIGCSE TS 2025), February 26-March 1, 2025, Pittsburgh, PA, USA*. Association for Computing Machinery, Pittsburgh, 1–2. doi:10.1145/3641555.3705168

[16] Sumukhi Ganesan, Albert Lionelle, Catherine Gill, and Carla Brodley. 2025. Does Reducing Curricular Complexity Impact Student Success in Computer Science? *SIGCSE TS 2025 - Proceedings of the 56th ACM Technical Symposium on Computer Science Education* 1 (2 2025), 360–366. doi:10.1145/3641554.3701915

[17] Michel Grosz, Michal Kurlaender, and Ann Stevens. 2022. Capacity and Flexibility in Community College CTE Programs: Program Offerings and Student Success. *Research in Higher Education* 63, 1 (2 2022), 140–188. doi:10.1007/S11162-021-09645-9

[18] Michael Haungs, Christopher Clark, John Clements, and David Janzen. 2012. Improving first-year success and retention through interest-based CS0 courses. *SIGCSE'12 - Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (2012), 589–594. doi:10.1145/2157136.2157307

[19] Gregory L. Heileman, Chaouki T. Abdallah, Ahmad Slim, and Michael Hickman. 2018. Curricular Analytics: A Framework for Quantifying the Impact of Curricular Reforms and Pedagogical Innovations. (11 2018). https://arxiv.org/abs/1811.09676v1

[20] Gregory L. Heileman, Hayden W. Free, Johnny Flynn, Camden Mackowiak, Jerzy W. Jaromczyk, and Chaouki T. Abdallah. 2020. Curricular Complexity Versus Quality of Computer Science Programs. (6 2020). https://arxiv.org/abs/2006.06761v1

[21] Gregory L. Heileman, Ahmad Slim, Michael Hickman, and Chaouki T. Abdallah. 2017. Characterizing the complexity of curricular patterns in engineering programs. In *ASEE Annual Conference and Exposition, Conference Proceedings*, Vol. 2017-June. American Society for Engineering Education. doi:10.18260/1-2--28029

[22] Gregory L. Heileman, William G. Thompson-Arjona, Orhan Abar, and Hayden W. Free. 2019. Does Curricular Complexity Imply Program Quality? *ASEE Annual Conference and Exposition, Conference Proceedings* (6 2019). doi:10.18260/1-2--32677

[23] Amanda Holland-minkley, Jefferson College, Usa E Jakob Barnard, Andrea Tartaro, and James D Teresco. 2023. Computer Science Curriculum Guidelines: A New Liberal Arts Perspective. 1 (2023). https://computing-in-the-liberal-arts.github.io/CS2023/

[24] Lindsay Jarratt, Freda B. Lynn, Yongren Shi, and Katharine M. Broton. 2024. Up-or-Out Systems? Quantifying Path Flexibility in the Lived Curriculum of College Majors. *Research in Higher Education* 65, 6 (9 2024), 1185–1207. doi:10.1007/S11162-024-09789-4

[25] Jinya Jiang, Richa Kafle, Christa Lehr, Simone Wright, Clarissa Guitierrez-Godoy, and Christine Alvarado. 2024. Understanding California's Computer Science Transfer Pathways. *SIGCSE 2024 - Proceedings of the 55th ACM Technical Symposium on Computer Science Education* 1 (3 2024), 604–610. doi:10.1145/3626252.3630956

[26] Natalia Khuri, Wendy Lee, K. Virginia Lehmkuhl-Dakhwe, Miri VanHoven, and Sami Khuri. 2020. Interdisciplinary Minor in Bioinformatics. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. ACM, New York, NY, USA, 407–412. doi:10.1145/3328778.3366804

[27] Sami Khuri, Miri Vanhoven, and Natalia Khuri. 2017. Increasing the capacity of STEM workforce: Minor in bioinformatics. *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE* (3 2017), 315–320. doi:10.1145/3017680.3017721

[28] 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.

[29] Kathleen J. Lehman, Carla Brodley, Mark Guzdial, Paul Tymann, and Aman Yadav. 2024. Re-making CS Departments for Generation CS. *SIGCSE 2024 - Proceedings of the 55th ACM Technical Symposium on Computer Science Education* 2 (3 2024), 1535–1536. doi:10.1145/3626253.3631655

[30] William H. Linder. 1979. The computer science minor, a description and a proposal. *ACM SIGCSE Bulletin* 11, 2 (6 1979), 40–42. doi:10.1145/988923.988928

[31] Hilary L Link. 2021. Minors Matter: How Interdisciplinary Solutions Benefit Institutions and Students. (2021).

[32] Albert Lionelle, Josette Grinslad, and J Ross Beveridge. 2020. CS 0: Culture and Coding. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (2020), 227–233. doi:10.1145/3328778.3366795

[33] Albert Lionelle, McKenna Quam, Carla Brodley, and Catherine Gill. 2024. Does Curricular Complexity in Computer Science Influence the Representation of Women CS Graduates?. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. ACM, New York, NY, USA, 729–735. doi:10.1145/3626252.3630835

[34] Engineering National Academies of Sciences and Medicine. 2017. Assessing and Responding to the Growth of Computer Science Undergraduate Enrollments. *Assessing and Responding to the Growth of Computer Science Undergraduate Enrollments* (10 2017), 1–229. doi:10.17226/24926

[35] Engineering National Academies of Sciences and Medicine. 2018. *Data Science for Undergraduates*. National Academies Press, Washington, D.C. doi:10.17226/25104

[36] Kim Parker and Ruth Igielnik. 2020. What We Know About Gen Z So Far | Pew Research Center. https://www.pewresearch.org/social-trends/2020/05/14/on-the-cusp-of-adulthood-and-facing-an-uncertain-future-what-we-know-about-gen-z-so-far/

[37] Ahmad Slim, Gregory L. Heileman, Wisam Al-Doroubi, and Chaouki T. Abdallah. 2016. The impact of course enrollment sequences on student success. *Proceedings - International Conference on Advanced Information Networking and Applications, AINA* 2016-May (5 2016), 59–65. doi:10.1109/AINA.2016.140

[38] Paul A Stock. 2019. Factors that Influence a College Student's Choice of an Academic Major and Minor. https://www.academia.edu/121124790/Factors_that_Influence_a_College_Students_Choice_of_an_Academic_Major_and_Minor

[39] James D. Teresco, Andrea Tartaro, Amanda Holland-Minkley, Grant Braught, Jakob Barnard, and Douglas Baldwin. 2022. CS Curricular Innovations with a Liberal Arts Philosophy. *SIGCSE 2022 - Proceedings of the 53rd ACM Technical Symposium on Computer Science Education* 1 (2 2022), 537–543. doi:10.1145/3478431.3499329

[40] Zoë J Wood, John Clements, Zachary Peterson, David Janzen, Hugh Smith, Michael Haungs, Julie Workman, John Bellardo, and Bruce DeBruhl. 2018. Mixed approaches to cs0: Exploring topic and pedagogy variance after six years of cs0. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 20–25.