Data-Driven Discovery of Anchor Points for PDC Content

Matthew McQuaigue
Erik Saule
Kalpathi Subramanian
mmcquaig@charlotte.edu
esaule@charlotte.edu
krs@charlotte.edu
UNC Charlotte
Charlotte, North Carolina, USA

Jamie Payton
payton@temple.edu
Temple University
Philadelphia, Pennsylvania, USA

ABSTRACT

The Parallel and Distributed Computing community has been interested in integrating PDC content into early CS curriculum to prime the students for more advanced materials and build a workforce able to leverage advanced computing infrastructure. To deploy this strategy at scale, it is important to identify anchor points in early CS courses where we can insert PDC content.

We present an analysis of CS courses that primarily focuses on CS1 and Data Structure courses. We collected data on course content through in-person workshops, where instructors of courses classified their course materials against standard curriculum guidelines.

By using these classification, we make sense of how Computer Science is being taught. We highlight different types of CS1 and Data Structure courses. And we provide reflection on how that knowledge can be used by PDC experts to identify anchoring points for PDC content, while being sensitive to the needs of instructors.

KEYWORDS

CS Education, Curriculum Guidelines, Course Model, Integrating PDC in Early CS

ACM Reference Format:
Matthew McQuaigue, Erik Saule, Kalpathi Subramanian, and Jamie Payton. 2023. Data-Driven Discovery of Anchor Points for PDC Content. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3624062.3624099

1 INTRODUCTION

Parallel and Distributed Computing is a topic that is critical to Scientific Computing, Machine Learning, Simulation, and Big Data; especially to process data at scale. Having a large population of scientists and engineers who can leverage modern computing hardware is critical to delivering the next generation of computing. The challenge has been on finding the best strategy to integrate PDC education into the CS curriculum without too much disruption.

The NSF/IEEE-TCPP PDC curriculum guidelines released in 2012 (PDC12) represents an effort to bring more parallel computing concepts into early computer science courses. To date, it has been moderately successful. But insufficient knowledge or training in PDC topics among instructors has been a challenge to overcome. In particular, instructors would like access to materials that are related to their current course learning objectives. Other strategies include having PDC experts create learning materials for instructors in early CS courses, but developing appropriate materials one course at a time is a herculean task.

The strategy we pursue is to get a deeper understanding of the structure of these early courses so that one can develop materials for many courses at once. However, we believe that the structure and content of these courses are more varied than we usually imagine them to be. We intend to use the ACM/IEEE Computer Science curriculum guidelines [11, 18] as a lingua franca for classifying/aligning the course content against these guidelines. By bringing together large numbers of courses into this framework, we can get a better understanding of how early CS courses are being taught, what topics are being covered and the level of student expectations. This level of understanding of a course structure then makes it possible to provide recommendations for PDC experts to develop materials (lectures, assignments, labs) that can be integrated into specific part(s) of a wide range of courses.

Over the past three years, we have built and used the CS Materials system [9, 10] to build a collection of early CS courses (CS1, CS2, Data Structures, Algorithm Analysis) aligned to the ACM/IEEE CS 2013 guidelines. CS Materials is a public resource that allows instructors to assess their course learning materials against the ACM and PDC Curriculum Guidelines, by classifying their course in the system.

In this paper, we describe the analysis of the early CS courses that we have collected over the past three years, variations within these courses and their implications for introducing PDC content into them.

Contributions. We use a NonNegative Matrix Factorization (NNMF) scheme to analyze two sets of courses, roughly corresponding to CS1 and Data Structures and Algorithms and analyze the resulting categories; the results bring out flavors of these courses corresponding to the underlying content, as well as the level of agreement among them. The factorization also allows us to look at the content of these courses and recommend PDC content into specific parts of

SC-W 2023, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0785-8/23/11…$15.00
https://doi.org/10.1145/3624062.3624099

these courses. Further study will be needed with a larger sample size to confirm these results.

2 BACKGROUND

2.1 Curriculum Guidelines

ACM and IEEE produce computing curriculum guidelines and the latest version is from 2013 [11] with an expected revision by Dec. 2023 [18]. The 2013 guidelines specify a 'redefined body of knowledge, a result of rethinking the essentials necessary for a Computer Science curriculum'. The guidelines provide numerous exemplars at the syllabus level of actual courses and programs that can be adopted by CS departments. The guidelines divide the body of knowledge into *knowledge areas*; which are further divided into *knowledge units*; which subdivide further into *topics* and *learning outcomes*. Learning outcomes have three levels of mastery: *familiarity*, *usage* and *assessment*. The CS Materials system we use currently supports the 2013 CS curriculum guidelines. Sub-areas of computing have also developed their own standards, such as parallel computing [14], cybersecurity [3], data science [2] and high school CS curriculum [6, 7].

The 2012 NSF/IEEE-TCPP curriculum for Parallel Distributed Computing [14] (we will denote PDC12) is an effort to accurately map the PDC topics that are necessary for all students to know. It is divided in four areas: Algorithm, Architecture, Programming, and Cross-Cutting and Advanced topics. Contrary to the CS13 guidelines, the PDC12 curriculum presents learning outcomes only as a description of topics rather than as separate items. The PDC guidelines also associate Bloom levels [5] (such as Know, Comprehend, and Apply) with the topics to clarify the minimum level of understanding a student should have. While the CS13 curriculum groups topics into a core-1 (must cover 100%), core-2 (should cover 80% at least), and elective; the PDC curriculum only exposes two levels: core and elective. The PDC curriculum is currently under revision with a new version coming in 2023 (a beta version was released in late 2020 [15]).

2.2 Learning Material Repositories

The CS education community has published pedagogical content, especially for early CS courses. For CS0, CS1, and CS2 assignments unrelated to PDC, Nifty assignments are a popular set of assignments [16]. For PDC content, the Peachy Parallel Assignments [8] and PDC Unplugged [13] are common resources.

Nifty Assignments. The Nifty assignments repository is a set of assignments that have been collected since 1999 (over 100 assignments) through an annual competition, as part of the ACM SIGCSE conference. These selected assignments are presented at the conference and are chosen based on their engagement, adaptability, and scalability. These assignments are intended for early courses like CS0, CS1, and CS2. They now include metadata including topics, difficulty level, strengths/weaknesses, dependencies, and variants.

Peachy Parallel Assignments. The EduPar and EduHPC workshops launched the Peachy Parallel assignments to promote well-designed, exciting and intriguing tasks that contain some [8] features of distributed and parallel computing. Peachy Parallel's emphasis on adoption has been successfully implemented in real-world educational settings. Peer review, publication, and presentation of the tasks at EduPar and EduHPC have resulted in 11 assignments released in 2018 and 9 published in 2019.

PDC Unplugged. A repository called PDC Unplugged includes a variety of unplugged parallel and distributed computing activities. The activities cover CS0, CS1, and CS2 early courses, Data Structures and Algorithms sophomore courses, and advanced courses. Throughout these sessions, there are around 130 different activities. These exercises offer a simple introduction to PDC resources and can be included in a larger assignment that starts with unplugged activities and progresses to a more formal programming activity. The unplugged activities in PDC Unplugged are linked to the entries of the curricular standards that they address.

3 DATA COLLECTION

3.1 CS Materials

The CS Materials system [9, 10] was constructed to facilitate the classification of learning materials against national standards for curriculum guidelines. Classifying learning materials against curriculum guidelines facilitates comparing learning materials or whole courses and programs against a common baseline. Currently, CS Materials supports the ACM/IEEE 2013 CS guidelines [11] as well as the NSF/IEEE-TCPP 2012 curriculum guidelines for parallel and distributed computing [14]. We currently have over 30 courses classified in our system. Each course is made up of a collection of materials that are mapped to topics and learning outcomes in the ACM/PDC guidelines. All courses, their classifications, and contributed materials can be accessed on our CS Materials website [17]. In total, about 1700 materials have been added to CS Materials.

3.1.1 Visualization. CS Materials currently supports three visualization types to display course topics/outcomes: radial views, similarity graphs, and a table (matrix) view for interactively classifying course topics and outcomes.

To understand a course's coverage in terms of topics or objectives, users can view a hit-tree in the CS Materials system. The hit-tree is a tree representation where items associated with the course are highlighted in a subset of the ACM/PDC classification tree. The tree is arranged radially by identifying the level with the most nodes, known as the reference level, and uniformly spacing all nodes at that level. The radial tree can also be used for alignment between two subsets of materials where the size of the node indicates the number of materials that are classified against that classification item, while the node color uses a divergent scale, ranging between the two sets of materials (mid-range of the scale represents the materials are fully aligned)

The matrix visualization provides users the ability to quickly edit the curriculum mapping of multiple materials at once. This matrix displays materials as columns and curriculum-mapped tags as rows. The cell indicates whether a particular tag maps to a material, and the view is interactive to allow to quickly edit curricular mappings. To help users understand the materials and their course mapping, entries in the matrix view are bi-clustered to highlight related material/tag patterns in the curriculum.

Class Name	CS1	OOP	DS	Algo	SoftEng	PDC
UNCC ITCS 2214 KRS Data Structures and Algorithms			X			
UNCC ITCS 2214 Saule Data Structures and Algorithms			X			
UNCC ITCS 3145 Saule Parallel and Distributed Computing						X
UNCC ITCS 3112 KRS Object Oriented Programming		X				
CCC CSCI 40 Kerney CS1	X					
Hanover cs225 Wahl Algorithmic Analysis 2021				X		
VCU CMSC 256 Duke Data Structures and Object-oriented Programming		X	X			
CCC CSCI 41 Kerney CS2						
BSC CAC 210 Wagner Data Structures and Algorithms			X			
UNCC ITCS 2215 KRS Algorithms				X		
GSU CSC4350 Levine Software Engineering					X	
Tulane CMPS1100 Kurdia Intro to Programming	X					
Knox CS309 Bunde Parallel Computing						X
LSU CSC 1350 Kundu Parallel Computation						X
UCF COP3502 Ahmed Computer Science 1 (CS1) Data structure and algorithm	X		X			
WashU CSE131 Singh Computer Science 1	X					
UNL CSCE 155E Bourke Computer Science I using C	X					
UNCC ITCS 4155 Payton Software Development Projects					X	
Tulane CMPS1500 Toups CS1	X					
UTSA Bopana Computer Network						

Figure 1: Courses in the dataset considered in this paper

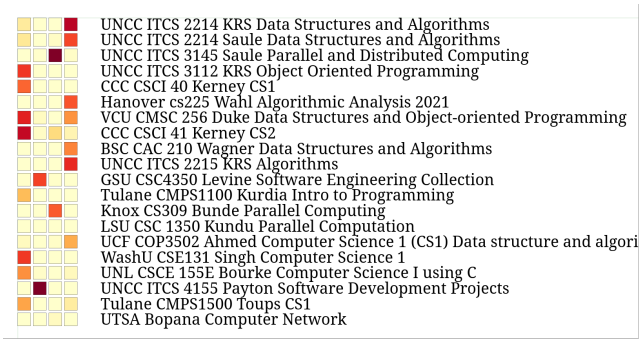


Figure 2: NNMF model of all courses with $k = 4$, W matrix only.

3.1.2 Searching. It can be helpful to search for materials such as a better set of slides or examples to explain complex concepts, engaging assignments that meet specific learning outcomes, or external materials to provide alternate explanations for students. The CS Materials system leveraging the mapping materials with topics and learning objective to find materials related to certain topics, learning objectives, and outcomes Materials can also be searched by course level, author, programming language and datasets used.

It can be difficult to find specific learning materials in current repositories. However, using CS Materials, searching for assignments or lecture materials that match specific topics and learning outcomes is easy, but also more robust. This approach is more comprehensive and reliable. Instructors can use the system to easily locate "additional materials" for their students to use in their studies.

It can also be difficult to understand how good the result of a search is and if the recommendations relate to the original search topic. To solve this, we create a graph where materials (including query and results) are vertices and the edges between them are weighted by the similarity they share. The similarities are then passed to a Multidimensional Scaling (MDS) [1] algorithm to map the materials to a 2D location where more similar materials are naturally clustered together.

3.2 Course Analysis Workshops

We conducted long-form workshops where instructors input and analyzed one of their courses. These workshops follow a 2-day format located at a university or colocated with a scientific conference. Some of the early workshops were conducted online. The first day is spent educating the attendees on modern course design, how to use CS Materials, and inputting their class in the system. The second day is spent analyzing their class. We instruct them on how to study the coverage of their class; on how to study the alignment between content delivery, activities, and assessment; on how to find new materials for their class; and on how to study the dependencies of topics in their classes.

Each workshop had about 10 attendees. In total, there are 31 courses fully classified in the system. For technical reasons, we had to exclude 11 and retain 20 in total. We list the courses in Figure 1. Based on the name of the courses, we grouped them as CS1, Object Oriented Programming, Data Structure, Algorithms, Software Engineering, or Parallel and Distributed Computing.

4 ANALYSIS

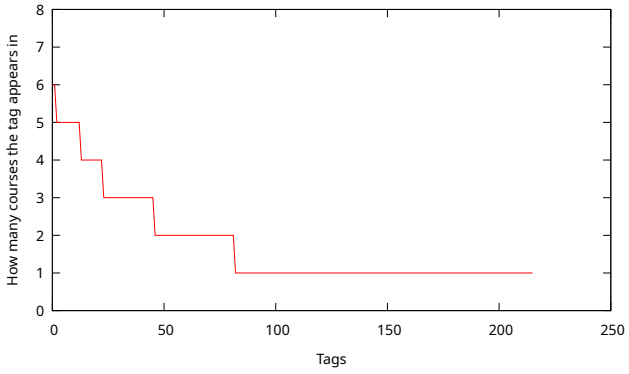
4.1 Methods: Non-Negative Matrix Factorization

A core assumption of our research method is that classifying courses against curriculum guidelines will lead to us understanding the structures and differences in courses. The first question we seek to answer is whether we see any difference in the courses based on their classification against the ACM/IEEE Computer Science 2013 guideline.

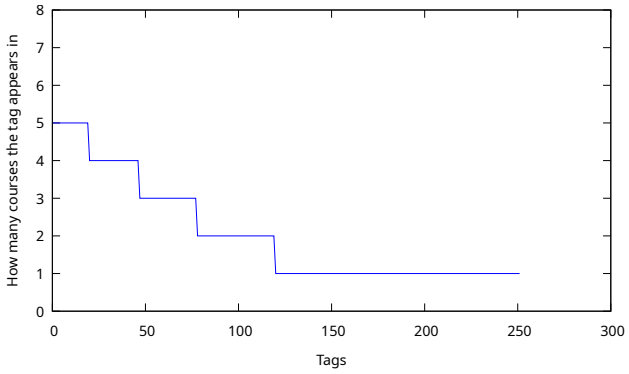
To perform that analysis we represent the courses as A, a 0-1 matrix where each row represents a course in our analysis, and each column represents an entry in the curriculum guideline. Then we will perform a nonnegative matrix factorization (NNMF) [12] to obtain two matrices W and H such that $A \approx W \times H$. While A is of dimension $Courses \times Curriculum$, W is of size $Courses \times k$, and H is of size $k \times Curriculum$ where k is a hyper parameter of this unsupervised machine learning model. All the NNMF we present are computed using scikit learn v1.3.0 with default parameters and random initialization.

This decomposition is particularly suitable for classifying objects when it is expected that these objects are linear combinations of a few types. NNMF is commonly used in Natural Language Processing for topic modeling [4] where, for instance, news articles can be modeled as a combination of topics, and these topics are associated with a particular set of words in the lexical field of that topic. Similarly, for courses, it is reasonable to expect that courses in the same space will cover the same type of computer science and if a course is outside of a single type, it will spend some time covering one and sometimes covering the other in what will approximate a linear combination. As an example, the parallel computing course of one of the authors can briefly be expressed as 20% theory, 40% shared memory programming, and 40% distributed memory programming.

To interpret the decomposition, one should think of the hyperparameter k as the number of types of courses we are trying to extract from the data. Matrix W is a mapping of particular courses to types of courses. And matrix H is a mapping of types of courses to the topics and outcomes in the guideline which is usually covered by that type of course.



(a) CS1



(b) Data Structures

Figure 3: Agreement in CS1 and Data Structure courses

4.2 Do we see the different types of courses in the data?

We computed a decomposition of all courses with $k = 4$ dimensions. We present a heat map of the W matrix in Figure 2. That matrix has 1 row per course and one column for each of the $k = 4$ dimensions. We expect each of the dimensions to represent a type of course and the decomposition supports that hypothesis. Dimension 4 has a high intensity on courses which seems to be about data structures. Dimension 2 has high intensity on courses related to software engineering. Dimension 3 has a high intensity in parallel computing courses. Dimension 1 has a high intensity in CS1 courses.

This result validates two hypotheses. First, classifying courses against curriculum guidelines enables one to differentiate and study the content of courses. Second, nonnegative matrix factorization uncovers structure in course classification data.

4.3 Is there agreement on what CS1 is?

While all the CS1 courses appeared in the same type in the analysis presented in Figure 2, they did not all belong to the type with the same intensity. This indicates that there are differences in these courses. To understand the differences, we first present in Figure 3a an overview of the agreement of topics in CS1. We show the distribution of how many courses tags appear in.

We have 6 courses in the data set that are called CS1 or intro programming. They map in total to over 200 curriculum tags. But only about 25 appear in 3 or more courses. And only 50 tags appear in 2 or more courses. This seems to indicate a significant disagreement between CS1 courses.

To understand the structure of that disagreement, we show in Figure 4 a tree view of the curriculum tags that appear in multiple CS1 courses. The curriculum guideline is organized as an ontology: there are knowledge areas that contains knowledge unit, which themselves contain topics and learning outcomes. It makes a tree visualization particularly appropriate. The red node is the root of the curriculum guideline. We labeled the knowledge areas (the first level nodes) with their names (SDF: Software Development Fundamentals, Algo: Algorithm and Complexity, Arch: Architecture and Organization, PL: Programming Languages).

The curriculum mappings that appear in 2 courses or more span 4 knowledge areas. But only 13 curriculum mappings appear in 4 courses or more and they all fall within Software Development Fundamentals and 12 of those are in the Fundamental Programming Concepts knowledge unit. This seems to indicate that there is only the most basic agreement on what CS1 covers.

4.4 Can we distinguish flavors of CS1?

Out of the count analysis and the agreement analysis, we expect to see different flavors of CS1. Nonnegative matrix factorization is a good tool to identify variants of a particular course. We computed nonnegative matrix factorization using only the CS1 courses and looked at the results for $k = 2$, $k = 3$, and $k = 4$. Manual inspection of the W and H matrices indicated that $k = 3$ was the most revealing. Indeed, $k = 4$ generated two dimensions which were almost identical, indicating an overfit. Using $k = 2$ seemed to not separate the courses as well as $k = 3$.

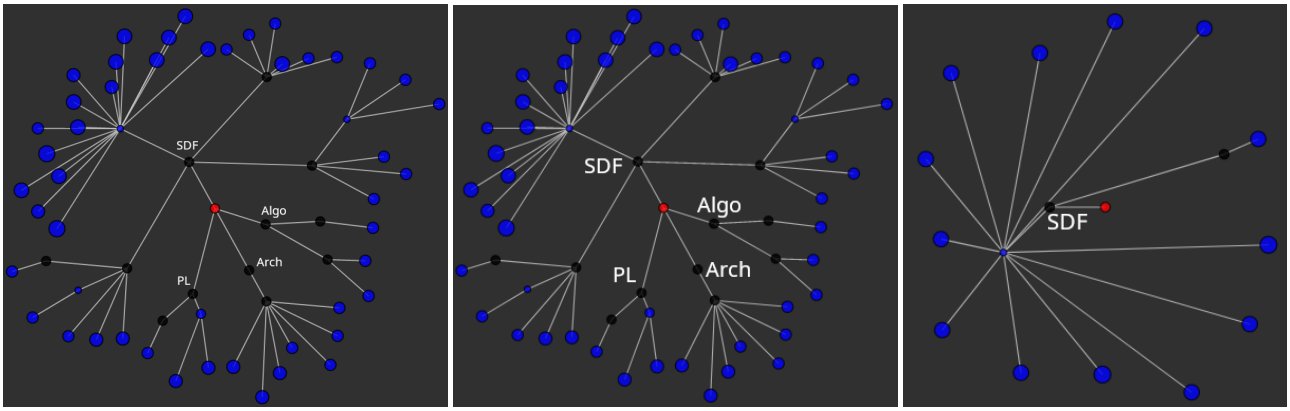
We provide visualization of both the W and H matrices in Figure 5. Let us first look at the H matrix to understand the three fundamental types extracted by the factorization.

Type 1 seems to contain primarily topics that fall within the Algorithm and Complexity Knowledge Area. These topics roughly group in Big Oh notation, complexity analysis, trees, divide and conquer algorithms and sorting. Topics in other Knowledge Areas are thematically related to sets, linked lists, stacks, and algorithm implementation from Software Development Fundamentals; and Trees and graphs from Discrete Structures.

Type 2 is a more eclectic set. It contains from SDF the core of imperative programming: variables, conditions, loops, function calls, and basic algorithms. From Architecture and Organization, it contains basic representation of information topics such as number encoding, array and struct encoding in memory, file I/O. And from SE and IAS, we find topics around correctness of programs and testing.

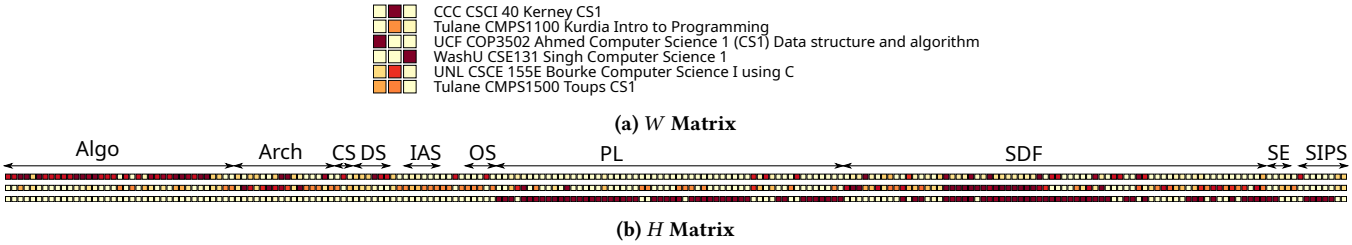
Type 3 sees most of its topics in the SDF and PL Knowledge Areas. It contains the basics of programming: variables, conditions, loops, and function calls. But it contains almost no algorithm content. Instead, it contains Object Oriented Programming topics: classes, inheritance, polymorphism, encapsulation, and generics.

Singh's class falls strongly in Type 3. Kerney's falls strongly in Type 2. And Ahmed's falls strongly in Type 1. Having a CS1 class



(a) agreement: 2 courses (b) agreement: 3 courses (c) agreement: 4 courses

Figure 4: Distribution of agreed upon classification in ACM curriculum for CS1 courses. Root in red



(a) W Matrix

(b) H Matrix

Figure 5: NNMF of CS1 courses. k=3

being primarily about algorithms and data structure and not introducing much programming seemed strange to us. Further investigations showed that at that institution the class called "Computer Science 1" is not the first of the sequence. It comes after classes called "Introduction to programming".

A similar structure happens at Tulane where both courses appear in our dataset. CMPS1100 is an "Introduction to programming" class, while CMPS1500 is called "CS1" and contains significant data structure and algorithm topics.

Two of the CS1 courses in our dataset are mostly imperative programming courses, two are a blend of imperative programming and algorithms, and one is purely a data structure and algorithm class, while one is an object oriented programming class. Further investigation showed that the object-oriented programming one is taught in Java, while the others are taught in C and Python.

4.5 Is there agreement on what DS is?

There are 5 Data Structure courses in our dataset and the distribution of counts of tags in different courses is given in Figure 3b. There is a higher agreement on the content of Data Structures than there was on CS1. Out of the about 250 curriculum tags of data structure courses, about 120 appear in two or more courses and 50 appear in more than 3 courses.

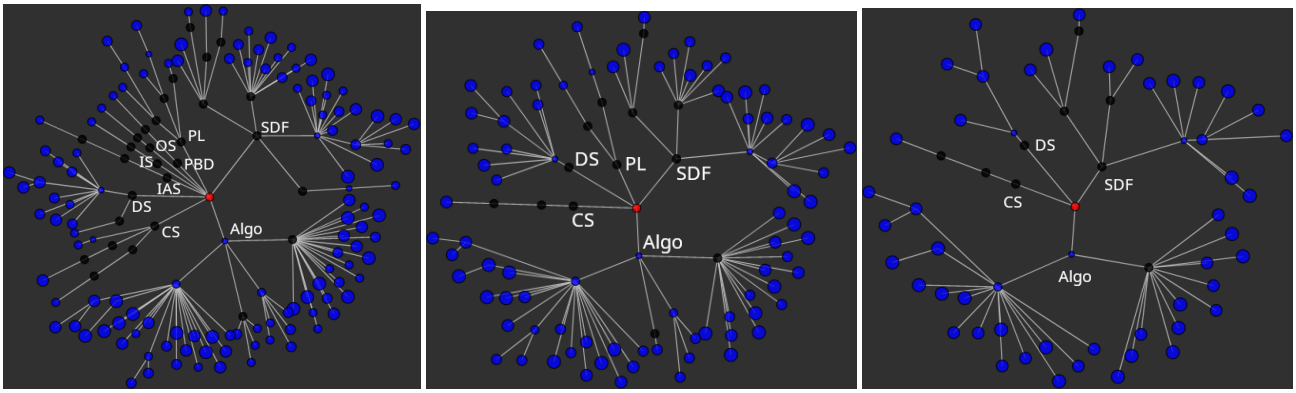
Figure 6 shows tree views of the agreement of topics in Data Structure courses. The curriculum entries that appear in 2 courses or more span many Knowledge areas. By 3 courses or more, the curriculum entries now span 5 Knowledge Areas: Algorithm and Complexity (Algo), Software Development Fundamental (SDF), Discrete Structures (DS), Computational Sciences (CS), and Programming Languages (PL). An agreement at 4 courses or more drops Programming Languages.

The agreement at 4 courses or more spans what most traditionally think of as a Data Structure class: Big-Oh notation and complexity analysis; Basic linear data structures such as arrays, linked list, stacks, and queues; Standard nonlinear data structures such as hash tables, binary search trees and graphs; Traversals of these data structures, including recursion; and basic algorithms related to indexing such as searching and sorting.

4.6 Can we distinguish flavors of Data Structures?

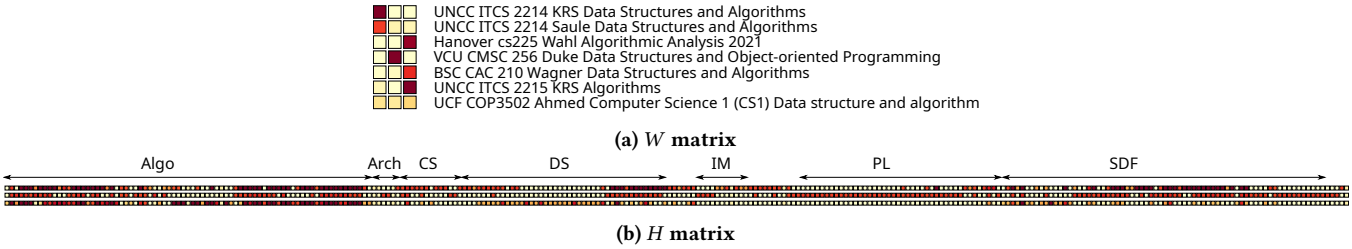
Based on the previous analysis we expect more agreement in the data structure courses than in the CS1 courses. So to try to distinguish flavors of Data Structure courses, we also included in the analysis courses that are only labeled "Algorithms". We performed nonnegative matrix factorization using $k = 2$, $k = 3$, and $k = 4$. Upon inspection, we found that $k = 3$ was leading to the most insight and we present the decomposition in Figure 7.

The three types extracted are more similar than in the CS1 analysis. All three types include what you would think as core data structures: big oh notation, trees, and graphs. Here are the differences.



(a) agreement: 2 courses **(b) agreement: 3 courses** **(c) agreement: 4 courses**

Figure 6: Distribution of agreed upon classification in ACM curriculum for DS courses. Root in red.



(a) W matrix

(b) H matrix

Figure 7: NNMF of Data Structure courses and Algorithm courses. k=3

Type 2 contains a significant amount of Object Oriented Programming topics that fall in the PL and SDF Knowledge Areas. Type 3 has more combinatorial algorithm topics: greedy, dynamic programming, counting, enumerating, and sets. Type 1 seems to have more problem-solving, datasets, APIs, visualization.

While the two courses named directly "algorithms" map in type 3 is not surprising, the BSC course also maps to type 3. VCU's Data structure course maps firmly in type 2. UNCC's 2214 courses are two sections of data structure taught by different instructors (authors of this paper) and both mostly match to type 1, even though the second one has some mapping small matching of the other two types. Interestingly, UCF's course seems to hit all three types evenly.

4.7 PDC courses

Even though we have only 3 PDC courses in our dataset, it seems relevant to discuss them here. We present the agreement tree views with 2 courses or more in Figure 8. Not surprisingly, most of the curriculum entries that 2 courses agree on are in the Parallel and Distributed Computing (PDC) Knowledge Area. There are also common tags in Discrete Structure, Algorithms and Complexity, Systems Fundamental, Software Development Fundamentals, and Programming Languages.

Excluding the entries that directly relate to concurrency or parallelism, there are only a few curriculum entries that are common between 2 or more courses. They are all related to concepts usually taught in CS1 and Data Structures: Directed graphs (as a model of computation), recursion and divide and conquer (for recursive

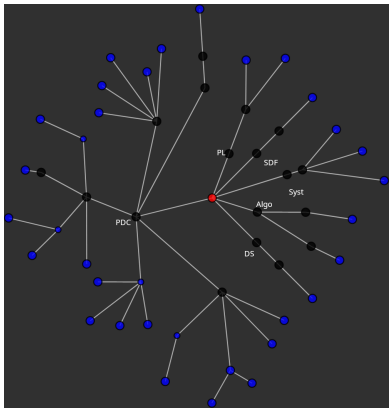


Figure 8: PDC course agreement: 2 courses

task based parallelism), and Big-Oh analysis (for parallel algorithm analysis).

5 DISCUSSION

5.1 Summary of results

We have confirmed that the classification of a class against curriculum guidelines can provide insight in the structure of a class. We have identified different types of CS1 courses: the Object Oriented

Programming ones, Imperative Programming ones, and the ones with algorithmic thinking.

We have identified roughly three types of data structure courses. While they all cover basic data structures, there are some with a focus on object-oriented programming, those with a focus on applications, and those which cover combinatorial algorithms.

5.2 What does it mean for PDC adoption?

A common strategy to improve the preparedness of students for PDC content is to splice some simple topics of Parallel Computing into early CS courses like CS1 and Data Structures.

What our analysis reveals is that there are different types of CS1 and Data Structure courses. As such, it is likely that one size will not fit all. If as a community, we want to create content for these courses, we need to take into account the major differences between these courses.

CS1. The CS1 type 2 courses cover in-memory representation of variables which type 1 and type 3 do not. As such, we could integrate into type 2 courses activities around the importance of order of operations in reduction, which matter for floating point values but do not for integer types. This type of discussion will not integrate well in CS1 of types 1 and 3 which do not appear to cover data representations.

Performing actual operations in parallel only will make sense to students if they can see problems where computation times are high. It is more likely that type 1 courses which include algorithmic thinking and implementation will naturally have programs with longer runtimes and will see the benefit of parallel computing. Probably parallel-for style syntax can be introduced and leveraged for students.

CS1 courses that follow type 3 are fundamentally object-oriented programming courses with little direct algorithmic development. So loop-based parallelization may not be the best approach there. But concurrency can likely be introduced in the sense that operations on two objects may not be strictly ordered for correctness. Maybe a promise-style concurrency management can be successfully deployed there, or a CORBA-style distributed systems programming.

Data Structures. All the courses we reviewed seem to be covering core data structure topics. As such possibly all of them could support discussions of concurrent access to data structures. But in particular the classes of Type 2 with a focus on Object Oriented Programming can certainly cover thread-safe types (and even highlight that it is the primary difference between Java ArrayList and Vector).

Type 3 data structure courses have more discussion of combinatorial algorithm topics. As such they are more likely to feature algorithms with higher runtime which would benefit from parallelism. Brute-force algorithms are perfect for cilk-like parallelism. Also, dynamic programming algorithms are perfect to discuss parallelization strategies since bottom-up parallelism is a good candidate for parallelization using parallel-for constructs. Top-down dynamic programming algorithm poses challenges in parallel since memoization induce complex dependency patterns which can justify a more complex tasking model.

All three types seem to be covering graphs. And as such, they can probably consider the Parallel Task Graph model of parallel codes and as assignments implement topological sorts to derive a feasible order of tasks and compute metrics like critical path to get a sense how parallel the graph is. Implementing a list-scheduling simulator would be a good application of priority queues, and graphs and would fit well in type 1 Data Structure courses.

5.3 Threats to Validity

The number of courses used to understand the course structure is somewhat small and might not accurately reflect the overall trend in CS education. There might be more variants of these courses that might be revealed with a larger pool of courses. The metric for measuring agreement uses references to ACM tags coming from the course materials; however, the depth at which the topic is covered is not taken into account (assumed constant), which might introduce a bias. NNMF can be thought of as a way to reveal data dimensions, but there other dimension reduction techniques, such as PCA, MDS that could be considered.

Also, participants in the workshop classified their materials against the curriculum guidelines that are structured as a tree. It is possible that this tree structure causes the participants to think in terms of the knowledge units of the CS2013 guideline which could bias the raw data collected.

6 CONCLUSIONS

In this paper, we argue that understanding better how early Computer Science is being taught will enable Parallel and Distributed Computing experts to build materials that would help a wide range of instructors integrate PDC into their courses. We ran workshops where participants classified the content of their course against the ACM/IEEE CS curriculum guidelines to get a better understanding of the structure of their course.

The collected data enables us to build models of early CS courses using Non-Negative Matrix Factorization. And we identified 3 main types of CS1 courses and of Data Structure courses. We discussed how that information can help us target PDC content into such courses.

In the future, we would like to expand the collection of courses that are in the system to strengthen the reliability of the analysis and possibly identify more types of courses. We would also like to build better models of courses by investigating other algorithms such as PCA and MDS. Finally, we would like to classify more of the publicly available PDC materials in the system to help recommend PDC materials for particular courses.

ACKNOWLEDGMENTS

This work was supported by grants from the National Science Foundation, Award Nos. OAC-1924057, CCF-1652442.

REFERENCES

[1] 2005. *Modern Multidimensional Scaling: Theory and Applications* (2nd ed.). Springer-Verlag New York. https://doi.org/10.1007/0-387-28981-X

[2] ACM Data Science Task Force. 2019. *Computing Competencies for Undergraduate Data Science Curricula (Draft).* Technical Report. ACM. available at http://www.cs.williams.edu/~andrea/DSTF/index.html.

[3] National Security Agency. 2018. *Centers of Academic Excellence in Cyber Defense (CAE-CD) – 2019 Knowledge Units.* Technical Report. NSA.

[4] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* 3 (March 2003), 993–1022.

[5] B. S. Bloom, M. D. Engelhart, E. J. furst, W. H. Hill, and D. R. Krathwohl. 1956. *Taxonomy of educational objectives: The classification of educational goals. Handbook I: Cognitive domain.* David McKay Company.

[6] College Board. Fall 2014. *Computer Science A: Course Description.* College Board AP. https://apcentral.collegeboard.org/pdf/ap-computer-science-a-course-description.pdf

[7] College Board. Fall 2017. *AP Computer Science Principles, Including the Curriculum Framework.* College Board.

[8] EduHPC. 2018. Peachy Parallel Assignments. http://tcpp.cs.gsu.edu/curriculum/?q=peachy

[9] Alec Goncharow, Matthew Mcquaigue, Erik Saule, Kalpathi Subramanian, Paula Goolkasian, and Jamie Payton. 2021. CS-Materials: A System for Classifying and Analyzing Pedagogical Materials to Improve Adoption of Parallel and Distributed Computing Topics in Early CS Courses. *J. Parallel and Distrib. Comput.* 157 (2021), 316–330. https://doi.org/10.1016/j.jpdc.2021.05.014

[10] Alec Goncharow, Matthew Mcquaigue, Erik Saule, Kalpathi Subramanian, Jamie Payton, and Paula Goolkasian. 2021. Mapping Materials to Curriculum Standards for Design, Alignment, Audit, and Search. 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, 295–301. https://doi.org/10.1145/3408877.3432388

[11] Joint Taskforce on ACM Curricula. 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* ACM/IEEE Computer Society. https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf

[12] Daniel Lee and H. Sebastian Seung. 2000. Algorithms for Non-negative Matrix Factorization. In *Advances in Neural Information Processing Systems*, T. Leen, T. Dietterich, and V. Tresp (Eds.), Vol. 13. MIT Press.

[13] S.J. Matthews. 2023. PDC Unplugged. http://www.pdcunplugged.org/. Accessed, July 2023.

[14] NSF/IEEE-TCPP Curriculum Working Group. 2012. *NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing : Core Topics for Undergraduates.* Technical Report. CDER. available at http://www.cs.gsu.edu/~tcpp/curriculum/sites/default/files/NSF-TCPP-curriculum-version1.pdf.

[15] NSF/IEEE-TCPP Curriculum Working Group. 2020. *NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing : Core Topics for Undergraduates (Version 2.0-beta).* Technical Report. CDER. available at https://tcpp.cs.gsu.edu/curriculum/?q=system/files/TCPP.

[16] Nick Parlante. 2018. Nifty Assignments. http://nifty.stanford.edu/

[17] Erik Saule. 2023. CS Materials. https://cs-materials.herokuapp.com/.

[18] The Joint Taskforce on Computing Curricula: Association for Computing Machinery (ACM), IEEE Computer Society, AAAI. 2023. *CS2023: ACM/IEEE-CS/AAAI Computer Science Curricula.* ACM/IEEE Computer Society/AAAI. https://csed.acm.org/wp-content/uploads/2023/03/Version-Beta-v2.pdf.