Classifying Pedagogical Material to Improve Adoption of Parallel and Distributed Computing Topics

Alec Goncharow, Anna Boekelheide, Matthew Mcquaigue, David Burlinson, Erik Saule, Kalpathi Subramanian
Computer Science
UNC Charlotte
Charlotte, NC, USA
Email: {agoncha1,aboekelh,mmcquaig,dburlins,esaule,krs}@uncc.edu
Jamie Payton
Computer and Information Sciences
Temple University
Philadelphia, PA, USA
Email: payton@temple.edu

***Abstract*—The NSF/IEEE-TCPP Parallel and Distributed Computing curriculum guidelines released in 2012 (PDC12) is an effort to bring more parallel computing education to early computer science courses. It has been moderately successful, with the inclusion of some PDC topics in the ACM/IEEE Computer Science curriculum guidelines in 2013 (CS13) and some coverage of topics in early CS courses in some universities in the U.S. and around the world. A reason often cited for the lack of a broader adoption is the difficulty for instructors who are not already knowledgable in PDC topics to learn how to teach those topics and align their learning objectives with early CS courses.**

There have been attempts at bringing textbook chapters, lecture slides, assignments, and demos to the hands of the instructors of early CS classes. However, the effort required to plow through all the available materials and figure out what is relevant to a particular class is daunting. This paper argues that classifying pedagogical materials against the CS13 guidelines and the PDC12 guidelines can provide the means necessary to reduce the burden of adoption for instructors.

In this paper, we present CAR-CS, a system that can be used to categorize pedagogical materials according to well-known and established curricular guidelines and show that CAR-CS can be leveraged 1) by PDC experts to identify topics for which pedagogical material does not exist and that should be developed, 2) by instructors of early CS courses to find materials that are similar to the one that they use but that also cover PDC topics, 3) by instructors to check the topics that a course currently covers and those it does not cover.

***Keywords*-PDC curriculum; exemplar; classification; adoption of PDC in early CS courses**

I. Introduction

Parallel and Distributed Computing (PDC) has risen as a topic since Dennard scaling ended around 2005 and since Internet systems have become ubiquitous. PDC, however has still not reached most of the classrooms. To remedy this problem, NSF/IEEE-TCPP curriculum guidelines were developed and published in 2012 [1], and a new iteration of these guidelines are expected to be finalized in 2019. Rather than simply proposing to add a PDC course in curriculum with low likelihood of adoption, a more promising strategy is to integrate PDC topics all across the undergraduate curriculum, from early CS courses such as programming and data structures to more advanced courses such as operating systems, and computer architecture. Further, the importance of PDC in Computer Science curriculum was well received and a joint ACM/IEEE group integrated PDC topics in their 2013 Computer Science curriculum guidelines [2] as a dedicated area, but also spread in multiple places in the guidelines.

While at a national level, there is an understanding that PDC topics are of importance, the practical integration of these topics in courses has been slow. Multiple strategies to help with the adoption of PDC integrated in curricula have been deployed with moderate success. Workshops to train instructors are effective at adjusting some courses, but the strategy is not scalable. Some books have been written to explain how to teach these topics to provide some materials [3], [4], but they take time to write, tend to be very specific, and need to be well publicized to reach a wide audience.

Most instructors of CS1/CS2 are somewhat unaware of Parallel and Distributed Computing topics, and typially ask "What topics should I adopt in my class?", and "How do I adopt them in an already packed class?" A set of well developed learning materials (assignments, videos, textbooks, course descriptions, and so on) can provide a scalable answer to such questions. While there are some materials online [5], [6], the questions "How do I find them?" and "Which ones are relevant for me?" remain.

This paper proposes a way to solve this problem by explicitly classifying pedagogical materials (assignments, lecture slides, exams, video lectures, book chapters, etc.)

against well accepted content ontologies. We classify learning materials against two established ontologies, the 2013 ACM/IEEE CS curriculum guidelines [2], and the 2012 NSF/IEEE-TCPP curriculum guidelines for Parallel and Distributed Computing [1]. We have developed CAR-CS, a prototype system to support our classification and demonstrate a scalable, central place of interaction. We have used CAR-CS to classify all Nifty Assignments [7], all Peachy Parallel assignments [5], and the materials used to teach ITCS 3145: Parallel and Distributed Computing at UNC Charlotte [8].

We demonstrate how classifying pedagogical material can help improve the adoption of PDC topics in early CS courses by tackling three different problems: 1) help PDC experts identify topics for which pedagogical material does not exist and that should be developed,. 2) help instructors of early CS courses to find materials that are similar to the ones they use but that also cover PDC topics, and, 3) help instructors to check the topics that their course currently covers and the ones it does not cover (and maybe should or could).

II. Related Work

A. Learning Material Repositories

Nifty Assignments: The Nifty assignments repository [7] 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. Selected assignments are presented at the conference and archived. The selection is primarily based on engagement, adoptability and scalability, and usually targeted at early courses (CS0, CS1, CS2). Nifty assignments now include metadata (topics, difficulty, strengths/weaknesses, dependencies, variants). The Nifty assignments are used as part of the initial assignment set by our CAR-CS system, as they represent a classical (non-PDC) learning materials for early CS courses.

Peachy Parallel Assignments: The Peachy Parallel Assignments [5] are a recent effort of the EduPar and EduHPC [9] workshops to publicize well designed, exciting, and interesting assignments that include some parallel and distributed computing aspects. Peachy assignments focus on adoptability and have been succesfully used in a real classroom. The assignments are peer reviewed and published and presented at EduPar and EduHPC; so far 11 Peachy Parallel Assignments have been presented. The Peachy Parallel Assignments are used as part of the initial set by our CAR-CS system as a representative set of what could be gathered from the PDC educational community.

EngageCSEdu: EngageCSEdu [10], [11] is an NCWIT sponsored repository that provides introductory CS course materials, primarily engaging assignments targeted at CS0, CS1, and CS2. The assignments are categorized by engagement practices to improve student inclusiveness, confidence and broadening participation in computing. The repository has over 800 assignments with a competition for excellence [12] and the assignments submitted are subject to an editorial process with peer review.

Model AI Assignments: Model AI assignments repository [13] is a repository patterned after Nifty assignments (same metadata) for motivating and growing students, educators and practitioners in AI. Assignments submitted for inclusion are peer reviewed and the ones that are accepted are presented at the Educational Advances in AI conference. There are about 40 assignments, going back to 2011.

Data Repositories: The CORGIS data repository [14], [15] is a large collection of tools, datasets and resources that can be used by educators as part of their programming assignments. The datasets range across a large number of disciplines and have been used in introductory courses, such as Computational Thinking [16]. Using real-world datasets can be highly engaging in introductory courses. Real-world applications and dataset have been successfully integrated in Data Structures courses [17], [18]. CAR-CS includes the usage of datasets as a dimension of interest for assignments.

CS in Parallel: CS in Parallel [6] is a repository containing a limited set of learning material that are used to teach parallel computing in various classes. Overall, the repository contains a few documents and organizes them according to the courses they fit into. Classifying against courses is difficult as courses are understood differently in different institutions. We believe it is preferable to classify against well accepted topics in order to enable the percolation of the material in early CS courses by empowering instructors to decide which topics fit best in their classes.

Other Repositories: Other repositories include those surveyed by Decker et al. [19] that also include learning materials for high school teachers, and detail their barrier to entry/participation [20].

Overall, existing repositories tend to only consider early computing education by focusing on courses such as CS0, CS1, and CS2; do not provide classification against well accepted content ontologies; and tend to focus on assignments rather than class materials. In comparison, the CAR-CS system aims to include a wide range of computer science topics and to provide a more expansive, fine-grained classification system that allows for greater expressiveness in assignment search queries.

B. Curriculum Guidelines/Standards

ACM regularly updates computing curriculum guidelines and the latest Computer Science curriculum is from 2013 [2], jointly sponsored by ACM and the IEEE Computer Society (we will denote this guideline CS13). The CS13 guidelines specify a 'redefined body of knowledge, a result of rethinking the essentials necessary for a Computer Science curriculum'. The guidelines also provide numerous exemplars of actual courses and programs that can be adopted by CS departments. In short, the guidelines divide the body of knowledge into a set of *knowledge areas*; knowledge areas

are further divided into into *knowledge units* which contain *topics* and *learning outcomes*. Learning outcomes are classified into three levels, *familiarity*, *usage* and *assessment*. The system we propose adopts the classification proposed by the ACM CS13 curriculum guidelines as a general Computer Science curriculum since it is widely accepted.

The 2012 NSF/IEEE-TCPP curriculum for Parallel Distributed Computing [1] (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 (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 2019. We used the NSF/IEEE-TCPP PDC12 curriculum in our CAR-CS systems as a domain specific curriculum.

Other sub-areas of computing have developed their own standards, such as cyber security [21] and high school CS curriculum [22], [23] which could also be used to provide analyses similar to the one we conduct.

The work that comes closest to our proposed ideas and system presented here is a syllabus repository project [24]. The authors of this work built a repository of syllabi of computing courses by crawling the Internet and classified them against the 2001 Computing Curriula standards [25]. They also proposed a syllabus maker and a comparison tool. Our work differs in that it focuses on course content, specifically on course materials, and ensures that the course is closely tied to a given standard. Furthermore, CAR-CS provides interactive tools to assess instructor materials and gauge their coverage in the context of their curriculum, and is highly extensible.

III. The CAR-CS system

A. Goals

Instructors are often looking for inspiration for new lectures, problem sets, and exercises that align with computing education curriculum standards (e.g., ACM Curriculum Guidelines, ABET standards) and that adequately address the learning objectives of their courses. However, it is difficult for educators to use traditional search tools to find existing problems and learning materials that may be useful in their own computer science courses. The lack of centralization of materials certainly presents a problem, but the most significant limitation is the lack of meaningful, searchable features that capture what makes a material useful for a given computer science course. Labeling by course title, as is done for most existing collections of course assignments and material [6], [7], is too simple a description. Course content vary widely across institutions, for instance, not everyone agrees on what CS1 should cover. One university may define a CS1 course for a quarter system while another creates a semester-long course. The choice of programming language can result in different specifications (and interesting twists!) in assignments in introductory courses. In addition, instructors are often looking for new ways to create assignments that promote student engagement and are relevant to a diverse student population. Some instructors may use a particular pedagogical approach, instructional technique, or running theme for assignments. For example, one instructor may focus on a media computation approach while others may want to use social media datasets.

Our approach to developing the Compelling Assignment Repository for Computer Science (CAR-CS) is intended to address the need for more meaningful searches of computer science course materials. CAR-CS pairs materials with properly curated metadata to create a more fine-grained structured representation of materials for search. CAR-CS uses classic material descriptors, such as course-level, programming language, and datasets. More importantly, CAR-CS also *relates materials to curriculum ontologies*. As a starting point, we build on the ACM Computer Science 2013 curriculum guidelines [2] and the NSF/IEEE-TCPP Parallel and Distributed Computing 2012 curriculum [1] to extract more meaningful, discipline-specific, fine-grained features to describe each material. Specifically, each material will be associated with the topics covered by the assignment and the learning outcomes that the assignment fulfills. Note that while we use these particular set of guidelines to identify requirements for and to populate an initial version of CAR-CS, other guidelines and standards (such as the ones mentioned in Section II-B) could be integrated in the system.

Mapping material to curriculum guidelines and other descriptive features opens up several opportunities for new search functionalities. For instance, one can explicitly filter against a group of features that is of interest to an instructor looking for material, or look for similarities to an existing material, and perhaps, to create variants of an existing material. It enables one to ask questions pertinent to a material or to a course, or to understand how a topic or a learning outcome is typically covered.

While the CAR-CS system is currently a proof-of-concept prototype, a fully fledged system will, in fact, be useful to instructors, provided the issue of material curation is addressed. We believe that a crowdsourced model can be used to address the need for curation. With such an approach, *instructors* can upload their own material in the system and a number of *editors* can review the uploaded materials. An editor has experience or credentials demonstrating knowledge of the standards used by the system, and can appropriately edit or fix classification issues with a submitted

material. Less knowledgeable *users* can suggest changes to the metadata which must be verified by an *editor*.

B. Design

The CAR-CS prototype system is built as a web service hosted on Heroku. Our current implementation uses the ACM CS13 Curriculum Guidelines [2] to classify material and the NSF/IEEE-TCPP PDC12 Curriculum Guidelines [1]. The data is modeled relationally and is stored in a postgreSQL database. A Django web server provides a RESTful API to the service and serves webpages to provide the main interaction with the service. Webpages are made dynamic by the use of JavaScript, the system supports dynamic queries thanks to the jQuery library that enables asynchronous communication with the RESTful back end. Interactivity and visualization are provided by the D3 JavaScript library [26].

In the database, each assignment is associated with a title, authors, URL and description. The classifications are usually hierarchical and therefore they are represented with a key, the key of the parent, a string description, and type (separating topics and learning outcomes). Tags, items in the classification, dataset used, and authors are associated with an assignment using a many-to-many relationship. The PDC12 and CS13 classifications are included in the system. Note that the ACM CS13 classification is completely hierarchical while there are "cross-cutting" topics in the PDC12 topic. However these topics in PDC12 are actually listed as a separate category and organized hierarchically. The model could be extended if the classifications were DAGs instead of trees.

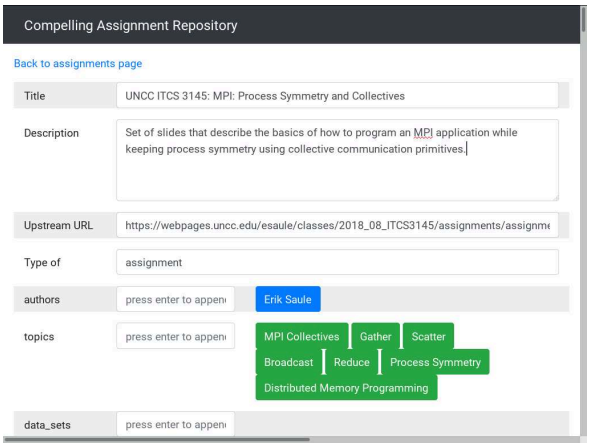
The system has been seeded using the Nifty assignments [7] which serve as a set of non-PDC material. We included all assignments from 2003 to 2018 and we excluded assignments for which links were broken. The authors served as both contributors and editors and entered about 65 Nifty assignments. We have also included all 11 Peachy Assignments [5]. And we have entered all of the learning materials from the class ITCS 3145: Parallel and Distributed Computing taught at UNC Charlotte [8]. The materials of class consist of lecture slides and scaffolded assignments on parallel algorithms to be implemented on shared memory systems (pthreads, OpenMP) and distributed memory systems (MPI and MapReduce-MPI [27]).

IV. Use cases

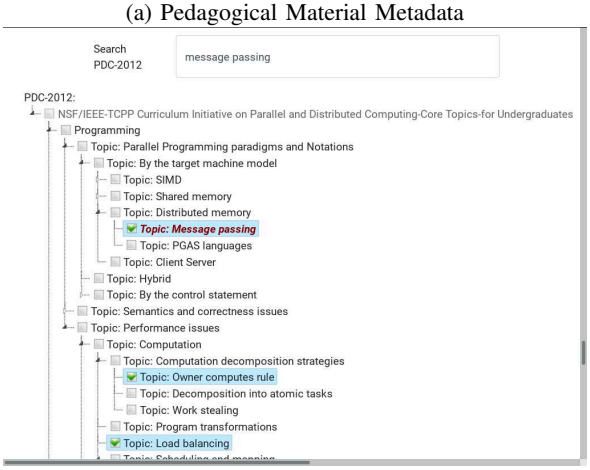
A. Entering new pedagogical material

The simplest task that one would want to perform on the CAR-CS system is to add a new pedagogical material. This process is relatively simple. A form guides the user to filling out basic information (title, authors, short description, URL, etc.). This information is used to build most of the meta data of the material, as seen in Figure 1a.

The mapping of a material to a classification ontology is done using a tree list which can be seen in Figure 1b. Nodes



(a) Pedagogical Material Metadata



(b) Editing Classification

Figure 1: Adding and Classifying Materials. (Online at https://cs-materials.herokuapp.com/assignments)

of the tree can be selected to indicate that the particular topic is covered by the material. The mappings that are selected can be viewed at the bottom of the material description. Entries can be searched for by entering a word or phrase that becomes highlighted in the classification.

We classified about 65 Nifty assignments, all 11 Peachy assignments, and all the materials from the class, ITCS 3145:Parallel and Distributed Computing, taught at UNC Charlotte [8]. That class is composed of 12 slide decks and 9 assignments. Inputting (including classifying) all the material took the instructor of that class (one of the authors, Dr. Saule) about a day of work, with each item taking between 15-25 minutes to input and classify. The classification of these three classes of material can be seen in Figure 2.

Keying the meta data is straightforward and fast, but classification is more involved, because of the size of the ontologies (the CS13 classification contains about 3000 entries). One could quickly make some selection but most likely doing so would miss relevant entries. For instance, in CS13,

parallelism related topics appear in three different places: *System Fundamental*, *Computational Science::Processing*, and in *Parallel and Distributed Computing*. Going quickly through the classification would most likely get a poor classification of the material.

In PDC12, some odd placement also happens; for instance, Amdhal's law (and related topics) falls under *Programming::Performance Issue::Data*. *Algorithm::Model based notions::Parallel and Distributed Models and Complexity::Notions from scheduling* misses Critical Path. The Map-Reduce programming model seems mostly missing. (There are entries for BSP; which is oddly bundled with Cilk; and Cloud Computing but these are not quite the same). Overall, the PDC12 guideline was a first attempt at classifying PDC topics, and certainly the 2019 edition of PDC is expected to correct these oddities.

In both classifications, topics related to middleware (design and implementation) seem to be mostly missing. Runtime systems appear under Programming Languages in CS13, but refer to different things. Also on many topics, both classifications seem to stay at a high level. While this is appropriate for curriculum guidelines, it is not as precise for classification of material as one would hope for. For instance, CS13 has an entry for Task-Based Decompositions, but recursive Cilk-style decomposition are different from OpenMP depends-style decomposition.

Also it seems that it could be useful not only to say that a material matches a topic, but at which level the topic is matched. Taking an example, an early assignment in ITCS 3145 was to implement a numerical integrator using the rectangle method. Naturally this assignment checks *Computational Science::Numerical Analysis::Numerical differentiation and integration*. But the assignment only covers integration and only requires the students to implement a single method from a provided formula. A numerical methods course would have a more comprehensive lecture on numerical integration and would check the box in the same way. Since both CS13 and PDC12 guidelines have incorporated Bloom levels, it would make sense to classify materials with Bloom levels as well.

Overall the process of classifying an assignment was found to be enlightening, and puts in perspective what the material does and does not do. While the classification is time consuming, we believe that crowdsourcing the classification (by making trusted users editors) would lead to accurate classification. Also the time required to classify materials decreases once the classifier understands the ontologies. We envision that once enough materials are classified, we would be able to leverage existing classification to provide recommendation on topics commonly used together.

B. Coverage of a Class

Using these classifications, one can easily investigate the coverage of a class. Consider the PDC classification of the class ITCS 3145 (shown in Figure 2f). Most of the classified topics falls in the *Programming* category, followed by the *Algorithm* category; the *Architecture* and *Cross Cutting and Advanced* categories are mostly left untouched. This is expected since the class focuses on programming and achieving speedup using shared and distributed memory computing by taking a dependency graph and scheduling approach rather than a performance and hardware approach to the class.

One can notice that topics related to distributed systems, complexity theory, complex algorithms, and tooling are not covered by the class. While the absence of most of these is by design, the absence of tools from the class is an omission of the instructor.

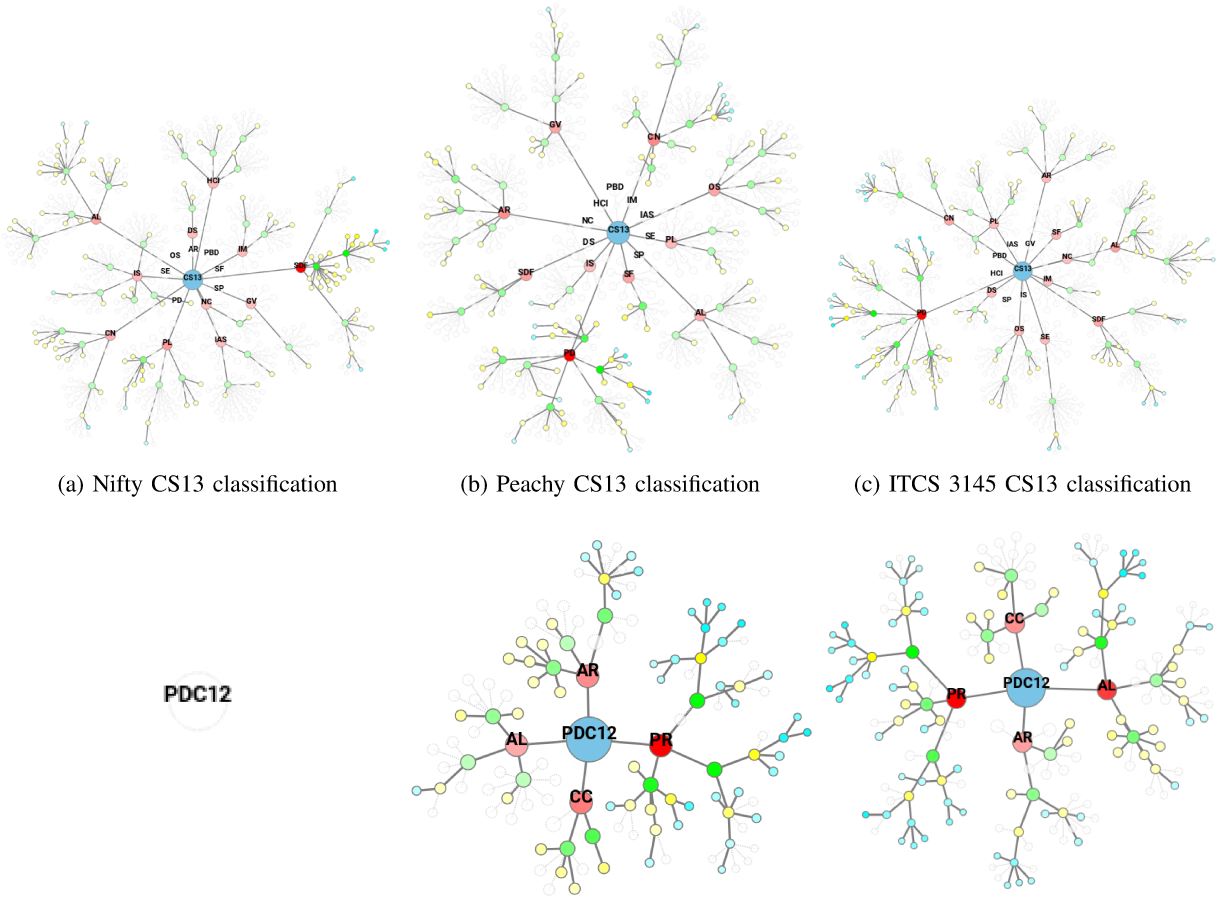
Looking at the coverage of ITCS 3145 in the CS13 curriculum (shown in Figure 2c) highlights that *Parallel and Distributed Computing* is the most covered area. This is expected for this course and the classification shows the previously reported gaps: parallel architecture and distributed systems are mostly left uncovered.

Algorithm and Complexity is the second most covered area by the class, which is consistent with the perspective provided by the PDC classification. It comes from the reliance on complexity notation, parallel task graph analysis, and some classic algorithms used as examples.

The third area highlighted by the analysis is *Computational Sciences*. This comes from the usage of some stencil based algorithms and numerical integration in a few assignments and because *Fundamental Parallel Computing* is an area of *Computational Sciences::Processing*.

Software Development Fundamental is the fourth area highlighted. Indeed basic programming constructs are dissected with a parallel twist and assignments are scaffolded using unit tests which appears in that category. The other partially covered areas from this course are expected: *Operating Systems*, *Programming Languages*, and *Architecture*.

The take home message of this analysis is the realization that some topics could have been touched by this class, but were not. We discussed that tools for parallel computing were not covered by the class and that it was highlighted by the PDC12 classification. But the mapping to the CS13 curriculum highlights non-PDC areas that are touched upon as side notes by the class such as unit tests and numerical integration. Some of the untouched areas like *Human Computer Interactions*, *Social Issues*, *Information Assurance and Security*, or *Platform Based Development* are not surprising. But the absence of mapping to *Graphics and Visualization* and *Intelligent Systems* reveals that the class could be made more engaging by having some assignments or examples derived from these areas.



(a) Nifty CS13 classification (b) Peachy CS13 classification (c) ITCS 3145 CS13 classification

(d) Nifty PDC12 classification (e) Peachy PDC12 classification (f) ITCS 3145 PDC12 classification

Figure 2: The three dataset classified against the PDC12 and CS13 ontologies. (Read-only version at https://cs-materials.herokuapp.com/coverage.) The classification are shown as a tree where the root is the name of the ontology. First level nodes are tagged with the 2 or 3 letter code that represent that section of the ontology. The color intensity of the node is proportional to the number of material that matches that entry of the ontology. The color palette is different for zeroth, first, and more-than-first level nodes. Ontology entry absent from the materials are transparent and their children are not included.

C. *Identifying gaps in existing offering*

Classification helps PDC educational experts identify where more efforts are needed to improve adoption of the Parallel and Distributed Computing curriculum. Consider the structure of the classification Nifty assignments (in Figures 2a and 2d) and Peachy Parallel assignments (in Figures 2b and 2e) as an example.

Clearly Nifty Assignments do not cover any PDC topics while Peachy Assignments do. The most common area of the CS curriculum covered by Nifty is *Software Development Fundamental*, followed by *Programming Language*, *Algorithms and Complexity*, and *Computational Sciences*. Unsurprisingly, the first CS13 curriculum topic of Peachy assignments is *Parallel and Distributed Computing*; the following ones are in Systems, and Architecture areas. *Software Development Fundamentals* (SDF) is low on the list; and topics in SDF covered by Peachy assignments relate to *Fundamental Programming Concepts* (variable, loops) and leave aside *Fundamental Data Structures*. Nifty Assignments seem to commonly touch upon Object Oriented Programming which does not appear in Peachy Assignments.

The take home message is that while Nifty Assignments and Peachy Assignments may have some commonalities, unless the PDC community develops assignments that align better with classic CS1-CS2 assignments, it is unlikely we will see massive adoption. Using standard classification is a way to measure the alignment between different communities and set of assignments.

D. *Finding Reference Material for Integrating PDC in Early Courses*

One of the key goals of classifying learning materials is to enable an instructor to find new material that covers the

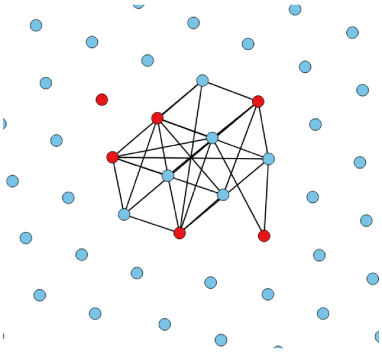


Figure 3: Similarity between Nifty Assignments (in blue) and Peachy assignments (in red). (Online version at https://cs-materials.herokuapp.com/similarity.)

same topics than what s/he is already using but that also contains PDC topics. A typical case would be to try to replace a lecture on looping construct with one that includes your classical *while* and *for* loops, but that also includes discussion of parallel loops (e.g., OpenMP *for* loop in a C programming lecture).

One of the ways we envision this happening in practice is once a class with all lectures and assignments have been keyed in, the system would identify slides, assignments that have similar classification to those in the class. Showing that the system helps in a real use case would be the ideal proof of concept. However we have not at this time performed that experiment on a real non-PDC class.

We show the potential of the system by showing that it is possible to extract meaningful similarities between a set of non-PDC assignments (the Nifty Assignments) and PDC assignments (the Peachy Assignments). Figure 3 presents a similarity graph between all Nifty Assignments and all Peachy Assignments. Blue circles represent Nifty assignments while red circles represent Peachy assignments. A Nifty assignment and a Peachy assignment are said to be similar if they share two classification items and this similarity is represented by an edge. The graph shows that most assignments have no similar assignment in the other set. This is expected since (as we reported above) many of the Nifty assignments are object oriented programming assignments while none of the Peachy Parallel assignments are. The Peachy assignments that do not match any other Nifty assignments are the ones that are systems oriented, such as dealing with middleware, or data races.

Yet, the system was able to identify some related Peachy and Nifty assignments. They essentially form a cluster because all the assignments share the classifications *Arrays* and *Conditional and iterative control structure*. The Peachy assignments "Computing a movie of zooming into a fractal," "Fire simulator and fractal," "Using a monte carlo pattern to simulate a forest fire," "Storm of high energy particles" are matched to Nifty assignments "Hurricane Tracker," "2048 in python," "Campus shuttle," "Nbody simulation," "Image editor," and "Uno." While some of the matches may not be perfect drop-in replacements in a particular class, one can see that these assignments could replace one another in a proper context.

V. Conclusion

Cloud systems, clusters, IoT devices, GPU acceleration, and multicore system are now standard components of the computing landscape. Yet CS students are usually taught Parallel and Distributed Computing only in a late elective in their junior or senior year. There have been efforts to remedy the problem, including a proposal for a Parallel and Distributed Computing curriculum guidelines in early CS education by an NSF/IEEE-TCPP group (PDC12) and the inclusion of PDC topics in the most recent ACM/IEEE Computer Science curriculum guideline (CS13). Despite these efforts, the adoption of PDC topics in early CS courses remains limited. A commonly cited reason is that instructors have difficulties finding relevant material to teach and learn these topics.

In this paper, we recommend classifying learning materials against standard ontologies such as PDC12 and CS13 and curating them in a single system provides many advantages for multiple actors. We have demonstrated this by building a prototype system and classifying materials from three sources: Nifty Assignments, Peachy Parallel Assignments, and the material used in a Parallel and Distributed Computing class. The prototype has been shown to benefit different groups of users by answering different questions. An instructor can search for materials on precise topics. A course instructor can get a better sense of the coverage of topics of his/her course. Any one performing the classification will get a better grasp of what the classification contains. PDC education experts can notice patterns in class materials that are not represented by existing or available PDC materials.

The Peachy Parallel Assignment set is still young but has been shown to be promising. Hopefully, by knowing what non-PDC assignments looks like, the community should be able to develop additional Peachy Parallel Assignments that will reach and impact a wider audience. Also, the methods presented in this paper are not restricted to Parallel and Distributed Computing topics, and could be adapted to other fields such as Computer Security, which suffers from similar issues.

The system that we developed is a prototype that will need to be fully built before being useful. In particular, a proper user account system, and roles (editor, submitter, user) need to be integrated to enable a larger scale curation of the material and wider adoption. Finally, we note that classifying materials is a time-consuming process. We believe that community experts can play the roles of editors to help with the classification process. Also, once more material is classified using the system, we should be able to suggest classifications to save time for the user.

Acknowledgments

The authors would also like to thanks Nick Parlante for his work on curating the Nifty assignments and David Bunde for his work in curating the Peachy Parallel assignments.

This work is supported by grants from the National Science Foundation (CCF-1652442, DUE-1245841, and DUE-1726809) and by a summer undergraduate research fellowship from the Charlotte Research Scholars Program.

References

[1] NSF/IEEE-TCPP Curriculum Working Group, "NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing : Core topics for undergraduates," CDER, Tech. Rep., 2012, available at http://www.cs.gsu.edu/~tcpp/curriculum/sites/default/files/NSF-TCPP-curriculum-version1.pdf.

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

[3] S. Prasad, A. Gupta, A. Rosenberg, A. Sussman, and C. Weems, Eds., *Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses*. Morgan Kaufmann,, 2015.

[4] ——, *Topics in Parallel and Distributed Computing: Enhancing the Undergraduate Curriculum: Performance, Concurrency, and Programming on Modern Platforms*. Springer International Publishing, 2018.

[5] "Peachy parallel assignments," https://grid.cs.gsu.edu/ tcpp/curriculum/?q=peachy.

[6] R. Brown, L. Shoop, and J. Adams, "CS in parallel," https://csinparallel.org/.

[7] N. Parlante, "Nifty assignments," 2018. [Online]. Available: http://nifty.stanford.edu/

[8] E. Saule, "Experiences on teaching parallel and distributed computing for undergraduates," in *Proc of IPDPSW 2018*, May 2018.

[9] E. Ayguade, LlucAlvarez, F. Banchelli, M. Burtscher, A. Gonzalez-Escribano, J. Gutierrez, D. A. Joiner, D. Kaeluu, F. Previlon, E. Rodriguez-Gutiez, and D. P. Bunde, "Peachy parallel assignments (EduHPC 2018)," in *Proc. of EduHPC*, 2018.

[10] A. Monge, B. A. Quinn, and C. L. Fadjo, "EngageCSEdu: CS1 and CS2 materials for engaging and retaining undergraduate CS students," in *Proc. of ACM SIGCSE*, 2015, pp. 271–271.

[11] NCWIT, 2018. [Online]. Available: https://www.engage-csedu.org/

[12] G. Sprint and A. O'Fallon, "Engaging programming assignments to recruit and retain CS0 students: (abstract only)," in *Proc. of ACM SIGCSE*, 2018, pp. 1093–1093.

[13] AAAI, "Model AI assignments," 2018. [Online]. Available: http://modelai.gettysburg.edu/

[14] A. C. Bart, "CORGIS Datasets Project: The Collection of Really Great, Interesting, Situated Datasets," 2016, https://think.cs.vt.edu/corgis/.

[15] A. Bart, E. Tilevich, S. Hall, T. Allevato, and C. Shaffer, "Transforming introductory computer science projects via real-time web data," in *Proc. of ACM SIGCSE*, 2014, pp. 289–294.

[16] A. C. Bart, R. Whitcomb, D. Kafura, C. A. Shaffer, and E. Tilevich, "Computing with CORGIS: Diverse, real-world datasets for introductory computing," *ACM Inroads*, vol. 8, no. 2, pp. 66–72, Mar. 2017.

[17] D. Burlinson, M. Mehedint, C. Grafer, K. Subramanian, J. Payton, P. Goolkasian, M. Youngblood, and R. Kosara, "BRIDGES: A system to enable creation of engaging data structures assignments with real-world data and visualizations," in *Proc. of ACM SIGCSE 2016*, 2016, pp. 18–23.

[18] K. Subramanian, "BRIDGES (Bridging Real-world Infrastructure Designed to Goal-align, Engage, and Stimulate),," 2018. [Online]. Available: http://bridgesuncc.github.io/

[19] A. Decker, M. M. McGill, L. A. DeLyser, B. Quinn, M. Berry, K. Haynie, and T. McKlin, "Repositories you shouldn't be living without," in *Proc. of ACM SIGCSE*, 2018, pp. 920–921.

[20] M. Leake and C. M. Lewis, "Recommendations for designing CS resource sharing sites for all teachers," in *Proc. of ACM SIGCSE*, ser. SIGCSE '17, 2017, pp. 357–362.

[21] N. S. Agency, "Centers of academic excellence in cyber defense (CAE-CD) – 2019 knowledge units," NSA, Tech. Rep., 2018.

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

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

[24] M. Tungare, X. Yu, W. Cameron, G. Teng, M. A. Pérez-Quiñones, L. Cassel, W. Fan, and E. A. Fox, "Towards a syllabus repository for computer science courses," in *Proc. of ACM SIGCSE*, ser. SIGCSE '07, 2007, pp. 55–59.

[25] Joint Taskforce on Computing Curricula, *Computing Curricula 2001 Computer Science*. ACM/IEEE Computer Society, 2001. [Online]. Available: http://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2001.pdf

[26] D3: Data Driven Documents, 2018. [Online]. Available: https://d3js.org/

[27] S. J. Plimpton and K. D. Devine, "MapReduce in MPI for large-scale graph algorithms," *Parallel Computing (ParCo)*, vol. 37, no. 9, pp. 610–632, Sep. 2011.