Independent Study - OS 2970
Course-Adaptive Content Recommender For Course Authoring

Hung Chau

School of Computing and Information, University of Pittsburgh, Pittsburgh, PA, USA
hkc6@pitt.edu

Abstract. Developing online courses is a complex and time-consuming process that involves organizing a course into a sequence of topics and allocating the appropriate learning content within each topic. This task is especially difficult in complex domains like programming, due to the incremental nature of programming knowledge, where new topics extensively build upon domain concepts that were introduced in earlier lessons. In this paper, we propose a course-adaptive content-based recommender system that assists course authors and instructors in selecting the most relevant learning material for each course topic. The recommender system adapts to the deep prerequisite structure of the course as envisioned by a specific instructor, while unobtrusively deducing that structure from problem-solving examples that the instructor uses to present course concepts. We assessed the quality of recommendations and examined several aspects of the recommendation process by using three datasets collected from two different courses. While the recommender system explored in this paper was built for the domain of introductory programming, our course-adaptive recommendation approach could be used in a variety of other domains.

Keywords: Course authoring; learning content recommendation; course model

1 Introduction

Over the past twenty years, most intelligent tutoring systems (ITSs) have focused their personalization efforts on helping students find an "optimal path" through available learning content to achieve their learning goals. A range of personalization technologies, known as course sequencing, adaptive navigation support, and content recommendation, can account for the learning goals and the current state of student knowledge and recommend the most appropriate content (e.g., a problem, an example, an educational video, etc.). However, in the context of real courses, there is not complete freedom in selecting the appropriate content for students. An instructor usually plans a course as a sequence of topics to be learned. To stay in sync with the instructor and the class, students are expected to work on course topics in the order that is determined by the

instructor's plan. In this context, the personalized selection of learning content should account for both a student's prospects (i.e., current knowledge levels) and the instructor's prospects (the preferred order of topics and learning goals).

Unfortunately, the current generation of ITSs rarely support adaptation to a teacher's preferences. In most of these systems, a sequence of topics is predefined and learning content items are statically assigned to these topics. While this approach works well for instructors who are happy to follow the sequence of topics that is defined by the ITS, the instructors who prefer a different topic structure will find such a system unacceptable, since it doesn't support their approach to teaching the course. These considerations are especially important when learning programming, where almost every instructor and every textbook introduces a unique course organization [1, 2].

Nowadays, a variety of learning content items could be accessed from different learning content repositories and portals [3, 4], while the majority of learning management systems offer authoring tools to structure a course into a set of topics and to add learning content to each topic. However, our work with instructors revealed that limited assistance provided by the current course authoring tool is not sufficient. While defining a sequence of topics is an easy task, selecting the most relevant content for each topic from a large collection of advanced learning content items is a real challenge. The instructors need to carefully review a large number of problems and examples in order to select those that best fit their learning goals for the topic. This is a time-consuming and error-prone process [5–7]. While a number of recommender systems have been developed to assist instructors in finding relevant content in online repositories [8], these systems attempt to adapt to the overall goals and interests of their users and are not able to consider the complex prerequisite-based structures of modern courses.

This paper presents *Content Wizard*, a content recommender system that has been specifically created to assist instructors with the course authoring process by recommending learning activities that are most appropriate to each of the course topics in the context of their preferred model of the course. The system leverages two valuable resources provided by instructors: *the order of course topics* and *problem-solving examples*, which instructors (or textbook authors) present to students to demonstrate course concepts.

This paper is organized as follows: Section 2 presents a brief review of related work, while Section 3 discusses the place of content recommendation in a course-authoring context and presents the interface of Content Wizard. The internal organization of the system and its recommending approach is described in Section 4. Section 5 presents an evaluation of Content Wizard's performance against a more traditional baseline, and Section 6 examines the performance of the approach on a deeper level. Finally, Section 7 presents a discussion of results and possible avenues for future work.

2 Related Work

The problem of authoring support in an ITS context has been extensively explored. Murray [5, 6] defines seven categories of ITS authoring tools and generally classifies them into two broad groups: pedagogy-oriented systems or performance-oriented systems. *Performance-oriented systems* focus on providing a rich educational environment, in which students can gain problem-solving expertise, procedural skills, concepts, and facts by practicing and receiving feedback and guidance from tutors. Authoring tools in this group include simulation-based learning, domain expert systems, and some special purpose systems. The prominent examples in this category are ASPIRE and cognitive tutor authoring tools (CTATs). ASPIRE [9] allows non-computer scientists to develop new constraint-based tutors with main support for generating a domain model and producing a fully functional system. Cognitive tutor authoring tools (CTATs) [10] allow authors to develop two types of tutors: cognitive tutors and example tracing tutors. The CTAT authoring process requires authors to give a definition of a task domain (such as the fraction addition problem), along with appropriate problems. *Pedagogy-oriented systems* focus on organizing instructional units and tutoring strategies. They support instructors in managing curriculum sequencing and planning, designing teaching strategies and tactics, composing multiple knowledge types (e.g., topics and concepts), and authoring adaptive hypermedia. Two examples in this category are InterBook and SitPed. InterBook [11] provides support for authoring adaptive electronic textbooks. It helps authors to create the book's structure and associate every section to domain concepts. SitPed [12] is a pedagogy-oriented authoring system that supports instructors in creating simple, hierarchical task models, authoring assessment knowledge, and creating tutor feedback and guidance.

Authoring tools in the pedagogy-oriented group frequently focus on supporting authors by defining the domain model as a set of knowledge components (concepts or rules), building a course structure, and associating course units and learning content with domain concepts [11, 13–15]. While our work follows the same approach, we minimize authors' load by deriving the intended course structure from easily available data, rather than requiring the authors to manually provide their intended course structure. The most recent systems in this group also offer learning content recommendation for course authors [8]; yet in most cases, content recommendation is based on instructor interests or on specific goals, and less than a handful of projects [16] focused on using the whole course structure for content recommendation. Our work attempts to advance this research direction by exploring a more powerful yet unobtrusive recommendation approach and by offering a more extensive evaluation than earlier efforts.

3 Content Recommendation for Course Authoring

The content recommendation approach presented in this paper was developed for a typical course authoring context. The essence of this scenario is that the

author designs the course structure as a sequence of *topics*. To support learning for each topic, a set of items of multiple content types is associated with each topic. This course structuring approach is supported by every major learning management system (where a topic usually corresponds to a course lecture), as well as by most textbooks (where a topic usually corresponds to a chapter).

To facilitate this course-authoring approach and to make it easier for instructors to reuse large volumes of "smart" learning content for computer science education [18], we developed a drag-and-drop course authoring tool (Fig. 1). The tool allows course authors to define a sequence of topics (shown on the left), select their preferred types of learning activities (the authors could use over 15 types in three domains), and add the desired content to each topic. To add content to a topic, the author selects one topic and one type of learning content (in Fig. 1, the topic *Variables* and the *Problems* content type is selected). Following that, the authoring system shows a list of all activities of the selected type that could be added to the topic through a drag-and-drop interface. The key problem here, as we discovered when working with an early version of the tool, is that this interface doesn't really help the authors to select precisely the right kind of content for each topic. This task is quite difficult: each selected item should cover the learning concepts that the instructor wants to introduce in the selected topic, but at the same time, it should not use concepts that students have not learned yet. The sheer amount of content to consider (e.g., 289 *interactive examples* and 223 *programming problems*) makes this task practically impossible without additional support.

The goal of Content Wizard is to provide the necessary support to make content selection both feasible and efficient. As the right side of Fig. 1 shows, Content Wizard ranks all content items of the selected type by its match to the selected topic in the course context. It also assigns "star" relevance ratings to all content items and puts a warning sign to items that, while superficially a good match, might include concepts that have not yet been introduced at this point in the course. The most important aspects of support offered by Content Wizard are: (1) this support is based on Content Wizard's understanding of the fine-grained course structure and prerequisite relationships on the level of domain concepts; (2) the instructor is not expected to define the fine-grained course structure, as required by some earlier approaches [13, 14]; instead, the course's structure is automatically derived from the order of course topics and the set of code examples that instructor shows at the target lecture (or provides in a book chapter). The next section explains this approach in detail.

4 Course-Adaptive Content Recommender System

4.1 The course model

The content recommendation that is provided by Content Wizard is based on a deeper-level course structure modeling. While the instructor may perceive the course as a sequence of topics, the deeper model assumes that the goal of each

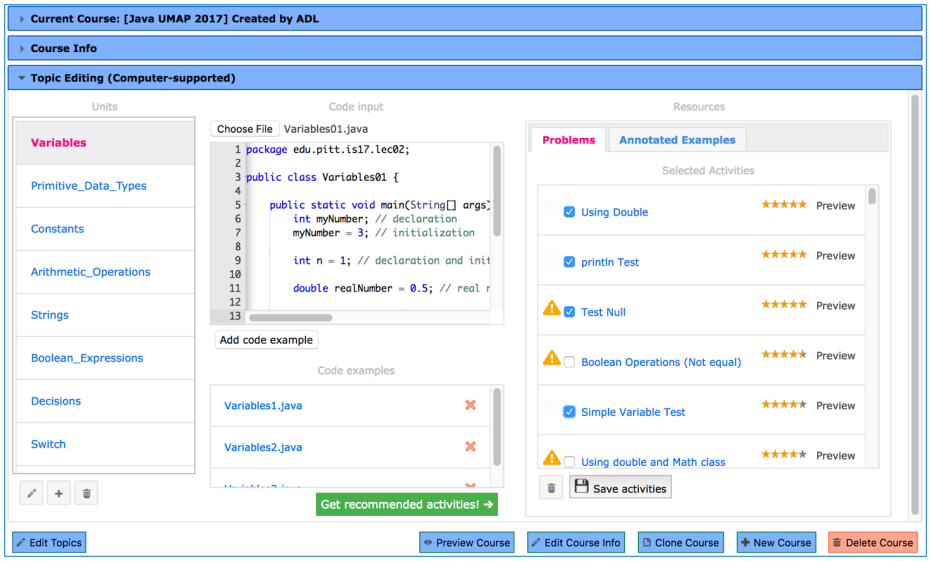


Fig. 1: Course-authoring tool with Content Wizard recommendations

topic is to introduce a set of fine-grained domain concepts (knowledge components). The course model is defined as a sequence of concept sets that are covered throughout the course (see Figure 2). The concepts associated with a specific course unit are the concepts that instructor aims *to introduce at that unit*. For example, Unit 2 is the first unit of the course where the concepts *Array Variable* and *Array Data Type* appear, among others. The concepts introduced in earlier units become prerequisite concepts for later units. For example, *Print*, *Int Data Type*, and other Unit 1 concepts are expected to be learned before students start Unit 2, and are considered prerequisites for Unit 2 and all following units. This deeper level concept-based course modeling is popular among ITS and Adaptive Hypermedia authoring systems [11, 13, 14] where it is assumed that course or system authors will create this model manually. The difficulty of manual modeling is a known bottleneck of the fine-grain course structuring approach, which prevents this approach from being more broadly used. However, Content Wizard is able to automatically derive this model by using worked examples that are provided by the course authors.

4.2 Worked code examples

Worked examples, in the form of complete programs or code fragments, are extensively used in teaching programming concepts. In each lecture, an instructor usually presents several worked examples that illustrate newly introduced concepts. Similarly, each programming textbook extensively uses examples and frequently offers access to the code of these examples through a CD included with the textbook or a Web site. The assumption behind the Course Wizard's automatic course structuring is that a set of examples presented for each unit

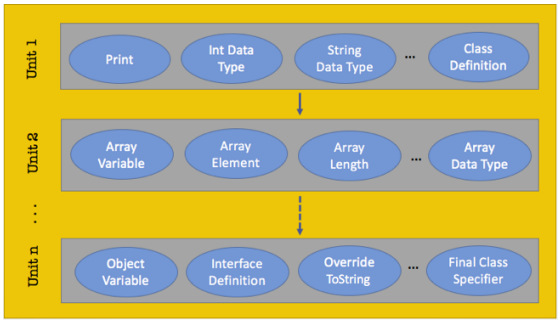


Fig. 2: Knowledge structure of a course

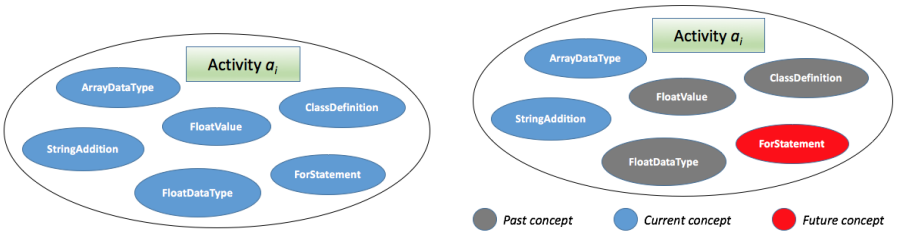
offers the best way to understand the concepts that the instructors want to introduce in this unit. To build a deep course model that follows the instructor's preferences, Content Wizard asks the course author to submit the plain code of each example that is used in the unit (see the middle column in Fig. 1).

Using the code examples provided for each unit, Content Wizard automatically creates the knowledge structure of the course, as shown in Fig. 2. First, it extracts all the programming concepts that are associated with each code example. This extraction is performed by using a Java concept parser [17], that returns a set of fine-grained concepts from Java ontology. [1]. Second, for each unit, it forms a set of *covered concepts* that merges concepts from all of the unit's examples. Finally, it sequentially processes the units to define the unit's content as concepts that are first introduced in this unit; i.e., all concepts extracted from Unit 1 examples become Unit 1 concepts; all *new concepts* extracted from Unit 2 examples (i.e., those that have not been introduced in Unit 1) become Unit 2 concepts, and so on.

4.3 Content representation and analysis

To identify a match between a unit and a learning activity, Content Wizard considers a set of concepts associated with a candidate activity and the course structure. Since all types of of learning activities available in the system (i.e., examples or problems) include code fragments, we use the Java concept parser [17] to represent each activity as a "bag" of Java programming concepts (Figure 3a). This "bag of concepts" representation could be used by a number of traditional recommendation algorithms. A match to a specific unit, however, depends on the position of the target unit within the course. When selecting an activity, instructors usually consider the balance of practicing newly introduced knowledge and reviewing learned knowledge, because students are most engaged when the material to be learned is neither too difficult nor too easy. Wang et al. [2] classify a learning activity in the progression as *reinforcement* (reviewing learned concepts), *recombination* of previously learned concepts, or *introduction* (introducing new concepts). Using the course model presented in Section 4.1,

[1] `http://www.sis.pitt.edu/~paws/ont/java.owl`



(a) Bag-of-concepts representation (b) Three-concept-sets representation

Fig. 3: Demonstration of representing learning content as programming concepts.

Content Wizard classifies each concept that appears in an activity into one of three categories (Figure 3b):

- *Past concepts (P)*: Concepts that were covered in previous units. These concepts are supposed to be known before starting the current unit.
- *Current concepts (C)*: Concepts that are covered in the current unit (and thus have not been covered in any previous units). We consider these concepts as targets of the current unit, according to the instructor's vision of the course.
- *Future concepts (F)*: Concepts that have not been covered up to the current unit. We assume that the instructor prefers to cover these concepts in future units (or not to cover them at all). Most likely, these concepts are not yet appropriate for students to learn in the context of the unit.

This representation reflects instructor preferences and enables our recommendation approach, in which recommended activities focus on current concepts, leverage learned concepts, and avoid future concepts.

4.4 The recommendation method

Content Wizard adaptively provides two valuable sources of information that can help instructors find the most appropriate content for each unit: *a ranked list* and *warning flags*. At every step of course creation, based on the current course model and the code examples provided for the target unit, the system will update the representations of all candidate learning activities during the recommendation process.

For each learning activity, a_i, consisting of three concept sets, P_i, C_i and F_i, the Wizard calculates its ranking score by linearly combining the contribution of the concept covered by the activity, according to the category to which each concept belongs. Equation (1) shows how we compute each content ranking score:

$$score_{a_i} = \alpha|P_i| + \beta|C_i| + \gamma|F_i| \tag{1}$$

α, β, and γ are the parameters controlling the importance of the three categories. The values for these parameters might be different for different domains and even for individual instructors, depending on how much they focus on current and past concepts and how much they want to avoid future concepts. Indeed, [2] shows the differences among the proportions of reinforcement, recombination,

and introduction of concepts of two Japanese textbooks and two online learning tools (Duolingo and Language Zen). Given sufficient volume of data (content selected by instructors for different domains and courses) these parameters could be learned from data; otherwise, they could be selected by using expert estimation. We explored both of these approaches in our evaluation, which is presented in the next section.

In addition to the ranking, we believe it is important for instructors to be aware of "not ready" learning activities that use concepts that do not appear in the code examples up to the current unit, because they could confuse less prepared students. We identify these activities as follows:

$$warning_{a_i} = \begin{cases} 1, & \text{if } |F_i| > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Activities with *warning* value 1 are annotated using a yellow warning icon (see the third, fourth, and sixth rows in Figure 1). The instructor can then evaluate whether an activity with potentially premature concepts should be assigned to the unit.

5 Evaluation

5.1 Datasets

To evaluate our proposed recommendation method, we collected three data sets from two different universities. Each dataset encapsulated instructor preferences in content selection (i.e., "ground truth").

Dataset 1: The data was collected from a Java class taught in the School of Information Sciences at the University of Pittsburgh in Fall 2016 (referred as IS17F16). The instructors followed a lecture-based format and created a course structure for IS17F16 that consists of 18 units (each unit includes two types of learning content, *annotated examples* and *parameterized problems*). No content recommendation functionality was used for content selection. As input, we collected code examples presented by the instructors in the course slides. All *annotated examples* in the content pool were used for ranking. As the ground truth, we used *annotated examples* that were selected by the instructors for each unit.

Dataset 2: The second dataset uses the same inputs as the first dataset for running the recommendation process. All items in the *problem* pool are ranked for recommendation (as shown in Figure 1), and the ground truth is the *problems* selected by the instructors for each unit of IS17F16.

Dataset 3: This dataset was extracted from the CS1 online programming course [2] taught at University of Helsinki, Finland. We mapped the course structure into ten coherent topics. Each topic has several *code examples* for students to learn new concepts (which we used as input) and several *coding exercises* to

[2] `http://mooc.fi/courses/2013/programming-part-1/material.html`

practice (which we used as the ground truth). Note that in this dataset, only *coding exercises* that were actually used in the course were ranked in the recommendation process, while in datasets 1 and 2, all items in the content pool were ranked for the course, including those that were not selected by the instructor.

5.2 Performance comparison

To assess the performance of our approach in a fair way, we estimate the values of the parameters in Equation (1) based on a preliminary analysis (we can't learn it from the same data that we use to evaluate the approach). We collected code examples of several topics from a Java programming textbook and ran the algorithm using Equation 1 while adjusting the parameter values in order to get the best recommendation results (by taking the book's contents as ground truth). The estimated values of α, β, and γ are set to **0.2**, **1**, and **-1.5**, respectively. As a baseline ranking approach, we use a popular content-based approach that ranks candidate items by cosine similarity between concept vectors that represent units and content items [16]. We refer to this approach as *tf*idf*, since we use a TF*IDF approach to assign weights for individual concepts in the concept vector. To measure ranking performance, we use three classical metrics: precision, recall, and F1 score (at top 3, top 5, top 10, and top 15).

Table 1: Performance comparison of Content Wizard vs. the baseline

Dataset	Method	Precision@top (%)				Recall@top (%)				F1@top (%)			
		3	5	10	15	3	5	10	15	3	5	10	15
1	**wizard**	**62.74**	**50.59**	**34.7**	**25.09**	**51.26**	**63.36**	**77.51**	**80.44**	**56.42**	**56.26**	**47.94**	**38.26**
	tf*idf	21.57	18.82	15.29	15.68	21.57	18.84	29.14	42.33	15.57	18.84	20.06	22.89
2	**wizard**	**47.05**	**37.64**	**32.94**	**26.66**	**34.23**	**41.91**	**66.32**	**76.86**	**39.63**	**39.66**	**44.01**	**39.59**
	tf*idf	21.57	17.65	14.11	14.11	17.63	23.05	33.33	43.08	19.40	20.00	19.83	21.27
3	**wizard**	**96.3**	**88.89**	**75.76**	**64.44**	**41.45**	**52.79**	**73.32**	**83.90**	**57.96**	**66.24**	**74.42**	**72.89**
	tf*idf	81.48	73.33	66.67	57.04	37.24	44.97	64.34	73.18	51.12	55.75	65.48	64.10

As shown in Table 1, Content Wizard outperforms the *tf*idf* method for all datasets. In all cases, Content Wizard archives a good recall performance of about 80% when presenting the top 15 results out of a pool of more than 200 learning items (i.e., 80% of all relevant items are included among the top 15 results). The table also shows two interesting differences between the F1 performance of both approaches on datasets 1-2 and on dataset 3. First, on the dataset 1 and 2, the performance of the Wizard is much better (2-3 times!) than the baseline while in the dataset 3 the relative difference is smaller. Second, the precision of both approaches is considerably higher for dataset 3. We believe that both differences stem from the differences in the nature of the datasets. The ranking tasks for the dataset 3 was much easier than for datasets 1-2. First, for dataset 3, recommender approaches had to rank only the actual items used in the course (and no "spares"). It was essentially a matter of matching each item to the best unit. The number of units to match was also much smaller (10 vs. 18). Recommendation for datasets 1-2 required ranking or all content

items in the repository out of which only a part was used in the course. Some of these items might be a poor match to the course, but some were not used by the instructors when creating the course simply because they wanted to select *some* relevant content for each unit, but not *all* relevant content. As shown by the data, a simple content-based algorithm might work reasonably well in simple cases, but in a more challenging (and realistic) context, Content Wizard offered a remarkable advantage.

6 Deeper Analysis

6.1 Finding the best values for the parameters

As presented in Section 5.2, the values of α, β, and γ used in performance evaluation were selected using a preliminary analysis. The performance of our systems with estimated parameters was better than the baseline, but it might still be improved by learning best parameters from the data. Since the precise balance between past, current, and future concepts may depend on the individual preferences of the instructor, the proper way to learn parameters for performance evaluation would be to use another earlier-authored course from the same instructor (with sufficient volumes of instructor-selected content). Since we have only one course for each instructor, the parameters learned from this course could not be used to evaluate recommendation performance on the same course. However, we could still post-assess the effectiveness of the estimated parameters and explore how the quality of the recommendation depends on the value of the parameters.

To achieve this goal, we ran 100 iterations from 0 to 10 with an increment of 0.1 for each of the parameters, for a total of 1,000,000 iterations. We found that within this range, the best values of α, β, and γ w.r.t F1@15 performances in dataset 1 are **0.167**, **1**, and **-3**, respectively; in dataset 2, they are **0.2**, **1**, and **-2.5**, respectively; and in dataset 3, they are **0.125**, **1** and **-2.3**, respectively (these values have been normalized by dividing all parameters by the values of β to have β equal 1). Although these best values vary slightly for each dataset, each set offers about the same small performance increase in comparison with the estimated values. For example, the F1@15 performance with the best data-derived values are 39.34, 41.18, and 74.31 for datasets 1, 2 and 3, respectively, as compared to 38.26, 39.59, and 72.89 F1@15 performance with manually estimated data.

Figure 4 shows how Content Wizard's performance changes with changing the parameters' values. Since the results are similar across the three datasets, we report only the results from dataset 2 (see Figure 4). To generate these figures, we consecutively fixed one of the parameters to its best value and plotted the change of performance for each reasonable combination of the remaining parameters. The results show that when the value of α increases (leading to the increased occurrence of past concepts in recommended content), the performance of the system tends to decrease. On the other hand, a larger absolute value of γ (leading to stricter penalty for future concepts) usually results in a better

performance. However, no single parameter could lead to the best performance; it is the combination of the contribution of all the concepts in the three categories. The results hint that the instructors in the courses used for our analysis do not pay a lot of attention to the *past concepts* and tend to avoid *future concepts* when choosing content for students to learn at the current unit.

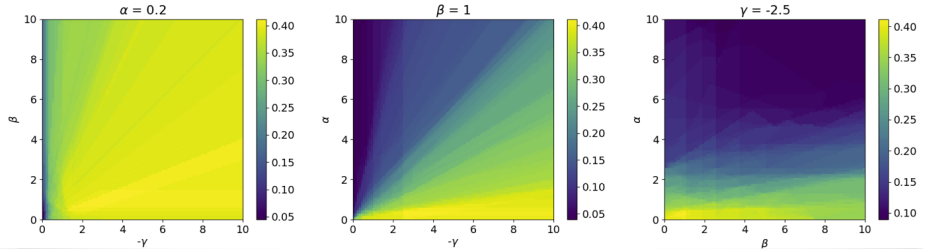


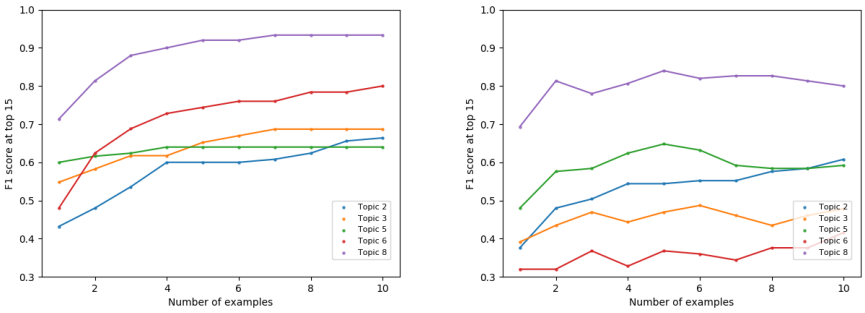
Fig. 4: F1@15 of the Wizard with different value sets of α, β, and γ in dataset 2.

6.2 How many code examples do we need?

One of the most important elements in our recommendation process is the code examples provided by instructors. The examples are vital to understand what an instructor expects students to learn in a given unit. It could be expected that the more examples are provided for each unit, the better items the Wizard recommends. But how many of these code examples are sufficient to achieve good quality results? In order to assess the impact of the number of provided examples, we picked the topics that had at least 10 examples and compared the performance of both approaches using from 1 to 10 examples (randomly selected from all examples provided by the unit) as input. As shown in Figure 5a, the performance of Content Wizard (measured by F1) consistently improves when the number of examples increases. In contrast, the baseline TF*IDF approach (Figure 5b) is not able to learn from an increasing number of examples. It could be also observed that with the first four to five examples the increase of quality reaches a *plateau*, which hints that four to five might be the optimal number of examples to ask instructors to provide.

6.3 Discovered limitations

A deeper analysis of recommendation performance helped us to reveal some limitations in our approach. First, the current approach doesn't take into account the fact that some concepts within a topic might be more important than others. For example, in the topic *while loop*, the concept *WhileStatement* is more important than concept *BreakExpression*. Exploring the importance of each concept and adding its weight to Equation (1) may potentially help Content Wizard to achieve a better ranking. However, it is still unclear how the importance of a specific concept can be derived from data rather than by asking the instructor.



(a) Content Wizard (b) TF*IDF

Fig. 5: F1@15 performances with different number of code examples for five topics in dataset 3 that have at least 10 code examples.

We tried a natural idea to use TF*IDF weights as importance weights, but this did not improve the overall performance of Content Wizard. Indeed, TF*IDF weights more heavily most unique concepts, i.e., *BreakExpression*, while the key topic concept, i.e, *WhileStatement* might dominate topic problems.

Second, we discovered that the code examples provided by the instructors are frequently not *exhaustive* in listing all concepts that the instructor wants to teach in a unit. By observing the automatically deduced course structure that was produced in our study, we noticed that although in most of the cases the concepts from the code examples cover all the concepts from the content selected by the instructors, some concepts such as *PostIncrementExpression* (+=) do appear in the selected content, but not in any provided code examples (though these do contain the concept *PostDecrementExpression* (-=)).

Finally, the Wizard will fail to recommend items for a unit that don't introduce new concepts, but instead use learned concepts to introduce a specific class of problems; for instance, *finding the maximum value in an array* or the topics *using truth values* and *instructors on code-writing and problem solving* in dataset 3. While a new *type of problems* could be considered as new knowledge [19, 20], it is not recognized by our Java parser.

7 Discussion and Future Work

This paper presented a course-adaptive content recommender system called Content Wizard, which assists instructors in authoring adaptive online programming courses. We introduce a novel unobtrusive example-based approach to build a fine-grained course structure that encodes an instructor's vision of course organization. We also presented a novel content recommendation approach that uses the whole course structure to recommend the most relevant content for each course unit. Altogether, these innovations aim to decrease the effort required to build a high-quality course based on reusable learning content (i.e., efficiency and time to author [5–7]) and facilitate the task of maintaining its coherent sequential structure. We believe that this approach could be most valuable for adaptive educational systems, such as ITSs or adaptive hypermedia, since personalization

algorithms require a much larger volume and variety of learning content for each topic (to allow fine-grain personalization), rather than static courses.

We assessed the performance of the proposed approach using three datasets collected from different universities. Comparing our system's performance with a standard baseline, we demonstrated that Content Wizard provides higher-quality recommendations, especially in more challenging and realistic contexts. The good recall performances suggest that Content Wizard could be efficiently used in a real-life context: with content ranked by Content Wizard, an instructor only needs to review the top 15 items out of several hundreds of items to select the ideal content for each course topic.

While our work indicates the strong potential of the suggested approach, we recognize that the approach itself and our evaluation process have several limitations, which we plan to address in future work. First, this study doesn't consider that different concepts might have different importance within a topic. Second, it doesn't account for the fact that the examples provided by instructors might not be exhaustive. While a group of related concepts is usually introduced in the same unit, only some of these concepts are usually illustrated in the code examples. In future work, we plan to extract the relationships between the programming concepts from the ontology introduced in Section 4.2, and assume that a group of closely related concepts is added as a whole to a unit once at least one concept is used in the examples. Third, the current Java parser is unable to recognize higher-level domain concepts, such as a specific *type of problem*. We intend to improve the Java parser in order to extract more complex knowledge of programming content.

On the evaluation side, while the data-centered (off-line) performance evaluation approach is a dominant way to evaluate recommender systems, we believe that only an online user study could provide a reliable assessment of the system as a tool to support course authors. In particular, a user study is essential to evaluate "beyond ranking" aspects of Content Wizard, such as content warning signs. In future work, we plan to engage course instructors in the evaluation process.

The current implementation and evaluation of our recommendation approach has been performed in the area of learning programming where problem-solving examples in the form of code are both popular and well-structured (allowing concept extraction). However, this approach could be easily adopted to other formal problem-oriented formal domains (math, physics, chemistry) that use structured worked examples (formulas or equations that could be analyzed for concept extractions). In the future, we plan to explore our approach in some of these domains.

References

1. Moffatt, D. V., Moffatt, P. B.: Eighteen Pascal Texts: An Objective Comparison. SIGCSE Bull. 14, 2 (June 1982), 2–10 (1982)

2. Wang, S., He, F., Andersen, E.: A Unified Framework for Knowledge Assessment and Progression Analysis and Design. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. ACM, New York, USA, 937–948 (2017)
3. Cafolla, R. (2006) Project MERLOT: Bringing Peer Review to Web-Based Educational Resources. Journal of Technology and Teacher Education 14 (2), 313-323.
4. Hislop, G., Cassel, L., Delcambre, L., Fox, E., Furuta, R., Brusilovsky, P., and Garcia, D. (2011) Sharing your instructional materials via ensemble. Journal of Computing Sciences in Colleges 26 (6), 160-162. x
5. Murray, T.: Authoring Intelligent Tutoring Systems: an analysis of the state of the art. International Journal of AIED 1, 10 (1999), 98–129 (1999)
6. Murray, T.: An Overview of Intelligent Tutoring System Authoring Tools: Updated Analysis of the State of the Art. In Authoring Tools for Advanced Technology Learning Environments: Toward Cost-Effective Adaptive, Interactive and Intelligent Educational So ware, Murray, T., Blessing, S.B., Ainsworth, S. (eds.). Springer Netherlands, Dordrecht, 491–544 (2003)
7. Sottilare, R.A.: Challenges to Enhancing Authoring Tools and Methods for Intelligent Tutoring Systems. In Design Recommendations for Intelligent Tutoring Systems, Sottilare, R.A., Graesser, A.C., Hu, X., Brawner, K. (eds.). Orlando, FL: U.S. Army Research Laboratory, 3–7 (2015)
8. Manouselis, N., Drachsler, H., Verbert, K., and Duval, E. (eds.) (2013) Recommender Systems for Learning. Berlin: Springer.
9. Mitrovic, A., Martin, B., Suraweera, P., Zakharov, K., Milik, N., Holland, J., Mcguigan, N.: ASPIRE: An Authoring System and Deployment Environment for Constraint-Based Tutors. International Journal on Artificial Intelligence in Education. 19, 2 (2009), 155–188 (2009)
10. Aleven, V., McLaren, B.M. , Sewall, J., Koedinger, K.R.:The Cognitive Tutor Authoring Tools (CTAT): Preliminary Evaluation of Efficiency Gains. In Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS'06). Springer-Verlag, Berlin, Heidelberg, 6170 (2006)
11. Brusilovsky, P., Eklund, J., and Schwarz, E: Web-based education for all: A tool for developing adaptive courseware. In: H. Ashman and P. Thistewaite (eds.) Proceedings of Seventh International World Wide Web Conference, Brisbane, Australia, 14-18 April 1998, pp. 291-300. (1998)
12. Lane, H.C., Core, M.G., Hays, M.J., Auerbach, D., Rosenberg, M.: Situated Pedagogical Authoring: Authoring Intelligent Tutors from a Students Perspective. In Artificial Intelligence in Education, Vol. 9112. Springer International Publishing, Madrid, Spain, 195–204 (2015)
13. Cristea, A.I., Aroyo, L.: Adaptive Authoring of Adaptive Educational Hypermedia. In Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems. Springer-Verlag, London, UK, 122–132 (2002)
14. Brusilovsky, P., Sosnovsky, S., Yudelson, M., Chavan, G.: Interactive Authoring Support for Adaptive Educational Systems. In Proceedings of the 2005 Conference on AIED: Supporting Learning Through Intelligent and Socially Informed Technology. IOS Press, Amsterdam, The Netherlands, 96–103 (2005)
15. Cabada, R.Z., Estrada, M.L.B., Garca, C.A.R.: EDUCA: A web 2.0 authoring tool for developing adaptive and intelligent tutoring systems using a Kohonen network. Expert Systems with Applications, 38 (8) (2011), 9522–9529 (2011)
16. Medio, C.D., Gasparetti, F., Limongelli, C., Sciarrone, F., Temperini, M.: Course-Driven Teacher Modeling for Learning Objects Recommendation in the Moodle LMS. In Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization (UMAP '17). ACM, New York, NY, USA, 141–145 (2017)

17. Hosseini, R., Brusilovsky, P.: JavaParser: A Fine-Grain Concept Indexing Tool for Java Problems. In The First Workshop on AI-supported Education for Computer Science. Springer-Verlag, Berlin, Heidelberg, 60–63 (2013)
18. Brusilovsky, P., Edwards, S., Kumar, A., Malmi, L., Benotti, L., Buck, D., Ihantola, P., Prince, R., Sirki, T., Sosnovsky, S., Urquiza, J., Vihavainen, A., and Wollowski, M. (2014) Increasing Adoption of Smart Learning Content for Computer Science Education. In: Proceedings of Proceedings of the Working Group Reports of the 2014 on Innovation and Technology in Computer Science Education Conference, Uppsala, Sweden, ACM, pp. 31-57.
19. Falmagne, J.C., Cosyn, E., Doignon, J.P., Thiery, N.: The assessment of knowledge, in theory and in practice. In Formal concept analysis. Springer, 61–79 (2006)
20. Doignon, J.P., Falmagne, J.C.: Knowledge Spaces and Learning Spaces. ArXiv e-prints (Nov. 2015). arXiv:math.CO/1511.06757 (2015)