Enriching Intelligent Textbooks with Interactivity: When Smart Content Allocation Goes Wrong

Alireza Javadian Sabet[1] [0000−0001−9459−2411], Isaac Alpizar-Chacon[2] [0000−0002−6931−9787], Jordan Barria-Pineda[1] [0000−0002−4961−4818], Peter Brusilovsky[1] [0000−0002−1902−1464], and Sergey Sosnovsky[2] [0000−0001−8023−1770]

[1] University of Pittsburgh, School of Computing and Information, Pittsburgh PA 152160, USA
{alj112,peterb,jab464}@pitt.edu
[2] Utrecht University, Department of Information and Computing Sciences, Princetonplein 5, 3584 CC Utrecht, Netherlands
{i.alpizarchacon,s.a.sosnovsky}@uu.nl

Abstract. One of the main directions of increasing the educational value of a digital textbook is its enrichment with interactive content. Such content can come from outside the textbooks - from multiple existing repositories of educational resources. However, finding the right place for such external resources is not always a trivial task. There exist multiple sources of potential problems: from mismatching metadata to mutually contradicting prerequisite-outcome structures of underlying resources, from differences in granularity and coverage to ontological conflicts. In this paper, we make an attempt to categorize these problems and give examples from our recent experiment on automated assignment of smart interactive learning content to the chapters of an intelligent textbook in a programming domain.

Keywords: Intelligent textbook · Smart content · Matching conflicts.

1 Introduction

One of the popular directions of augmenting electronic textbooks with novel functionalities is extending textbooks with "smart content" - interactive examples, simulations, and problems [3]. This direction is very important for the advancement of intelligent textbooks since learners' work with this smart content produces a much more reliable flow of information about learners knowledge and skill acquisition enabling better learning modeling approaches and better personalization.

However, the current platforms for development and delivery of interactive textbooks share the same problem: these textbooks are developed “as a whole” with text and interactive problems created together as a part of the authoring process. This approach allows developing excellent examples of interactive textbooks, but doesn’t support scaling up this process.

Our vision is a scalable approach for developing interactive textbooks, which could make interactive any textbook available in electronic format by augmenting it with smart learning content from existing repositories. An essential problem of this approach is matching smart content items to most appropriate textbook units. While originally this process has been performed manually, the ability to extract concept knowledge and other metadata from both textbooks and smart content items offers an opportunity to automate this process using AI techniques. In this paper we review existing approaches for augmenting textbooks with reusable learning content from content repositories and present a novel approach based on smart content indexing with domain concepts automatically extracted from a textbook. In following sections, we explain our current approach, present expert-based evaluation of the produced results, and discuss problems that have been revealed during this evaluation. We believe that an analysis of these problems will help in constructing more efficient content matching approaches for future interactive textbooks.

2 Related Work

2.1 Smart Content for Computer Science Textbooks

One of the first domains to embrace textbooks augmented with smart content was computer science education (CSE) where development of interactive learning activities from algorithm animations to automatically-assessed programming problems was a popular research direction [3]. The need to integrate interactive learning activities with online textbooks has been extensively discussed by computer science education community for many years [16] and some best examples of interactive textbooks were produced for computer science subjects. Existing interactive textbooks for CSE explored a range of smart content types to extend the core textbook. For example, ELM-ART [5] adaptive textbook for learning LISP included code examples that can be executed online and programming problems that were automatically evaluated by an intelligent program analysis component. OpenDSA [10] textbook for Data Structures and Algorithms included interactive algorithm animations and problems. RuneStone Python textbook [9] included interactive code examples, Parson’s problems, and code construction problems.

At the moment, several CSE research teams develop collections of smart learning content that could be reusable across multiple courses and textbooks. Most of these collections use LTI communication standard that enables smooth connection of smart content items to all kinds of host systems from LMS to LTI-compatible textbook platforms like OpenDSA [10]. Recently, the first catalog of LTI-compatible smart learning content for CSE have been published [11].

This development opens the way to practical application of intelligent content matching technologies discussed in this paper.

2.2 External Content Allocation Approaches

The problem of allocating learning content from content repositories has been explored long before smart interactive content became popular. This direction of research has its roots in similar problems of early learning management systems. While these systems provided space to be filled with all kinds of learning content and facilitated its inclusion, it become evident that a single instructor is not able to create the expected amounts of quality content. To answer the needs of the instructors, various repositories of learning content (also known as learning object repositories) were created. Good examples of these repositories are Ariadne [20] and Merlot [7], which were the focus of many projects exploring the development and use of content repositories. The original model of work supported by these repositories was to help users in finding relevant learning content for specific parts of their courses by offering flexible search tools. Here the needs were expressed by the user through queries and the search tools helped to find content that matches these needs.

The increased popularity of recommender systems encouraged many researchers to explore the use of recommender technologies to increase support to instructors looking for relevant content [13]. In a typical scenario, by observing user queries and selected content, a recommender system could learn about user needs and intention and recommend matching content that the user might not be able to find by herself [14, 17, 19]. Similarly, the increased popularity of exploratory search systems such as faceted browsing tools, encouraged an alternative approach to help used in finding most relevant learning content - extended interfaces for content selection and analysis. Instead of investing in artificial intelligence-based recommender, these systems seek to augment instructors own intelligence by providing visualization-based tools that helped users to understand which concepts are covered by each candidate item [6] and how this item fit to the target place in a course or a textbook [1].

The problem of the current generation of content allocation tool is the amount of human labor they required. The process assumes the engagement of domain experts who should analyze each target context where new content should be added, formulate a proper query and examine each candidate item. While recommender systems and content visualization tools help in this process, the whole procedure is still slow and expensive.

In response to this problem, a new generation of content matching tools attempted to further automate the process by building some form of a knowledge model for each target context, building comparable knowledge models for each candidate item, and then engage IR and AI approaches for automated matching. While a human instructor or textbook author might still be required to examine and approve matching results, the speed and the cost of the matching process is decreased considerably making it more scalable. Some early examples of this approach could be found in [8, 15]. In this paper we present and evaluate a new approach for automated matching of smart content to textbook sections. A unique feature of this approach is the ability to reconcile differences between concepts used to index book sections and concepts used to index smart content by creating a “bridge” between these concept spaces.

3 Method

3.1 The Textbook

For our experiment on automatic matching of smart content to textbook sections we have used textbook *Python for Everybody - Exploring Data Using Python 3* [18] (PYTHON). We applied our workflow for the automatic extraction of knowledge models from textbooks [2] to the PYTHON textbook. We identified the individual index terms from the Index section of the textbook using the extracted model of the textbook. Additionally, we collected the set of associated page references for each index term using the model of the textbook. We refer to the index terms as *BookConcepts* in the rest of the paper. In this way, it was possible to link *BookConcepts* to the sections of the textbooks. As a result of this process, each book section get connected to a set of concepts presented in its section. The *BookConcepts* set contained 852 concepts identified in the PYTHON textbook.

3.2 The Smart Content

In this work, we use four smart learning content types from several providers which cover both Python program understanding and construction skills: Animated Examples (52 items in total), Examples-Challenges (42 items in total), Tracing Problems (51 items in total) and Parsons Problems (34 items in total). Each of these items can be individually allocated to a proper place in the course. *Animated Examples* show the step-by-step execution of Python code snippets while making explicit the state of the variables within memory. *Examples-Challenges* are a set of activities that allow students to examine an example which explains how to solve a problem, and then gives them the option of solving a similar problem after where they have to drag-and-drop some lines of code to complete the right solution. *Tracing Problems* are parameterized problems that ask students to predict the output or the final value of a variable after executing a short code snippet. Finally, *Parsons Problems* are puzzle-like problems where the lines of code to solve a problem are presented in an unsorted way so students need to work on putting the lines of code in the right order. More details about the system with these four type of smart content can be found in [4]. All these smart learning activities were automatically indexed with Python ontology concepts by a parser application that extracted concepts from the code of the examples and problems.

From now on, we will refer to Tracing Problems (QuizPET) and Parson Problems as *Exercises*; to Animated Examples and Example-Challenges (Program Construction Examples) as *Examples*; and to the set of concepts used for indexing the *Exercises* and *Examples* as *SmartConcepts*. In total, *SmartConcepts* set contained 48 concepts extracted from smart content items (i.e., *Exercises* and *Example*) used in this work.

3.3 The Bridge between Concept Spaces

Given that all textbook sections were indexed using *BookConcepts* and all smart content items were indexed with *SmartConcepts*, it is not possible to match smart content items to book sections directly, first, some connections should be made between these two concept spaces. To create this bridge, all *Exercises* were manually indexed by the course instructor with the corresponding *BookConcepts*: for each *Exercise*, the instructor indicated which *BookConcepts* should be considered as the expected learning of these problems (i.e., mastery of which *BookConcepts* could be demonstrated by solving a specific exercise). Due to the broad coverage of the textbook, and the very specific nature of coding *Exercises*, only 47 *BookConcepts* were used for annotation. Essentially these were the concepts that the learner can master by practicing problem solving.

3.4 Automatic Content Allocation Procedure

The complete allocation procedure included the following steps.

Assigning book sections to lectures Since we planned to evaluate the approach in a lecture-based class, content allocation was performed lecture by lecture. The first step of the process was assigning a set of book sections as readings for each lecture. As it is usually done in college classes, this step was performed by the course instructor. By assigning readings to the lectures the instructor implicitly specified the concept goal for each lecture.

Creating a *BookConcepts* Profile of Each Lecture The goal of this step was creating a formal *BookConcepts* profile of each lecture as a set of *BookConcepts* that each lecture intends to teach. To achieve this goal we first created a full concept profile of each lecture by creating a union of *BookConcepts* from all textbook sections assigned from this lecture and then performed prerequisite-outcome separation process originally suggested in [6]. This process examines each lecture in sequence starting with the first lecture and separates all concepts from the full concept profile of a lecture into *outcomes*, concepts that the learning targets of the lecture and *prerequisites* - concepts that are used in this lecture, but were learning targets of previous lectures. For the first lecture, all of its full profile concepts are considered as outcomes, for the second only new concepts are considered as outcomes while all concepts already appearing the the earlier lectures (even if they are not explicitly mentioned in the lecture) become prerequisites, an so on.

Assigning Exercises to Each Lecture Considering the *BookConcepts* profile of the lecture and the *BookConcepts* index of each *Exercise* we assign *Exercises* to the lecture with the highest number for which the following two rules are observed: (1) the *Exercise* should not run ahead of the lecture: all *BookConcepts* of an *Exercise* should be either prerequisites of outcome of the lecture; (2) the *Exercise* should contribute to practicing the lecture goals: at least one *BookConcept* of the *Exercise* index should be a part of lecture outcomes.

After the first round of automatic exercise assignment, we assessed the results as explained in Section 4.2. The team examined mismatches, determined the source of problems, resolved problems and repeated the allocation step to reach 100% correct assignment.

Creating a *SmartConcepts* Profile for Each Lecture A *BookConcepts* profile of each lecture enabled us to do *Exercises* assignment since *Exercises* were manually indexed by *BookConcepts*, however it was not sufficient to assign *Examples*, which were only indexed with *SmartConcepts*. However, the presence of *Exercises* indexed by both *BookConcepts* and *SmartConcepts* created a bridge between the two distinct concept spaces and enabled us to create a *SmartConcepts* profile of each lecture. The process was similar to creating the *BookConcepts* profile explained above – we extracted SmartConcepts from all *Exercises* assigned to each lecture and performed the prerequisite-outcome separation process explained above. The result of this process was a list of prerequisite and outcome *SmartConcepts* for each lecture (in addition to already obtained list of prerequisite and outcome *BookConcepts*).

Assigning Examples to Each Lecture Since every lecture is now described as a set of *BookConcepts* **and** a set of *SmartConcepts*, we can run an *Example* allocation process for each lecture by matching *SmartConcepts* from lecture profile and *SmartConcepts* from *Example* index. This process was performed in the same way as *Exercise* assignment explained before, with the difference that *Example* assignment used *SmartConcepts* instead of *BookConcepts*.

After completing *Example* assignment, we performed the second round of evaluation explained in Section 4.3. We discovered and classified additional problems. Following this analysis, all problems were fixed. At the end of this process all *Examples* were assigned to their correct places in the course structure.

To measure the difficulty, the procedure considers the **number of concepts** and **number of lines**. In other words, the fewer number of concepts and number of lines of an *Exercise*, the easier the *Exercise*.

4 The Evaluation and Problem Analysis

This section provides the reader with the necessary information regarding the evaluation procedure (in Sect. 4.1) and problem analysis (in Sect. 4.2 and 4.3).

4.1 The Evaluation Procedure

As explained in Section 3.4, we perform expert evaluation of the automatic content assignment procedure twice. First round of evaluation was performed to assess automatic *Exercise* assignment and the second round to assess *Example* assignment. The evaluation was performed from a perspective of a course instructor examining appropriateness of smart learning content for each of the course lectures. The goal of evaluation was to detect misplaced content items and detect potential flaws of the process that lead to this misplacement. Every discovered case of misplacement was recorded and discussed by the team. Through this discussion we connected each case to a specific problem which are reviewed below in details since understanding of these problems is critical to improve this and future automatic allocation approaches. It is important to stress that after achieving this evaluation goal in each round, we fixed all discovered problems to ensure that the next round starts with correct data. The problems that were results of manual errors were fixed by re-running the allocation procedure. The problems related to conceptual issues of the process were fixed manually by allocating each content item to its agreed proper place.

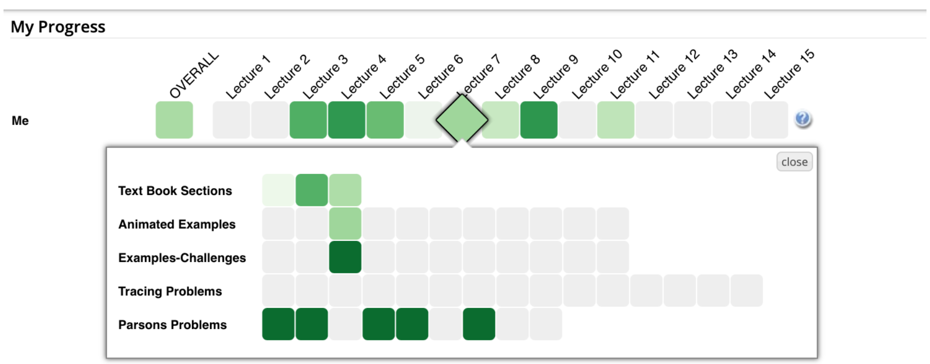

Fig. 1. The MasteryGrids interface with list of learning content allocated for each lecture

To facilitate the evaluation process, we placed all allocated content into the MasteryGrids interface to experience the content in exactly the same way as an instructor or a student will see it in the course. Figure 1 presents the Mastery Grid interface used by the students. The course is shown as a sequence of lectures. Clicking on each lecture brings up a panel that shows all lecture-related content including book sections, *Examples* (e.g., Animated Examples), and *Exercises*. Note that green colors that display individual student knowledge and progress were not used in the examination process. From this interface, clicking on a specific content item, provide access to the selected smart content - see Figure 2 showing an opened an example-challenge item. Using this interface, the experts analyzed each automatically-allocated content item assessing its relevance to the assigned lecture.

4.2 Evaluating Exercise Allocation

The key source of exercise mis-allocation were manual indexing “glitches”. Since manual indexing was done by the single course instructor and without clearly specified rules, the indexing was in several cases inconsistent or incomplete. Most interesting among the observed manual indexing problem was the generality problem. The *BookConcepts* in the textbook form a taxonomy where a number of specific concepts are covered by a more general concept. For example, “for loop” and “while loop” are specific concepts, but both of them are covered by a more general concept “iteration (loop)”. Similarly, Python prescribes several specific cases of indentation in loops, functions, conditional statements, etc, but all of these specific cases are covered by a more general concept of “indentation”.

It is natural that most textbooks introduce general concepts when two or more specific cases were presented. I.e., in the PYTHON book, first code examples with indentations were introduced Chapter 3, but it explicitly appears as a concept and becomes a part of *BookConcept* section index only in Chapter 4 by which time several indentation cases are presented and the introduction of this general concept becomes meaningful. Due to this role of more general concepts, many cases where the instructor used a general *BookConcept* (like “iteration (loop)” or “indentation”) lead to exercise misplacement. I.e., if an exercise that really belongs to Chapter 3 was indexed with “indentation” (because, indentation was featured in the exercise code) it will be allocated to Chapter 4 since this is the first place where this *BookConcept* appears.

We concluded that all observed manual indexing issues could be resolved by creating more formal rules of indexing assembled in a “codebook”, for example, as suggested in [21]. One of these rules should be exclusion of too general concepts like loops, indentation, statement, etc. from content indexing. To fix these problems, the instructor reviewed *Exercise* indexing making it more consistent, removing more general concepts and only keeping the specific concepts. For example, “iteration” as an initial concept was replaced by “for loop” and “while loop”. After that, automatic exercise allocation process was repeated, which fixed all observed errors.

4.3 Evaluating Example Allocation

In total, our automatic content allocation procedure assigned 94 *Examples* to the lectures. Among them 80 *Examples* were allocated correctly and 14 had to be manually reallocated. In other words, the automatic content allocation was successful more than 85% of the time. The cases of wrong example allocations were distilled to two main groups of issues, namely *coverage-related* and *parser-related* (imperfect matching) issues which are explained below.

Coverage-related issues Coverage related issues have their roots in clusters of similar concepts in programming languages, such as a set of *arithmetic operation* concepts, that are usually learned together in the same lecture. Due to their similarity, instructors rarely pay attention whether or not all of these concepts are covered by lecture *Examples* or practice problems. This is what happened more than once in our case. Despite of several *Exercises* specifically focused on practicing simple arithmetic operations, all correctly allocated to Lecture 3, none of them included "floor division" operation. As a result, "floor division" concept has not been added to the *SmartConcepts* profile of Lecture 3. At the same time, "floor division" appeared as a part of code in one of Lecture 5 problems. While it was a side concept there, it was the first lecture where "floor division" appeared and it was assigned as one of the outcomes in Lecture 5 *SmartConcepts* profile.

However, some of the *Examples* focused on practicing arithmetic operations did include "integer division" in their code. (see Figure 2). The parser successfully identified "floor division" concept in the body of code (see Lines 12 and 15) and included it in the *Example* index. With that this example was allocated to Lecture 5 (where it was out of place) instead of its correct location in Lecture 3.

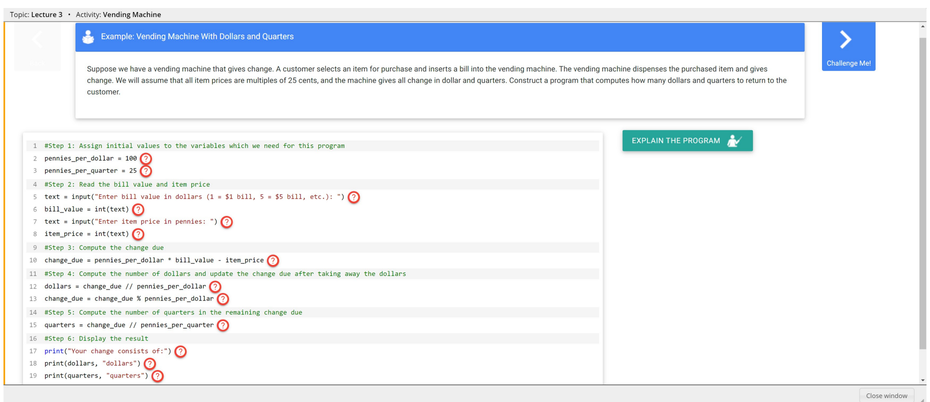

Fig. 2. Presentation of an Example-Challenge in MasteryGrids. The shown example demonstrates a coverage-related issue (missing "division" arithmetic operation in Lines 12 and 15).

In our evaluation study, we observed 4 coverage-related issues which had to be reallocated manually. This reallocation comprised 4.2% of all cases. A possible long-term solution for this category of problems would be using concept grouping techniques: a set of concepts (such as "arithmetic operations" that are usually introduced and practiced together should be combined into an "unbreakable group". Whenever one of the concepts in such a group is added to a set of concept outcome of a specific lecture, the rest of them should be added too.

Parser-related issues (imperfect indexing) Several mis-allocation cases were traced to the insufficient sensitivity of our parser that was used to extract *SmartConcepts* from code. Most of these errors were related to the minimalistic approach to syntax in Python where the same syntactic form is used to express concepts that are semantically or pedagogically different. For example, the parser was not able to recognize important file operations `open(file)` and `file.close()` since from syntactic prospect these cases are not different from any other functions or methods. As a result, instead of being assigned to a lecture focused on working with files, these *Examples* were misplaced in a lecture focused on functions and methods. We had to reallocate three examples of this kind which comprised 3.2% of *Examples*.

Similarly, the parser was not able to identify the use of a "class constructor" as a special case since it was also looking just like any other function call (see Figure 3). It resulted in mis-allocation of two examples which were not placed into a lecture on object-oriented programming where they belong. We had to manually reallocate two *Examples* of this kind which comprise 2.1% of all *Examples*.

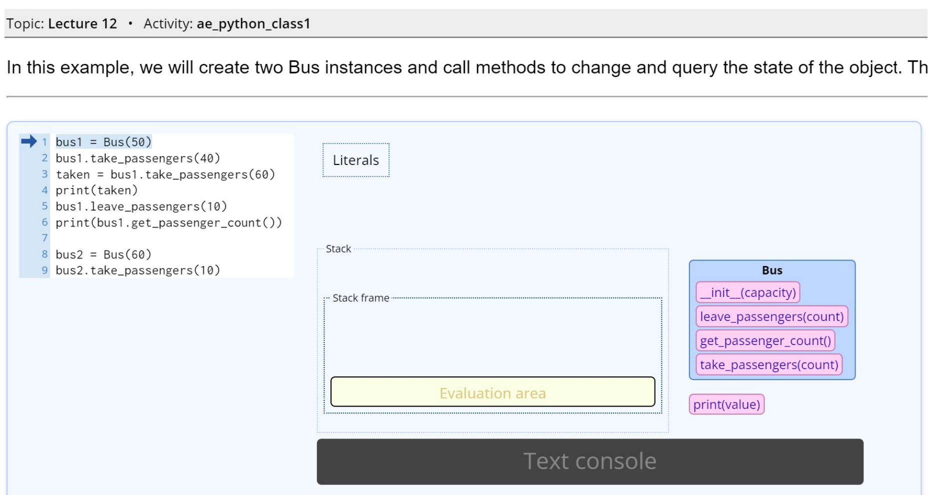

Fig. 3. Presentation of an Animated Example in MasteryGrids. This example demonstrates a parser-related issue (inability of detecting creating object from a Class).

Yet another example, is parser's failure to recognize as a special case `.format()` expression. This special case indicated that the *Example* should be allocated to "string methods and regular expression" lecture, but for our parser it was looking just like any other attribute causing these *Examples* to be mis-allocated. To fix this issues we had to reallocate five *Examples* from the prior lecture to the corresponding lecture compromising 5.3% of the whole *Examples*.

All of these cases point to the weakness of the current parser version. A long term solution to address the observed problem is to refine the parser so that concepts that are similar syntactically but different pedagogically are recognized as different concepts. When developing a concept parser for Java [12] we achieved this goal by creating a syntax-independent Java ontology and creating rules to match leaves of Abstract Syntax Tree to ontology concepts. For Python, however, we created a simplified version of the parser.

5 Conclusion and Future Work

In this paper, we presented the first evaluation of our automatic smart contact allocation procedure based on automatic indexing of a textbook and a collection of smart content for programming. We focused the paper on examining misallocation cases detected during expert evaluation process since we believe this analysis is important for the progress of research on automatic allocation. We are currently running a classroom study to explore to what extent the results of our automatic allocation were acceptable to students. In our future work we will focus on resolving the problems discovered in this study through developing a content allocation “codebook” for instructors and a better version of smart content parser.

References

1. Albó, L., Barria-Pineda, J., Brusilovsky, P., Hernández-Leo, D.: Concept-level design analytics for blended courses. In: 14th European Conference on Technology Enhanced Learning (EC-TEL 2019). vol. 11722, p. 541–554. Springer International Publishing (2019), https://doi.org/10.1007/978-3-030-29736-7_40
2. Alpizar-Chacon, I., Sosnovsky, S.: Knowledge models from pdf textbooks. New Review of Hypermedia and Multimedia **27**(1-2), 128–176 (2021)
3. Brusilovsky, P., Edwards, S., Kumar, A., Malmi, L., Benotti, L., Buck, D., Ihantola, P., Prince, R., Sirkiä, T., Sosnovsky, S., Urquiza, J., Vihavainen, A., Wollowski, M.: Increasing adoption of smart learning content for computer science education. In: Working Group Reports of the 2014 on Innovation and Technology in Computer Science Education Conference. pp. 31–57. ACM (2014). https://doi.org/10.1145/2713609.2713611
4. Brusilovsky, P., Malmi, L., Hosseini, R., Guerra, J., Sirkiä, T., Pollari-Malmi, K.: An integrated practice system for learning programming in python: design and evaluation. Research and Practice in Technology Enhanced Learning **13**(18), 18.1–18.40 (2018). https://doi.org/10.1186/s41039-018-0085-9
5. Brusilovsky, P., Schwarz, E., Weber, G.: Electronic textbooks on www: from static hypertext to interactivity and adaptivity. Web Based Instruction. Educational Technology Publications, Englewood Cliffs, New Jersey pp. 255–261 (1997)
6. Brusilovsky, P., Sosnovsky, S., Yudelson, M., Chavan, G.: Interactive authoring support for adaptive educational systems. In: 12th International Conference on Artificial Intelligence in Education, AIED’2005. pp. 96–103. IOS Press (2005)
7. Cafolla, R.: Project merlot: Bringing peer review to web-based educational resources. Journal of Technology and Teacher Education **14**(2), 313–323 (2006)

8. Chau, H., Barria-Pineda, J., Brusilovsky, P.: Course-adaptive content recommender for course authoring. In: 13th European Conference on Technology Enhanced Learning, EC-TEL 2018. pp. 437–451. Lecture Notes in Computer Science, Springer (2018)
9. Ericsson, K.A.: Towards a science of the acquisition of expert performance in sports: Clarifying the differences between deliberate practice and other types of practice. Journal of sports sciences **38**(2), 159–176 (2020)
10. Fouh, E., Karavirta, V., Breakiron, D.A., Hamouda, S., Hall, S., Naps, T.L., Shaffer, C.A.: Design and architecture of an interactive etextbook–the opendsa system. Science of computer programming **88**, 22–40 (2014)
11. Hicks, A., Akhuseyinoglu, K., Shaffer, C., Brusilovsky, P.: Live catalog of smart learning objects for computer science education. In: Sixth SPLICE Workshop "Building an Infrastructure for Computer Science Education Research and Practice at Scale" at ACM Learning at Scale 2020 (2020), https://cssplice.github.io/LAS20/proc/SPLICE_2020_LS_paper_7.pdf
12. 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 (AIEDCS) at the 16th Annual Conference on Artificial Intelligence in Education, AIED 2013. pp. 60–63 (2013), https://sites.google.com/site/aiedcs2013/proceedings
13. Manouselis, N., Drachsler, H., Verbert, K., Duval, E.: Recommender Systems for Learning. Springer, Berlin (2013)
14. Manouselis, N., Kosmopoulos, T., Kastrantas, K.: Developing a recommendation web service for a federation of learning repositories. In: 2009 Int. Conference on Intelligent Networking and Collaborative Systems. pp. 208–211. IEEE (2009)
15. Matayoshi, J., Lechuga, C.: Automated matching of its problems with textbook content. In: Second Workshop on Intelligent Textbooks at 21st International Conference on Artificial Intelligence in Education (AIED 2020). CEUR (2020)
16. Rößling, G., Naps, T., Hall, M.S., Karavirta, V., Kerren, A., Leska, C., Moreno, A., Oechsle, R., Rodger, S.H., Urquiza-Fuentes, J., et al.: Merging interactive visualizations with hypertextbooks and course management. In: Working group reports on ITiCSE on Innovation and tech. in comp. science education, pp. 166–181 (2006)
17. Ruiz-Iniesta, A., Jiménez-Díaz, G., Gómez-Albarrán, M.: User-adaptive recommendation techniques in repositories of learning objects: Combining long-term and short-term learning goals. In: 4th European Conference on Technology Enhanced Learning (ECTEL 2009). Lecture Notes in Computer Science, vol. 5794, pp. 645–650. Springer-Verlag (2009)
18. Severance, C.R.: Python for everybody: Exploring Data using python 3. CreateSpace Independent Publishing (2016)
19. Sicilia, M.A., García-Barriocanal, E., Sánchez-Alonso, S., Cechinel, C.: Exploring user-based recommender results in large learning object repositories: the case of merlot. In: 1st Workshop on Recommender Systems for Technology Enhanced Learning (RecSysTEL 2010) at the 2010 ACM conference on Recommender systems, RecSys '10. vol. 1 (2), p. 2839–2848. ACM (2010)
20. Verhoeven, B., Cardinaels, K., Van Durm, R., Duval, E., Olivié, H.: Experiences with the ariadne pedagogical document repository. In: ED-MEDIA'2001 - World Conference on Educational Multimedia, Hypermedia and Telecommunications. pp. 1949–1954. AACE (2001)
21. Wang, M., Chau, H., Thaker, K., Brusilovsky, P., He, D.: Knowledge annotation for intelligent textbooks. Technology, Knowledge and Learning p. in press (2021), https://link.springer.com/article/10.1007/s10758-021-09544-z