Integrating Textbooks with Smart Interactive Content for Learning Programming

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

[1] Utrecht University, Utrecht, The Netherlands
{i.alpizarchacon,s.a.sosnovsky}@uu.nl
[2] University of Pittsburgh, Pittsburgh, USA
{jab464,kaa108,peterb}@pitt.edu

Abstract. Online textbooks with interactive content emerged as a popular medium for learning programming and other computer science topics. While the textbook component supports acquisition of programming concepts by reading, various types of "smart" interactive learning content such as worked examples, code animations, Parson's puzzles, and coding problems allow students to immediately practice and master the newly learned concepts. This paper attempts to automate the time-consuming manual process of augmenting textbooks with "smart" interactive content. We introduce an ontology-based approach that can link fragment of text with "smart" content activities, demonstrate its application to two practical linking cases, and present the results of its pilot evaluation.

Keywords: electronic textbook · introductory programming · interactive learning content.

1 Introduction

[3] Electronic textbooks and various kinds of "smart" interactive systems such as Intelligent Tutoring Systems (ITS) or virtual labs have been traditionally considered as two opposite ways to leverage the power of computers for human learning. The research on electronic textbooks attempted to enhance learning-by-reading supported by traditional textbooks by augmenting them with internal hyperlinks [39], semantic references [3], links to external material [23], annotations [33, 48], and even question answering [17]. In contrast, interactive learning tools focused on supporting learning-by-doing by offering students a chance to solve problems with an assistance of an intelligent tutor [8, 47], examine interactive worked examples [34, 49], or explore simulations [36].

Gradually, the recognition of complementary nature of learning-by-reading and learning-by-doing encouraged an increasing stream of research on integrating textbooks with interactive content [15]. This work has been most noticeable

in computer science domain, such as learning programming languages, where researchers and practitioners developed and explored a broad range of "smart" interactive learning content [13] such as coding problems [20], interactive examples [49], Parson's puzzles [37], and program visualizations [36]. Starting from the early attempt to augment multimedia and Web-based programming textbook with live problems [10], intelligent tutors [14], and interactive animations [11], the research on programming textbooks with "smart" interactive content lead to the development of modern online interactive textbooks that are used by thousands of users [42, 21].

Yet, textbooks with "smart" interactive content are still a minority among other types of learning tools due to considerable problems of integrating traditional text with "smart content". While technical problems associated with integration are being gradually addressed by modern interoperability standards, the conceptual problems related to linking text with interactive content (which interactive activity is the best match for a section or text?) are not yet resolved. The current generation of interactive textbook is still developed by manual allocation of interactive content developed by the textbook authors to textbook sections. This approach has scaling problems and complicates the reuse of smart learning content created by other authors. While approaches for automatic linking of textbook sections with various types of text-based resources such as other textbooks [23] of Wikipedia [3] have already been developed, automatic linking of text and complex activities has not been attempted. In this paper we present our first attempt to cross the border between text and smart interactive content for learning computer programming. As a domain to explore linking text with interactive content, programming domains offer one substantial advantage. A well-structured nature of programming code associated with interactive content makes it possible to extract knowledge components from the code in a scalable way [26]. We introduce a novel ontology-based linking approach and demonstrate its use to solve two types of automatic linking problems: augmenting textbook sections with smart interactive content and extending topic-focusing collections of smart content with relevant reading resources. We also present an attempt to evaluate the quality of linking and discuss our experience.

2 Related work

Effective integration of various types of learning content and systems serving it has been both an important practical problem and a long-standing research challenge for the developers of educational software. On a more practical side, several issues have been addresses with different degrees of success. For example, we have a range of standard protocols for reliable identification of users across multiple systems [27, 25, 40]. There is also a strong community support behind standards for learning record stores aggregating educational data from external sources [2, 30]. At the same time, several interoperability standards have struggled to reach wider adoption despite initial promises [28, 1, 29]. From the research perspective, the Artificial Intelligence in Education (AIED) community has ex-

plored the problem of integration of intelligent and adaptive educational systems on multiple levels, including distributed personalisation architectures [12, 45] and centralised student modelling servers [31, 16], mapping domain models [43] and educational ontologies [18]. Ultimately, the motivation for such integration is a composition of a richer, more effective educational environments that can provide guided access to an assortment of educational content of different types and enable deeper learning. Reading material is an integral component of such educational setups as the main source of conceptual knowledge and potential destination for reflective and remedial learning activity.

From the architecture perspective, there are two primary models for integrating textbooks with smart interactive content:

- linking external interactive content into the relevant parts of a textbook;
- linking relevant fragments of a textbook into an existing interactive education system;

The former method has been implemented in several successful system. For example, the classic adaptive system for learning LISP - ELM-ART [14] - is organised as an electronic textbook augmented with training exercises. Students reading the textbook can practice their knowledge, thus providing ELM-ART with evidence for student modelling and adaptation. Another example of a similar organisation is Runestone textbooks [21] augmented with several types of interactive content including Parson's puzzles. Examples of the second method are less numerous. [44] describes OOPS - an adaptive service that recommends relevant sections from a textbook to students solving self-assessment quizzes on Java. It is worth noting that another important distinction of the OOPS service is that it linked textbook sections to relevant quizzes and questions in an automated way.

The automated linking of textbooks is another important stream of research. However, we are not aware of other examples of automatic linking of textbooks to smart interactive content besides OOPS. Most authors have looked into different ways to either cross-link multiple textbooks within the same domain [23], or integrate textbooks with external repositories of reading material [3, 35]. In this paper, we seek to fill this gap by implementing a linking model between textbooks and smart interactive programming content.

3 Systems

3.1 Intextbooks

The main platform used in our studies of linking textbooks with interactive content is the Intextbooks (Intelligent textbooks) system [4, 7]. Intextbooks consists of two main components. The online component supports students' interaction with the online electronic textbooks (see Fig. 1), while the offline component performs transformation of PDF textbooks into online interactive textbooks through modeling and conversion to HTML. After extracting a knowledge model

from a PDF textbook, it converts it into an HTML/CSS representation with a fine-grained DOM (Document Object Model) enriched with semantic information extracted from the content and formatting of the textbook. As a result, this implementation is flexible in terms of potential interactivity as virtually any textbook object (from a chapter to a keyword) can become an object of targeted interaction.

The offline component extracts a semantic model of a textbook using a rule-based system. Its ruleset captures common conventions and formatting guidelines for textbook formatting, structuring, and organization. Such elements and tables of contents and indices play a crucial role. More information can be found in [6]. Additionally, the domain terms extracted from the textbook index are linked to DBpedia[4] resources using a category of interest to indicate the primary domain of the textbook. As a result, the model is enriched with additional semantic information [5]. Then, the knowledge model is serialized as an XML file using the Text Encoding Initiative (TEI)[5] and the textbook is converted into an HTML representation. Finally, TEI and HTML representations are synchronized, meaning all elements of the TEI model are connected to the DOM elements of the HTML version of the textbook. The online Web-reader presents processed textbooks to students. Every time a student requests a textbook, the reader displays the synchronized HTML representation of the textbook and supports various interactions with it.

3.2 Python Programming Personalized Practice System

To assess the value of automatically extracting concepts from Python textbooks, we also decided to explore linking the different sections of these books with the learning units presented in the Python Personalized Programming Practice System (P^4), based on their conceptual similarity. P^4 is an online personalized system offering students in introductory Python programming courses to practice their skills using several types of interactive learning materials. The system is designed as a non-mandatory practice and self-assessment tool that each student could use for individual needs. P^4 was developed by using the Mastery Grids system [22] as its core. Each topic within the Python course is represented by a square cell in the top row (see top left in Fig. 2). Students can monitor their progress by checking the color of the grid cells, i.e., the greener the cell the more correct activities they have within that topic. For accessing the learning materials on a specific topic, students have to click the corresponding topic cell, which opens the learning activities selection section (see left center part in Fig. 2). Several types of learning activities are presented here, all of them in a different row, ranging from "Animated Examples" to "Parsons Problems".

On top of it, personalized guidance is provided, based on a concept-level model of student's Python knowledge. The conceptual structure of the student model is driven by an ontology of Python programming concepts [6] (from now on,

[4] http://dbpedia.org
[5] https://tei-c.org/
[6] http://acos.cs.hut.fi/static/python-parser/ontology.png

the Python ontology), which was homologically created by using a Java ontology as a template [26]. The ontology is composed by leaf and inner nodes. A leaf node represents a Python concept. Inner nodes are used as a hierarchy of classes for the concepts.

The model is built by observing student behavior in the system and represents the probability of students knowing each Python concept. To make this learner model "open" to the student, it is visualized as a bar chart on the bottom part of the activity selection interface (see Fig. 2). Each bar depicts one concept, and the height represents the estimated level of knowledge (i.e., the taller it is, the more the estimation of knowledge). Based on these concept estimations, P^4 recommends the three learning activities that are more appropriate for the student at that stage by following a specific learning goal (e.g., knowledge maximization or misconceptions' remediation). Recommended learning materials are highlighted with stars of different sizes within the interface (see Fig. 2).

3.3 Intextbooks/P^4 integration

The research presented in this paper primarily benefits from the annotation of the textbook's content with domain terms in Intextbooks and the topic-concept-activity model in P^4. Each content unit (page, sub-chapter, chapter) is annotated with its corresponding domain terms in the resulting knowledge models for a textbook. When those domain terms are linked to the Python programming concepts used in P^4, two potential integrations are enabled: (1) learning activities from P^4 can be displayed along with the corresponding content units in Intextbooks, and (2) content units from textbooks in Intextbooks can be additional learning activities associated with the most appropriate topics in P^4.

We present how this two-way integration for learning programming looks like in each of the two systems. As a working example, we linked the content from the *"Python for Everybody"* textbook [41] with the topics-concepts-activities in P^4. Specifically, we show the link between a sub-chapter from the textbook and the "While Loops" topic (see Figures 1 and 2).

Figure 1 reflects the addition of learning activities associated with specific sub-chapters in Intextbooks. Since each sub-chapter is annotated with domain terms, learning activities from P^4 that cover the same conceptual terms can be included in Intextbooks. When a user is navigating a sub-chapter linked with one or more learning activities, these are displayed as additional content (see top-right panel on Fig. 1). The user can interact directly with the learning activities without leaving the system.

Figure 2 shows how "Textbook readings" were added as an additional type of learning activity (last row) in P^4. When a "Textbook readings" cell is clicked, P^4 directs students to the Reading Mirror system [9] (online reading tool integrated within P^4), specifically focusing on the corresponding sub-chapter that has been associated with the topic given the concepts it covers. In the same way, as the other types of learning activities, "Textbook readings" are capable of being recommended to students as well (specially when the concepts covered were just introduced or need to be reinforced).

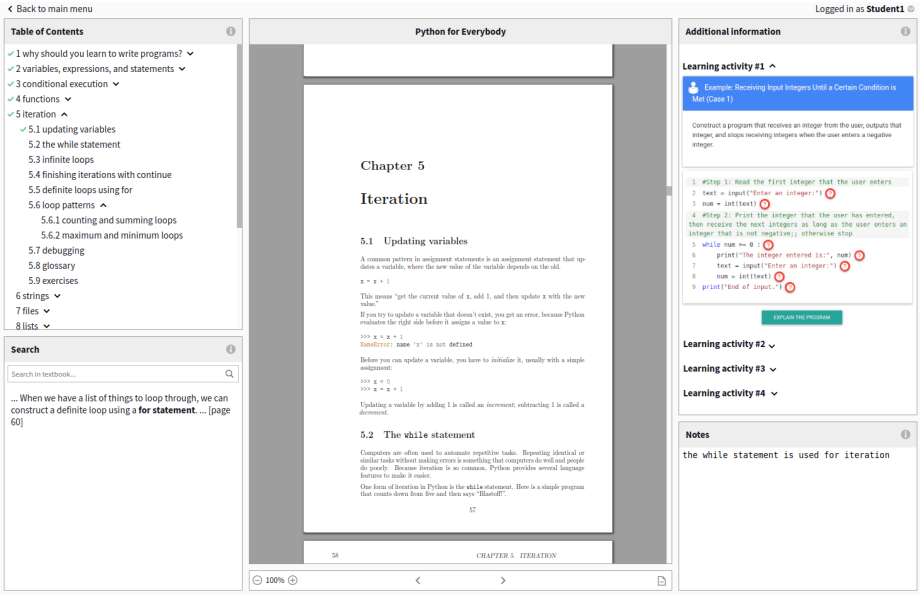

Fig. 1. Integration of external learning activities as an additional content within Intextbooks, given the automatic linkage between textbook domain terms and Python programming concepts

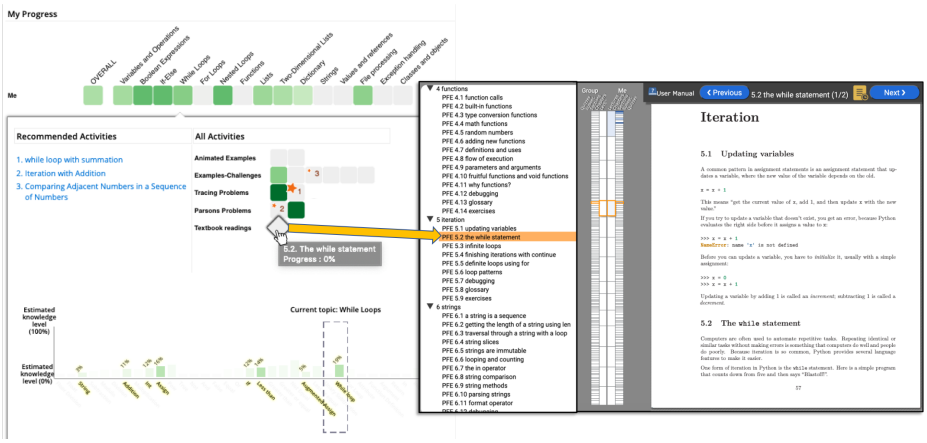

Fig. 2. Integration of textbook sections as an additional learning material within P^4, given the automatic linkage between domain terms and Python programming concepts

4 Methodology

Our goal is to map the knowledge models extracted from Python textbooks to the concepts and topics model available in the P^4 system. This mapping will allow

the interchanging of content from both models: (1) learning activities from P^4 can be displayed directly in our Intextbooks system, and (2) sub-chapters from the Python textbooks that present and discuss the topics used in P^4 can be displayed as an additional type of content. Our methodology for mapping both models include three main steps and two sub-steps:

1. Knowledge model extraction and glossary unification
2. Glossary - ontology linking
3. Granular content linking
   3.1. Textbook sub-chapters to P^4
   3.2. P^4 learning activities to textbook sub-chapters

Figure 3 shows the elements from both systems (Intextbooks and P^4) used in the methodology. Each step is described in the following subsections.

4.1 Knowledge model extraction and glossary unification

The first step in our methodology is to extract the list of all the index terms present in the Python textbooks to create a single unified glossary of terms to be mapped with the Python ontology.

First, for each textbook, a knowledge model is extracted and enriched with semantic information from DBpedia. For the enrichment process, the DBpedia category *Computer_programming* is used. Then, a glossary of index terms is created for each textbook. Each term in the glossary has a preferred label (PF), a set of alternative labels (AL), and an external concept (EC). The first one corresponds to the ID of the term, the second to a set of alternative names for the same term, and the last one, to an external concept from a different glossary or ontology. For example, a glossary term is {PF: if statement; AL: statement <> if; EC:- } (<> is used for hierarchical index terms). Initially, no term has an external concept associated with it; this will be done in the next step. The presented glossary term corresponds to the "id statement" and "statement <> if" index terms from the *"Python for Everybody"* textbook [41].

Since two terms representing the same concept can be written differently, we merge all glossaries to identify repeated terms. Terms that are identified as the same are merged, keeping the individual terms as alternative labels. Besides exact textual matching, three cases are handled: (1) different lexical forms (e.g., *branch* and *branches*), (2) acronyms (e.g., *API* and *APIs (Application Programming Interfaces)*), and (3) synonyms (e.g., *Turing Completeness* and *Turing complete programming language*). The first case is handle by comparing the terms using their stems (computed for each word using the snowball stemming algorithm [38]). Secondly, acronyms are handled by first identifying if an index term contains some text between parenthesis and then splitting the term into two. Each part of the term is compared against the other terms. Finally, synonyms are handled with the help of DBpedia: terms that were linked to the same DBpedia resource during the enrichment process are merged. Also, we use the `dbo:wikiPageRedirects` property of the DBpedia resources, which

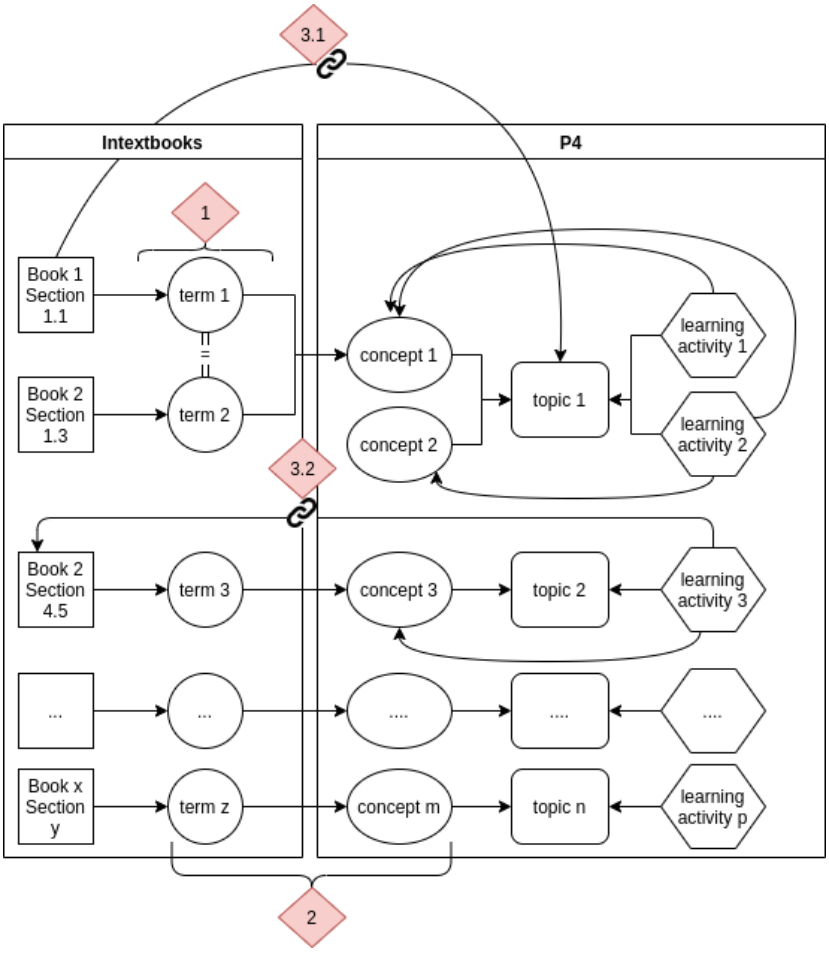

Fig. 3. Elements from both Intextbooks and P[4] used in the different steps of the methodology

indicates synonyms, common misspellings, and acronyms to increase the matching between the terms. The result of this step is one unified glossary with the index terms from all the textbooks. The number 1 in Fig. 3 illustrates that the unification of the different terms from the textbook forms the unified glossary.

4.2 Glossary - ontology linking

The next step in the methodology is to link the terms from the glossary to the concepts in the Python ontology.

First, the concepts in the Python ontology are extracted. For each concept, both the single name (e.g., *while*) and the compound name with the parent

classes (e.g., *while* <> *iteration statement* <> *statement* <> *python language* <> *python*) are retrieved. Then, a linking strategy is applied. First, glossary terms are linked to the Python concepts using exact textual matching between the different labels of the terms and the single name of the concepts. Then, stemming is applied to find more matches. Third, a small list of abbreviations is used. The list was created manually since the Python ontology uses some abbreviated names (e.g. *int* and *div*) that appear with a full name in the glossary (e.g., *integer* and *division*). Fourth, the parent classes of the Python concepts are used. Since the class hierarchy in the Python ontology has a deeper granularity than the names used in the textbooks, matches between a complete glossary term and a partial compound concept name are accepted. For example, the glossary term *if statement* is linked to the concept *if* <> *selection statement* <> *statement* <> *python language* <> *python* since the term matches the concept's single name (*if*) and one of its parent classes (*statement*). Finally, the glossary terms that are not linked to any Python concept are compared to the inner classes of the Python ontology (e.g., *Boolean Expression*) using the mentioned strategies (textual matching, stemming, and synonyms). The result of this step is that the linked glossary terms have a Python concept as an external concept (e.g., {PF: if statement; AL: statement <> if, statement <> conditional; EC: if }). The number 2 in Fig. 3 illustrates the linking between the terms in the glossary and the concepts from the Python ontology used in P^4.

4.3 Granular content linking

The final step is to linking the textbooks from Intextbooks with the learning activities from P^4.

Textbook sub-chapters to P^4 First, a list of the concepts introduced in each topic is extracted from P^4. Then, for each textbook, the linked glossary terms to Python concepts are used to get the individual index terms in the textbook associated with the external concepts. After that, we get from the knowledge models the sub-chapters associated with the index terms. Now, we have sub-chapters mapped to index terms, index terms mapped to concepts, and concepts mapped to topics. Finally, using the concepts as a bridge, we link each sub-chapter to the topics in P^4 where the Python concepts are introduced. This sub-step in represented with the number 3.1 in Fig. 3.

P^4 learning activities to textbook sub-chapters In P^4, each learning activity has a set of associated concepts, plus the topic where it is used. We first extract this list of learning activities and compute the topics that are prerequisites using the associated concepts and the topics where they are introduced. Then, for each sub-chapter linked to a topic in P^4, we select as candidates the learning activities that belong to the same topic and have one of the concepts associated with the sub-chapter. Then, we check that the topic prerequisites for each learning activity have been introduced in other previous sub-chapters. If all

the prerequisites are met, we create a sub-chapter-learning activity pair. This sub-step is represented with the number 3.2 in Fig. 3.

5 Results

We applied our methodology using 5 different Python textbooks ([19, 24, 32, 41, 46]) and the mentioned Python ontology. In this section, we describe and analyze the obtained data for each step of the methodology.

Knowledge model extraction and glossary unification After creating the knowledge models for the textbooks, we got a variate number of index terms in each one (817, 848, 834, 451, 1105), producing a total of 4055 different elements. After merging the terms, we got a unified glossary with 3250 elements (a reduction of almost 20%). Additionally, when processing the Python ontology, we got 108 elements: 73 leaf concepts and 35 inner classes.

Glossary - ontology linking After linking the terms in the glossary to the elements in the Python ontology, we got 53 instances. Some terms in the glossary were linked to the same Python concept. Thirty-six concepts and ten classes from the Python ontology were linked to the 53 terms in the glossary.

Granular content linking At the final step of the methodology, we got multiple contents linked in both systems. First, 266 different index terms from the five textbooks were linked to 217 concepts and 49 classes from the Python ontology. Of those 266 terms, only 186 are linked to concepts used in P^4, since not all the Python concepts from the ontology are currently a part of the system. Using the linked index terms and the Python concepts, 245 different sub-chapters from all the textbooks were mapped to the topics in P^4. The number of linked sub-chapters for each topic is as follows: 36 to 'Variables and Operations', 35 to 'Boolean Expressions', 12 to 'If-Else', 3 to 'While Loops', 19 to 'For Loops', 40 to 'Functions', 49 to 'Lists', 11 to 'Dictionary', 17 to 'Strings', 3 to 'File Handling', 5 to 'Exceptions', and 15 to 'Classes Objects'. Regarding linking learning activities to sub-chapters, 2240 possible mappings were analyzed, from which 1790 fulfilled the prerequisite restrictions. If we only assign each learning activity once in the whole textbook, instead of to multiple sub-chapters, the average of unique learning activities mapped per textbook is 67, from a total of 157 different activities available in P^4.

General Analysis The obtained data shows that despite the small number of glossary terms that are matched to Python concepts (53), the number of both linked sub-chapters to topics (245) and the learning activities to sub-chapters (1790) is promising. P^4 can benefit from incorporating textual material explaining the concepts used in each topic. Additionally, multiple textbooks enable

more personalization: a student can select one or more textbooks to get the textual material recommendations according to their preferences. Intextbooks can present additional interactive content to the learners as they progress and navigate throughout a textbook.

Currently, we link the same learning activity to all the fitting sub-chapters within a textbook, which will cause problems if the textbook is read sequentially. This approach has been used because Intextbooks could generate personalized navigation paths for each student. Hence, it is helpful to have an extensive mapping of learning activities. The system will need to be aware of this situation and not display learning activities that have been seen already in previous sub-chapters. Finally, the order of topics and the association of concepts to learning activities in P^4 allows the comparison of prerequisite-outcome relations with the textbooks. As we found, the order of topics in four of the five textbooks was similar to the one in P^4. However, textbook [46] was an exception, producing no links to learning activities since the required prerequisites were not introduced before, or not at all, in the textbook. Since the purpose of the textbook was not to teach Python but to use it for data science, the author assumes familiarity with the language, resulting in fewer programming concepts being introduced.

6 Validation

In order to assess the automatic concept linking approach described above, two domain experts with experience on teaching Python independently rated the match quality (appropriateness) of the textbook sections conceptually linked to each of the topics presented in the P^4 system. The following rating schema was used: 3 as a good match, 2 as a partial match, and 1 as a bad match. 55 sections of the *"Python for Everybody"* textbook were automatically associated to one of the 17 topics of the Python course. Note that the topical structure of that Python course was defined by Python instructors from several universities, who used P^4 systems in to support Python practice in their courses. After the scoring, the results were examined to determine possible causes for the discovered bad matches. The examination revealed that a number of low matching scores were produced by the sections titled as "Glossary" that were included in each chapter rather than assembled at the end of the book in a more traditional way. The nature of these sections make them poor independent learning resources since they served as a reminder of already learned content. We decided to exlude these sections from matching and evaluation.

After glossaries were removed, inter-reliability between raters was calculated. using weighted Cohen's Kappa. The resulting Kappa 0.63 is considered as moderate inter-reliability. Given that raters did not have made ratings fully independently and that they only had access to a very short and concise description of the rating schema, this result is deemed as positive. The main disagreement was registered in the topics of *Boolean Expressions*, and *Lists/ Strings*. The point of disagreement here was that in the textbook some chapters introduce some concepts blended together (e.g., *boolean operators* and *if statement*) while in the P^4

course they are clearly separated and taught one after another (i.e., first *boolean expressions*, and then *if-else*). In terms of general mutual agreement, both raters coincided in thinking that there was a good match in 45% of the textbook associations, a partial match in 12% of the cases and a bad match in 14% of the total linkages. Considering the evaluations that lead to disagreement (29%), 17% involved positive evaluations (either one of the two ratings as 3 or 2), while only a 12% lead a bad rating for the match. Finally, in total, 74% of raters' pairs of evaluations were at least either partial or good matches, so we can conclude that the automatic linking approach that we followed lead to acceptable good results which can be used in the context of automatic integration of learning content.

7 Discussion and Future Work

In this paper we demonstrated that a two-way linking between textbook sections and smart learning content items could be generated by automatic extraction of concepts from programming textbooks using a combination of textbook metadata and different ontologies as underpinning tools for this task (e.g., DBPedia, Python ontology). We consider our work as the first step to resolving the problem of automatic linking and plan to continue research in this direction.

Given that the textbook concept extraction works by analyzing the textual content, a natural next step would be exploring a more "fine-grained" association of concepts, e.g., at a paragraph level rather than on section level. This approach will enable us to recommend practice learning material to the users "in context", right after the student reads the corresponding lines where a concept is introduced. In a similar way, given that the textual information presented in the textbook generally focuses on presenting the concepts in an introductory way, a more "fine-grained" association in terms of textual units will enable us to recommend remedial reading when students fail in certain learning activities which could reflect the learner's misconception(s) on certain concept(s).

One potential problem of the proposed methodology is that the index terms from the textbooks are not always a high-quality representation of a domain. They can potentially suffer from high subjectivity, poor coverage and granularity, lack of semantics, and ambiguity. For example, in one of the textbooks used, the term "tuple" is used both to refer to the Python data type and a database tuple. Another problem specific to the programming domain is how to differentiate reserved keywords from natural language in the content accurately (e.g., for "if" or "or"). Finally, although the used knowledge models have been created for different domains (statistics, history, computer science, literature, and information retrieval [7]), the integration with other types of content than the one used in this research is an exciting path to follow.

To more reliably assess the value of our current approach and its suggested extensions, each linking approach and its interface implementation has to be evaluated in classroom studies from the prospect of pedagogical usefulness and quality of integration between online textbooks and online practice systems.

References

1. ADL Initiative: Shareable content object reference model. technical specification (4th ed.). https://www.adlnet.gov/assets/uploads/SCORM_2004_4ED_v1_1_Doc_Suite.zip (2004)
2. ADL Initiative: Experience API. https://github.com/adlnet/xAPI-Spec/blob/master/xAPI-About.md (2017)
3. Agrawal, R., Gollapudi, S., Kannan, A., Kenthapadi, K.: Study navigator: An algorithmically generated aid for learning from electronic textbooks. Journal of Educational Data Mining **6**(1) (2014)
4. Alpizar-Chacon, I., van der Hart, M., Wiersma, Z.S., Theunissen, L., Sosnovsky, S.: Transformation of pdf textbooks into intelligent educational resources. In: Proceedings of the Second Workshop on Intelligent Textbooks. vol. 2674, pp. 4–16. CEUR-WS (2020)
5. Alpizar-Chacon, I., Sosnovsky, S.: Expanding the web of knowledge: One textbook at a time. In: Proceedings of the 30th ACM Conference on Hypertext and Social Media. p. 9–18. HT '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3342220.3343671
6. Alpizar-Chacon, I., Sosnovsky, S.: Order out of chaos: Construction of knowledge models from pdf textbooks. In: Proceedings of the ACM Symposium on Document Engineering 2020. DocEng '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3395027.3419585
7. Alpizar-Chacon, I., Sosnovsky, S.: Knowledge models from pdf textbooks. New Review of Hypermedia and Multimedia pp. 1–49 (2021)
8. Anderson, J., Reiser, B.: The lisp tutor. Byte **10**(4), 159–175 (1985)
9. Barria-Pineda, J., Brusilovsky, P., He, D.: Reading mirror: Social navigation and social comparison for electronic textbooks. In: Proceedings of the First Workshop on Intelligent Textbooks co-located with 20th International Conference on Artificial Intelligence in Education (AIED 2019), Chicago, IL, USA, June 25, 2019. CEUR Workshop Proceedings (2019)
10. Boyle, T., Gray, G., Wendl, B., Davies, M.: Taking the plunge with clem: the design and evaluation of a large scale cal system. Computers and Education **22**(1/2), 19–26 (1994)
11. Brown, M.H., Najork, M.A.: Collaborative active textbooks. Journal of Visual Languages and Computing **8**(4), 453–486 (1997)
12. Brusilovsky, P.: Knowledgetree: A distributed architecture for adaptive e-learning. In: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters. pp. 104–113 (2004)
13. Brusilovsky, P., Edwards, S., Kumar, A., Malmi, L., Benotti, L., Buck, D., Ihantola, P., Prince, R., Sirkiä, T., Sosnovsky, S., et al.: Increasing adoption of smart learning content for computer science education. In: Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference. pp. 31–57 (2014)
14. Brusilovsky, P., Schwarz, E., Weber, G.: ELM-ART: An intelligent tutoring system on world wide web. In: Frasson, C., Gauthier, G., Lesgold, A. (eds.) Third International Conference on Intelligent Tutoring Systems, ITS-96. Lecture Notes in Computer Science, vol. 1086, pp. 261–269. Springer Verlag (1996)
15. Brusilovsky, P., Schwarz, E., Weber, G.: Electronic textbooks on www: from static hypertext to interactivity and adaptivity. In: Khan, B.H. (ed.) Web Based Instruc-

tion, pp. 255–261. Educational Technology Publications, Englewood Cliffs, New Jersey (1997)
16. Brusilovsky, P., Sosnovsky, S., Shcherbinina, O.: User modeling in a distributed e-learning architecture. In: International conference on user modeling. pp. 387–391. Springer (2005)
17. Chaudhri, V.K., Cheng, B., Overtholtzer, A., Roschelle, J., Spaulding, A., Clark, P., Greaves, M., Gunning, D.: Inquire biology: A textbook that answers questions. AI Magazine **34**(3), 55–72 (2013)
18. Dolog, P., Nejdl, W.: Semantic web technologies for the adaptive web. In: The adaptive web, pp. 697–719. Springer (2007)
19. Downey, A.: Think Python. Green Tea Press, 2nd edn. (2015)
20. Edwards, S.H., Murali, K.P.: Codeworkout: Short programming exercises with built-in data collection. In: 2017 Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE'17. pp. 188–193. ACM Press (2014)
21. Ericson, B.: An analysis of interactive feature use in two ebooks. In: Proceedings of the First Workshop on Intelligent Textbooks co-located with 20th International Conference on Artificial Intelligence in Education (AIED 2019), Chicago, IL, USA, June 25, 2019. CEUR Workshop Proceedings, vol. 2384, pp. 4–17. CEUR-WS.org (2019), http://ceur-ws.org/Vol-2384/paper01.pdf
22. Guerra, J., Hosseini, R., Somyurek, S., Brusilovsky, P.: An intelligent interface for learning content: Combining an open learner model and social comparison to support self-regulated learning and engagement. In: Proceedings of the 21st international conference on intelligent user interfaces. pp. 152–163 (2016)
23. Guerra, J., Sosnovsky, S., Brusilovsky, P.: When one textbook is not enough: Linking multiple textbooks using probabilistic topic models. In: Hernández-Leo, D., Ley, T., Klamma, R., Harrer, A. (eds.) Scaling up Learning for Sustained Impact. pp. 125–138. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
24. Guttag, J.: Introduction to computation and programming using Python, Revised and Expanded Edition. The MIT Press (2013)
25. Hardt, D., et al.: The oauth 2.0 authorization framework (2012)
26. 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 2013). pp. 60–63 (2013)
27. Hughes, J., Cantor, S., Hodges, J., Hirsch, F., Mishra, P., Philpott, R., Maler, E.: Profiles for the oasis security assertion markup language (saml) v2. 0. OASIS standard (2005)
28. IEEE: IEEE standard for learning object metadata. https://standards.ieee.org/standard/1484_12_1-2020.html (2020)
29. IMS Global: Learning tools interoperability core specification. https://www.imsglobal.org/spec/lti/latest/ (2019)
30. IMS Global: Caliper analytics specification. https://www.imsglobal.org/spec/caliper/latest/ (2020)
31. Kay, J., Kummerfeld, B., Lauder, P.: Personis: a server for user models. In: International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems. pp. 203–212. Springer (2002)
32. Kazil, J., Jarmul, K.: Data wrangling with Python. O'Reilly (2016)
33. Liesaputra, V., Witten, I.H.: Seeking information in realistic books: A user study. In: Joint Conference on Digital Libraries, JCDL 2008. pp. 29–38 (2008)
34. Linn, M.: How can hypermedia tools help teach programming. Learning and Instruction **2**, 119–139 (1992)

35. Meng, R., Han, S., Huang, Y., He, D., Brusilovsky, P.: Knowledge-based content linking for online textbooks. In: 2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI). pp. 18–25. IEEE (2016)
36. Naps, T.L., Eagan, J.R., Norton, L.L.: Jhave – an environment to actively engage students in web-based algorithm visualizations. In: Thirty-first SIGCSE Technical Symposium on Computer Science Education. vol. 32, pp. 109–113. ACM Press (2000)
37. Parsons, D., Haden, P.: Parson's programming puzzles: a fun and effective learning tool for first programming courses. In: Proceedings of the 8th Australasian Conference on Computing Education-Volume 52. pp. 157–163 (2006)
38. Porter, M.F.: Snowball: A language for stemming algorithms (2001)
39. Rada, R.: Converting a textbook to hypertext. ACM Transactions on Information Systems **10**(3), 294–315 (1992)
40. Sakimura, N., Bradley, J., Jones, M., De Medeiros, B., Mortimore, C.: Openid connect core 1.0. The OpenID Foundation p. S3 (2014)
41. Severance, C.R.: Python for everybody exploring data using Python 3 (2016)
42. Shaffer, C.: Opendsa: An interactive etextbook for computer science courses. In: Proceedings of the 47th ACM Technical Symposium on Computing Science Education. pp. 5–5. ACM (2016)
43. Sosnovsky, S., Dolog, P., Henze, N., Brusilovsky, P., Nejdl, W.: Translation of overlay models of student knowledge for relative domains based on domain ontology mapping (2007)
44. Sosnovsky, S., Hsiao, I.H., Brusilovsky, P.: Adaptation "in the wild": ontology-based personalization of open-corpus learning material. In: European Conference on Technology Enhanced Learning. pp. 425–431. Springer (2012)
45. Trella, M., Carmona, C., Conejo, R.: Medea: An open service-based learning platform for developing intelligent educational systems for the web. In: Workshop on Adaptive Systems for Web-Based Education: tools and reusability (AIED'05). pp. 27–34 (2005)
46. Vanderplas, J.T.: Python data science handbook: essential tools for working with data. O'Reilly, 1st edn. (2017)
47. Weber, G.: Cognitive diagnosis and episodic modelling in an intelligent lisp-tutor. In: C., F. (ed.) Intelligent Tutoring Systems. pp. 207–214. Univ. of Montreal, artc106 (1988)
48. Winchell, A., Mozer, M., Lan, A., Grimaldi, P., Pashler, H.: Can textbook annotations serve as an early predictor of student learning? In: the 11th International Conference on Educational Data Mining. pp. 431–437 (2018)
49. Yudelson, M., Brusilovsky, P.: Navex: Providing navigation support for adaptive browsing of annotated code examples. In: Looi, C.K., McCalla, G., Bredeweg, B., Breuker, J. (eds.) 12th International Conference on Artificial Intelligence in Education, AI-Ed'2005. pp. 710–717. IOS Press (2005)