TEXTBOOK DEPENDENCY WEB

An Undergraduate Research Scholars Thesis

by

JOSEPH MARTINSEN

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Philip B. Yasskin

May 2019

Major: Computer Engineering

TABLE OF CONTENTS

ABSTRACT

Textbook Dependency Web

Joseph Martinsen
Department of Computer Engineering
Texas A&M University

Research Advisor: Dr. Philip B. Yasskin
Department of Mathematics
Texas A&M University

After a textbook has been written and published, one may want to customize it for a particular audience; it may be desirable to delete or reorder some of the chapters or sections. However, there may be dependencies among the chapters, sections, examples and exercises which make it very tedious to rearrange the order of not only the chapters but also, the associated exercises. Martinsen has defined a structure to describe the dependencies among the chapters and sections in a portion of the online Calculus book, MYMathApps Calculus, being written by Dr. Yasskin and Dr. Meade. Further, he has built a GUI for an instructor or institution to reorder the chapters and sections by drag and drop consistent with the required dependencies as specified by the original author.

DEDICATION

To. . .

ACKNOWLEDGMENTS

Akash,

NOMENCLATURE

Editor	A particular instructor, representative of the adopting institution or publisher who wants to reorder the textbook
DAG	Directed Acyclic Graph
Net	Refers to the DAG structe use for dependencies
Topic	The content contained in a unit
Unit	Book/Part/Chapter/Section//Cul-de-sac/Page providing the content of the book
Assessment	A tutorial/exercises providing a problem that is given to a student to solve
GUI	Graphical user interface
Graph Database	A particular database utilizes a graph structures for queries with nodes, edges and properties to represent and store data

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION

1.1 Background

Textbooks go through a long and arduous process before a student or professor is able to view and use it. This process requires much work and effort into not only validating the content of the textbook but also validating the ordering of the textbook as whole. Much like a jigsaw puzzle, each chapter fits one after another based on the dependency of the topics being taught. On top of these chapters being ordered, the exercises must also be placed in the correct place in order to not give an exercise that is based on a topic that has not been presented to the reader previously.

After all this work on ordering has been completed (among other meticulous things), the textbook is finally ready for publication. As with many good textbooks, many professors and institutions may enjoy the content within the textbook but would prefer delivering the content to a student in a different order than the current order of the textbook. Most of the time, the work and effort that has gone into meticulously ordering the chapters and exercises must now be revisited again and modified. This process is nearly as time consuming and laborious as the first going through this process and is frequently done imperfectly.

1.2 Resulting Platform

In order to achieve, a full stack web application was built. The integration points from the textbook to the application consists of five networks. In the current version of the platform three of the networks into the platform is all that is required for the structures and dependencies of any given textbook to be fully understood and used.

The next piece of the platform is the most important. It the actual piece of the platform

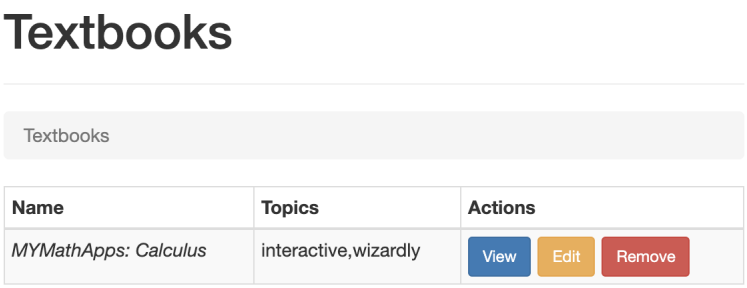

Figure 1.1: The selection and upload of networks for a given textbook

that allows a consumer to reorder and restructure the original ordering of the textbook. It is fully interactive and draggable. Each unit and sub unit can be toggled to view more or less about units. There is also a button to toggle all units to be viewed or to collapse all units until only the independent root units are viewable.

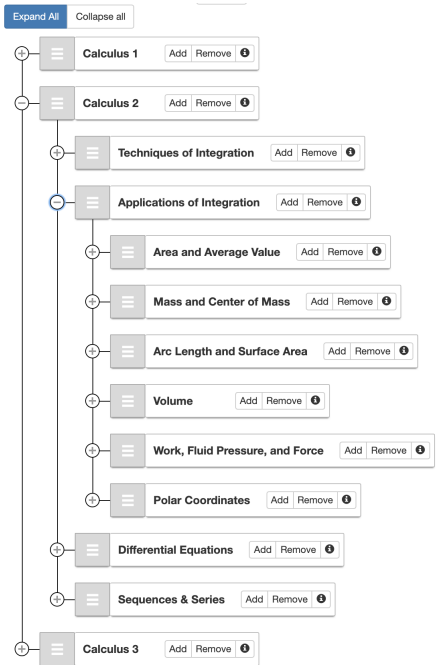

Figure 1.2: Unit hierarchy.

On top of being toggleable, each unit is draggable Fig 1.3. This allows for each unit to be moved to any other point in the structure of the tree or table of contents. During each of these movements, a request is sent to the server to check the and verify that all dependencies are met. If any dependency is not met, a message is then sent to the front end for it to be displayed to the user.

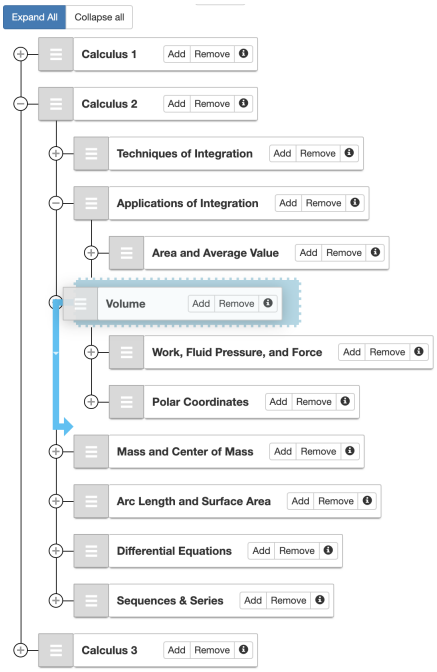

Figure 1.3: Reordering of units through the use of dragging.

2. LITERATURE REVIEW

2.1 Existing Technology

In the current field, most re-orderings of chapters and exercises are done by hand by either the textbook authors or an editor who is either a particular instructor, a representative of the adopting institution or publisher. An exception is TopHat™ who, for their interactive textbooks, has the ability to reorder in a drag and drop manner but one lacking feature is feedback of any sort being provided to the editor [1]. I was able to test drive this product and it felt much like changing the order of a power point presentation as it is in its current state. Besides the technology features that TopHat offers on the rest of their platform, which is outside the scope of this discussion, much of what their reordering seems to achieve can simply be done by reordering the pages of the textbook in some sort of PDF viewer.

From speaking with several Math professors at the Joint Math Meeting 2019, this idea of being able to reorder structural components of their textbook is an issue they commonly face and is not brand new. None of the professors I interacted with were aware of a simple and useful tool that would help them complete this task. There was one platform for identifying dependencies between subjects and textbooks that then allows a professor to generate a curriculum based off these topics, but this is done on a much higher textbook level, not by chapter, section or exercise.

2.2 Approach

My approach to this situation is similar approach to that used by TopHat but builds heavily on the fact that the original author has an idea of the flow of a textbook, including the repercussions that may result from changes to the order. While it is not feasible for a

professor to provide this feedback in person, the realtime feedback aspect can be preserved by creating a platform that *understands* the original authors concerns and then provides them to the editor of the textbook as soon as they make may result in possible conflict of dependencies.

There are existing papers on utilizing trees and identifying the differences between two different trees. One state of the art algorithm identifies differences then generates the minimum of differences from one tree to another [2][3]. An adaptation of this algorithm was utilized by the React core team at Facebook [4]. This algorithm was used under the basis of two different assumptions. One was "two elements of different types will produce different trees" and the other was "The developer can hint at which child elements may be stable across different renders with a key prop" [4]. While the diffing of trees is not a major portion of the design, the idea of topics instead of using key props is utilized for efficiently resolving dependencies while using a depth first search that validates dependencies on nodes already visited.

3. THE ALGORITHM

3.1 Introduction

The core portion of the platform that will allow reordering of aspects of the textbook can simply summed up as *The Algorithm*. This underlying algorithm is independent of the technology or implementation. It is simply a high level description and analysis of the proposed solution to the given problem.

With an understanding of the algorithm, the implementation should follow in a natural and simple manner.

3.2 Motivation for the Algorithm

The algorithm has other items to consider outside simply just solving the given issue. As with most algorithms, there is a thought and focus on completing the desired task in an efficient and optimal manner. This means that if the algorithm is able to provide feedback but if it is done in an clunky manner, the algorithm is still considered a failure and not useful.

In addition to efficiency, correctness is another major point of focus. In this context, correctness relates to properly conveying the thoughts and concerns of the original author to the editor trying to modify the order of a textbook. As a result of a particular action or modification by the consumer, there should never arise a situation where a suggestion or warning provided by the algorithm conflicts with the thoughts or viewpoints of the original author of the textbook. These messages provided by the algorithm should be simply an extension of the original if not exactly the same as if the author was at the computer sitting with the user of the platform.

Correctness also relates to truly reordering the textbook as desired and specified by the

one who modified the order of the textbook. After a change by the consumer, they should be able to clearly understand what type of change they are proposing, how this affects the textbook as a whole, how nearby sections or chapters may be affected and finally after committing these changes, the actual textbook should properly reflect these changes as desired by the consumer.

3.3 The Design

3.3.1 Structure

The design first has to answer the question of what. What information is necessary to properly accomplish the given task. At the root of everything, there is the interdependency of topics. This is the core dependency upon which all other dependencies are built upon or derived from. All the following data types all directly or indirectly are tied to a particular topic or group of topics. A unit is very generic and could designate several different types. A unit is either a book, part, chapter, section, page or cul-de-sac. For the purpose of dependency mapping, there has not been a use case that distinctly separates the types of units for the purpose of dependency mapping. So they have been simply grouped together into type unit. An assessment is a tutorial or exercise providing a problem that is given to a student to solve. These are different from the units because of how they are presented and how the underlying content is used. A unit is simply a presentation of content and/or topics to the reader. An assessment is a formal way of allowing students to practice their skills and test the students knowledge. Another special consideration of assessments is their dynamic and flexible nature. Unit structure is what the editor will likely spend a majority of time reordering and tweaking. Once the unit structure is in place, the assessments should automatically reorder and populate to the correct units given the new unit dependency mapping. For the purpose of relating these two, a unique id is assigned to each topic.

The next topic of focus is how to properly structure the necessary information to com-

plete the given task. Since this is in fact a hierarchical dependency mapping problem at its core, a directed acyclic graph (DAG) was chosen to map and store the dependencies. The reason being because a given topic can have one of two true relations and one metarelation with any another topic. A given topic may depend on another topic because material must first be introduced in the latter topic in order to properly deliver the content in the former topic. This is an example of a child relationship. The reverse relation is also a relationship. This reverse relationship is an example of a parent relationship Fig 3.1.

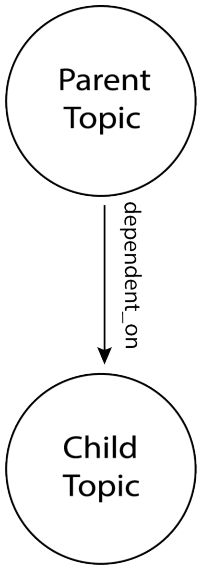

Figure 3.1: Parent child topics.

Finally, the metarelation is the situation where two topics are located on similar levels of the hierarchy in the dependency mapping and do not have a parent or child relationship with one another. These topics are independent of one another, meaning that the order of these two topics in no way affect one another Fig 3.2.

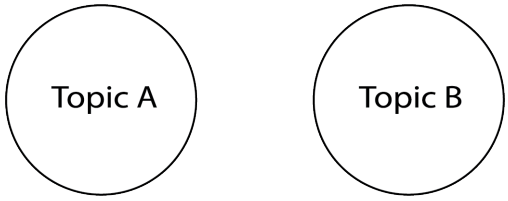

Figure 3.2: Unrelated topics. Can have any number of parents and children but do not relate to one another.

The previous dependency mapping completely describes the requirements for mapping topics but units and assessments require more mappings.

Units have two relationships to map too. A particular unit can introduce one or more topics. A given unit then depends on N topics and M units. A list of unique ids of these N topics and M units are stored on the unit node Fig 3.3.

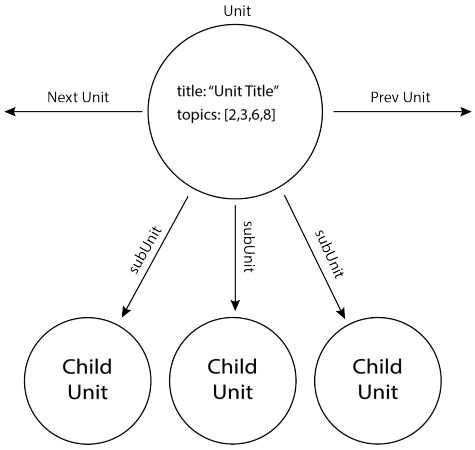

Figure 3.3: Unit mapping.

The final mapping is for the assessment data type. Assessments have three different possible data types it can depend on. They can depend on a topic, a unit or another exercise. The topic and unit dependency should ideally be the same but for our purposes,

both must be satisfied in order for this dependency to be satisfied. In the absence of one, specifically the topic dependency, the other can be used. This is because if the topic dependency is missing for an exercise but unit dependencies are listed, the topic dependency can be generated by traversing the unit dependency mapping for the given units the exercise depends on. The final dependency on exercises is independent of topic or unit. The situation for exercise dependency arises when the results of one exercise are used in another exercise. In this situation, a given exercise should follow immediately after the dependent exercise or after another exercise that shares the same dependency on the same exercise.

Outside of dependency mapping, there is the actual order of the textbook and order of exercises. They can be contained with a DAG much like all the other mappings. These are what actually describes how the textbook will actually be ordered. These mappings are the only ones that are mutated for the sole purpose of reordering a textbook. There is a base mapping that is initially defined by the author for the first publication of the book. These base mappings should be structured in a way such that all dependencies are satisfied.

Finally it must be formally defined what *satisfying all dependencies* mean. In short, a unit or assessment satisfies it's dependencies when its position in the book occurs after all of its dependencies. The book satisfies all of its dependencies when all of its units and assessments satisfy their dependencies.

3.3.2 Flow

For a textbook that is going to be integrated into this dynamic modification flow, several dependency trees/DAGs need to be laid out describing the textbook. Specifically the different orderings as defined by the original author and a consumer and how they interact with one another must be laid out. They are as follows.

1. Net of topic interdependency (created by author)

2. Net of unit interdependency + topic dependency Fig 3.3 (created by author)

3. Net of assessment interdependency + topic or unit dependency (created by author)

4. (a) Net of unit ordering (created by author)

   (b) Net of unit ordering (created by editor)

5. (a) Net of assessment ordering (created by author)

   (b) Net of assessment ordering (created by editor)

The workflow is as follows.

1. Original author creates nets 1, 2 and 3 during or upon completion of writing the textbook. Nets 1, 2 and 3 should not be modified for the current version of the textbook. A modification of nets 1, 2 or 3 should be done to correct a dependency mistake or to add or rewrite a unit or topic. These modifications may result in a new edition of the textbook. These should be checked to see if all dependencies are satisfied.

2. Nets 4a and 5a are automatically generated from nets 1, 2 and 3.

3. The editor specifies the desired ordering of the revised textbook by modifying net 4a to produce net 4b.

   (a) During this process warnings are provided to the editor if any of their modifications in net 4b cause a dependency to not be met as defined in nets 1, 2 and 3.

4. Using nets 1, 2 and 3 as well as the user modified net 4b, net 5b is generated defining the order of the assessments.

   (a) The editor can preview each assessment and modify the order in which it appears.

(b) During this process, warnings may be provided to the user if any of their modifications in net 5b cause a dependency to not be met as defined in nets 1, 2, 3 and 4b.

5. The newly generated order in the form of nets 4b and 5b can then be utilized to modify the text and links in the given textbook.

4. THE IMPLEMENTATION

4.1 The Textbook

For the implementation of the design, I have gone with proof of concept model. Taking a single textbook and applying this workflow for the purpose of working out any workflow issues and testing the algorithm in a real world situation. The textbook that will first be integrated is Dr. Philip B. Yasskin and Dr. Meade Mathematics textbook *MYMACalc: A Web Based, Interactive, Multimedia Calculus*. This textbook was initially written in LaTeX but has since been converted into a web based textbook.

4.2 Textbook Build Process

As aforementioned, the textbook is web based and as such, is developed to conform to web standards. Each page is written in Hypertext Markup Language (HTML). In order to better clarify as well as lighten the load on the author(s) of the textbook, a JavaScript object notation (JSON) file defines the order of the textbook. This order is similar to tree 1 as previously presented. From this ordering, the textbook is then built and structured. This JSON structure allows means that tree 1 is completed. Within tree 1, topics are then manually added to each unit in order to begin the process of building the other trees. Once these topics have been added, tree 3 and 5 can be generated by the implicit ordering defined by tree 1. The topics names are then replaced with ids of the topics. The author is then given the opportunity to then tweak trees 3 and 5 to indicate the correct dependency mapping of these trees.

4.3 The Tech Stack

4.3.1 Database

A graph database is utilized for storing the five different trees. The particular database used was Neo4j. This database allowed for DAGs to be stored in it's entirety. It also allowed for each querying and finding of nodes by simply information about the node. There was also an interface provided by this service that would visually show nodes that are matched by a given query as well as the relationship between each node.

4.3.2 Server

The purpose of the server was to fetch and store data to the database and perform the computational portions of the algorithm. This included identifying warning messages and ensuring that all dependencies are met for each change or modification that a consumer makes.

4.3.3 Client

The client utilized ReactJS. This is front end library that had existing modules that allowed for easily modifying and reordering a JSON tree. This was not a simple implementation so it did require a good amount of tweaking in order for it to fit the necessary use case.

4.4 Integration with Textbook

Full integration with the textbook involves two different parts. One is adding all 5 textbook trees for use by the platform and the other is being able to export the necessary data in order to actually modify the textbook. As mentioned previously, the textbook being used had a good portion of tree 1 completed. With some effort and additional mapping, trees 2, 3, 4 and 5 can be constructed.

4.4.1 Import Process

The import process put simply requires the original author to upload all five trees. From there the server is then able to validate and identify if there are any possible dependencies not satisfied in trees 1 and 2. If there are, the author is then provided messages as to what dependencies are not met and allows the autor to modify them using the graphical user interface. This graphical user interface is the same as the one the consumer uses for reordering latter on.

4.4.2 Reorder Process

This is the process where the consumer has the control to modify the structure of the units and the exercises. There are two views for accomplishing this. One is focused on simply the unit and the other focuses more on the exercises and the the units they are directly tied to. During each modification or tweak that lasts for longer than two seconds, the new and updated tree is sent to the server for verification. The server then verifies dependencies and identifies if any have been violated. Depending on the order of the violation and where in the tree the violation occurred, a message is generated to inform the consumer of what the violations are and possible suggestions as to how to fix the ordering.

After the consumer has modified the ordering of the units, they are able to review these changes. This review screen highlights changes, additions or deletions of units. This screen allows the user to see all the changes they are attempting to do before they commit them. Once the user approves these changes, this new ordering is then stored as the new ordering for the particular consumer. Along with storing these changes, the consumer is given the option to allow tree 2 to be automatically generated. Regardless of if the user allows for this to happen, they are then taken to the exercise reordering screen. The same reordring for units then follows with warnings and messages appearing as well as

the review screen showing before a user fully commits their ordering changes.

4.4.3 Export Process

Due to the structure and modifications to the build process of the textbook in use, the export process is rather trivial. It is simply an export of trees 1 and 2. With the textbook in use, especially with tree 1, these trees in JSON format are then placed in the appropriate location in the build process. This allows for the existing build process to generate the new ordering of the textbook as specified by the consumer.

5. CONCLUSION

5.1 Challenges

One challenge encountered was the fact that the textbook was already written without this design or thoughts about the algorithm. In order to get the textbook into a place where it could be integrated with the algorithm required some slight modifications and additions. In the algorithm's current state, this is what most textbooks are required to do in order to fully integrate into the platform. This textbook used did have a similar tree structure as tree 1 (the ordering of units). With some modifications of this tree, it became the same format as required.

Another large problem encountered is related to the compounding complexity of the project. Each chapter, section and page was treated as simply as a unit. While for the purposes of dependency mapping, this group was allowable, for the use of the author of the textbook, each entity is different in both information they contain and how they may be treated for the purpose of writing the textbook.

time to work on exercise

5.2 Broader Impact

The main focus of the thesis thus far has been with a single textbook, but a broader goal is to allow any textbook to utilize this platform with the underlying algorithm. Allowing an author to use this, especially during the initial design of the textbook, will allow seamless reordering to the textbook in the event that an adopting institution or professor decides to modify the original ordering of the textbook.

5.3 Future Plans

The integration portion of the platform and textbook has not been entirely completed. There has been tweaking and modifications to the platform, the build ordering of the textbook and easily converting them into to use by the platform. A goal is to have complete integration with the textbook to the point that it can actually be tied into the build process. As a build step, any modifications can be done using the platform and then immediately carried out by producing a new version of the textbook with the appropriate modifications.

time to work on exercise

REFERENCES

[1] "Reordering chapters in your top hat textbook." Web, April 2017.

[2] P. Bille, "A survey on tree edit distance and related problems," *Theoretical Computer Science*, vol. 337, no. 1-3, pp. 217–239, 2005.

[3] D. Tsur, "Faster algorithms for guided tree edit distance," *Information Processing Letters*, vol. 108, no. 4, pp. 251–254, 2008.

[4] B. Vaughn, "Reconciliation." Web, 2019.