Analyzing Student Performance with Personalized Study Path and Learning Trouble Ratio

James Shing Chun Yip   Raymond Chi-Wing Wong
The Hong Kong University of Science and Technology
{scyipaa, raywong}@cse.ust.hk

ABSTRACT

Massive Online Open Courses (MOOCs) have become very popular nowadays and are attracting a lot of attention from both learners and researchers. Learners benefit from their availability of learning materials such as lecture notes, videos and self-tests. However, most MOOCs are a one-size-fit-all model and does not meet the individual needs of learners. For example, each learner is shown a "pre-defined" order of study for each different learning concept in a course. It is observed that different learners have different learning paces, and how good a learner understands one learning concept is heavily dependent on how good s/he understands the prerequisite of this learning concept (which is also a learning concept). Motivated by this, some researchers would like to propose some personalized suggestions for each learner so that they could learn certain concepts in different order. In this paper, we introduce a method for visualizing an ordering of learning concepts concisely and showing how strongly one learning concept is related to another learning concept.

Keywords

Learning Analytics, Visualization, Educational Data Mining

1. INTRODUCTION

Most courses in MOOCs are used to present their curriculum in a chapter-by-chapter format. For example, it can be a list of videos with lecture notes for each chapter in an online platform like Coursera. This design is linear (or predefined) and insufficient for emphasizing the relationships between chapters or concepts. In contrast, our knowledge should be based on prerequisites and even the prerequisites of prerequisites. These relationships recursively form a learning graph where each node corresponds to a learning concept. Then an edge from one node to another node corresponding to that one learning concept is a prerequisite of another learning concept. This graph is actually a directed acyclic graph (DAG) which is able to generate many topological order paths. Note that there are many ways to study all concepts throughout a course.

As a clear learning path enables both instructors and students to improve learning efficiency. This motivates us to visualize the learning path in a clear and simple way. In this paper, we introduce a method of visualizing an ordered learning graph to represent the order of the learning path. In our graph, it contains learning concepts of a course that link together with a number which we call *learning trouble ratio* showing the strength of dependency as well as a weight. This ratio indicates the difficulty of such a pair of concepts for the students to study. It also emphasizes the concepts both students and instructors should pay attention to. Not only that, the ratios for each pair can be dynamically updated by using students' learning performance and the learning graph.

We choose the course "COMP102X Introduction of Java Programming" with a duration of 14 weeks offered by HKUST on Open edX. The course is administered by Professor T.C. PONG since June 2014, as a basis for developing our system and also an extension of a course on Open edX platform. This course is composed of 12 concepts with 91 questions. 10564 unique students, with at least one record of question submission, enrolled in the course. A demonstration (`https://www.youtube.com/watch?v=KdxVHWSMNi4`) and source codes (`https://github.com/shingyipcheung/study-path-backend`) are available. In this paper, our contributions can be summarized as: We proposed a method of visualizing a learning graph which can be "weighted" and "ordered"; We use the data from an existing MOOC to recommend the learning paths to students with the compact visualization; We make use of modern web frameworks and open source libraries to implement the system that is available online; We provide examples to suggest actions for instructors and students based on the visualization.

The remainder of this paper is summarized as follows: In Section 2 and Section 3 we introduce related works and the problem we are focusing on. In Section 4 we explain our methods in details from collection to generate Learning Trouble Ratio and Personalized Path. In Section 5, we demonstrate how to use modern frameworks and libraries to implement the visualization. Finally, we suggest some cases to help the instructor to identify students who are in "trouble" in Section 6, and conclude in Section 7.

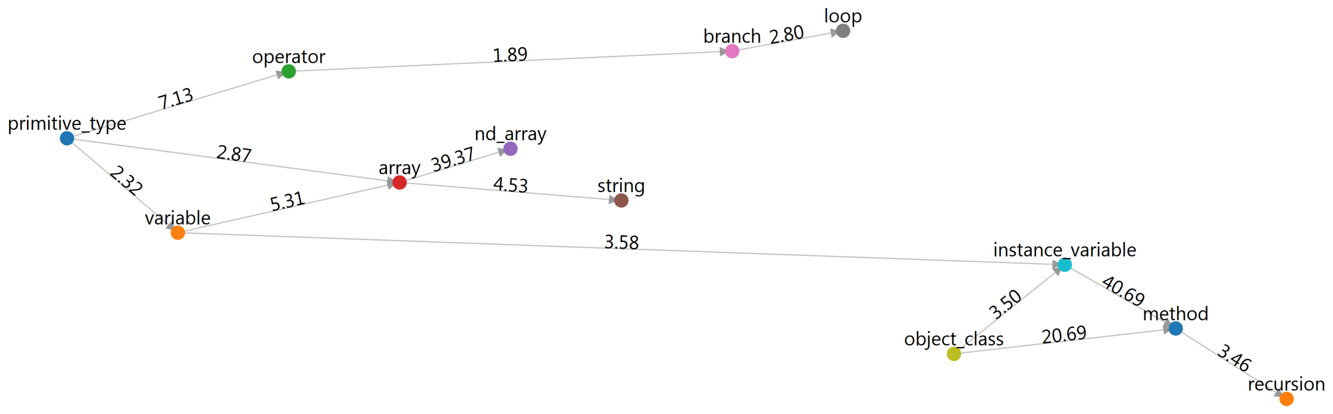


Figure 1: Default ordered learning graph of COMP102X Introduction of Java Programming

2. RELATED WORK

The task of prerequisite relation, to predict whether a concept or skill is a prerequisite of another, has been studied by different approaches. The data are mostly collected from Wikipedia. In [[3], [13], [9]], semantic and video features are designed to compare the relatedness between concepts, and predict the relations by the supervised classification approach. Liang [10] et al. proposed an objective function as a soft margin SVM where each course is represented by the term frequency-inverse document frequency (tfidf) to recover the strength of prerequisite relations. Adjei et al. [1] developed a system to stress the problematic links in a prerequisite skill structure where they used linear regression to find the strength as the coefficients in the equation. Our approach is relatively simpler and more intuitive, we use the assessment results to measure the "troublesome" of the relations by the probability ratio. This measurement also points out some pairs may create problems for students.

For visualizing graphs, previous studies have developed different tools for various applications. North [11] created a visual debugger called dotty which can provide an interactive operation for users. Another paper proposed a visualization method for *Extract Method* refactoring which is for moving a part of the original code to a separate new method in the form of the program dependency graph [7]. A node on the graph is defined as a line of code, and an edge indicates a reference or a definition of a variable. Their work aims to identify any potential nodes that can be refactored without using big data, whereas in our work the goodness assessments of each node are data-driven. In this paper, we focus on visualizing the learning dependency graph containing concepts as nodes which could be helpful for science and engineering subjects because of the high correlation among the concepts.

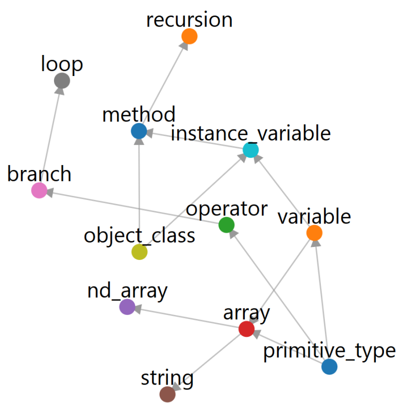
After MOOCs became popular, more studies emerged on visualizing educational data. Qu and Chen [15] pointed out that education stakeholders will benefit from the intuitive visualization to reduce the learning curve. Shi et al. [16] developed a system which contains several views to aid learning analytics by using clickstream data from Open edX. For example, their demonstration provides the graph of a course including the demographic info of students. Although their visualizations aid the instructor to see the overall picture in different kinds of data, it might be too low-level for students to understand what is being presented to them. In contrast, we try to use the learning graph with suggested study pathways to point out which concepts should be learned earlier.

Grosse and Reed [6] created a web application Metacademy for autodidacts to learn science subjects with the help of a learning (concept) graph. Their work is very similar to ours but without the education data provided by MOOC. Further, the application only contains a single learning path handcrafted by the creators for learners to follow. Compared with their application, our work has two major advantages which could not be found in their studies. First, we combine the learning graph structure and MOOC data to provide strength among the concepts - learning trouble ratio which emphasizes the correlations that students should pay attention to. Second, we provide multiple learning paths which are in topological orders. Our graph can be transformed in an ordered way to represent one of the paths.

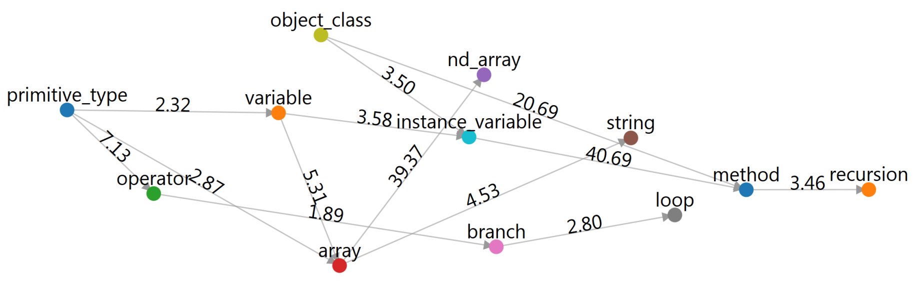
3. PROBLEM DESCRIPTION

We are given a learning graph containing nodes which correspond to a learning concept and edges corresponding to the correlation between two learning concepts. Specifically, we represent a relationship between two (learning) concepts with a dependency. A concept could be dependent on one or more concepts. For example, in Figure 1 (showing the relationship of learning concepts in a Java programming course), we learn `primitive_type` first and then `operator` since `operator` depends on `primitive_type`. We use a directed graph to represent all dependencies among all concepts in a course. In general, each node in a graph is a learning concept. In layman's terms, the graph is a representation of the prerequisites of concepts. The graph is always sparse since there are not many dependencies from one to other learning concepts. Therefore, we use an adjacency list to represent the directed graph. Figure 1 shows the learning graph of the course COMP102X in Open edX offered by Prof. T. C. Pong drawn by *D3.js* [2].

In this paper, we study two problems. The first problem



(a) Initialized with random positions



(b) Drawing from left to right

Figure 2: Learning graphs with crossed edges

is to recommend a personalized study path to each student based on the dependency graph. The second problem is to visualize the personalized study path of each student in a concise way.

In this paper, we plan to show the study path as seen in Figure 1 such that the first learning node in the order is shown in the leftmost place in the graph, the second learning node is shown in the second leftmost place, and so on.

However, how to visualize the study path concisely is a challenge. In Figure 1, we could observe that no edges cross each other. Due to personalization, different nodes may appear in different positions in the graph. Without "careful" design, it is "possible" that one edge may cross another edge. For example, if the node `operator` in Figure 1 is moved to a place under the (virtual) horizontal line passing through the node `variable`, the edge between `variable` and `array` will cross with the edge between the `primitive_type` and `operator`. Drawing directed graphs in a clear way requires different settings such as node positions given the graph structure. Figure 2 shows graphs without any adjustments to the node positions which looks very unclear.

4. METHODOLOGIES

In this section, we propose how to generate the learning trouble ratios and personalized study paths from the student performance and the dependency graph. First, we take the course COMP102X as an example to formulate the score calculation in Section 4.1. Next, we use the scores under such a pair of concepts to quantify its correlation in Section 4.2. The remainder of this part explains the algorithm to generate all topological arrangements given a learning graph in Section 4.3. A fitness function is designed to measure which arrangements fit a student. We also describe how to visualize the directed graph that minimizes the number of crossed edges in Section 4.4.

4.1 Learning Performance

4.1.1 *Student Performance Collection*

Concept	Question id	Weight
`primitive_type`	Q1	1.0
`primitive_type`	Q2	1.0
`primitive_type`	Q3	0.8
`primitive_type`	Q4	1.0
`variable`	Q5	1.0
`variable`	Q6	1.0
`variable`	Q4	0.6

Table 1: Example. Mapping of Concepts to Questions with weights

According to Open edX documentation, the students' learning progress data are stored on the table `courseware_studentmodule`. This table holds the most recent course state, including the most recent problem submission and unit visited in each subsection. We can retrieve information about who answered which question and also the obtained score.

Specifically, we retrieved the score of each question of each student by 5 fields (`module_type`, `module_id`, `student_id`, `grade`, and `max_grade`). `module_id` is the key for each problem. We only consider `module_type` with value problem. The term `max_grade` might be confusing since it denotes the number of sub-problems for a problem. The relative score of a problem is between 0 and 1.

4.1.2 *Problem Weighting*

Instructors could set the weight to a problem for a learning concept. Specifically, each problem is mapped to one learning concept or more. Each learning concept is mapped with at least two questions. An example in Table 1 shows there are 4 problems in `primitive_type` with weights between 0.8 to 1. The problem with id "Q4" is mapped to both `primitive_type` and `variable`.

4.1.3 *Student Score Calculation*

With the problem weight mapping and the progress table from the previous two sections, we use the following formula to calculate the scores of all students.

$$\text{score}_{\text{concept}} = \sum_{p \in \text{concept}} w_p \cdot \frac{\text{grade}_p}{\text{max\_grade}_p} \tag{1}$$

where w_p is the pre-defined weight of the problem p, grade_p and max_grade_p are the field values in Section 4.1.

Each score of a concept is scaled by min-max normalization. Therefore, all scores of the concepts of a student in a course are found.

$$\text{score}_{\text{course}} = \{\text{normalize}(\text{score}_{\text{concept}}) | \text{concept} \in \text{course}\} \tag{2}$$

By this calculation, we found 10564 unique students with at least one score (some of them dropped out) in COMP102X. The mean scores are summarized in Table 2.

concept	mean
`array`	0.712297
`branch`	0.423561
`instance_variable`	0.519257
`loop`	0.521106
`method`	0.235022
`nd_array`	0.312867
`object_class`	0.542028
`operator`	0.375814
`primitive_type`	0.315577
`recursion`	0.748562
`string`	0.572801
`variable`	0.463743

Table 2: mean concept scores in COMP102X

4.2 Strength of Correlation Between Learning Concepts

To estimate the strength of the correlation of each pair in the dependency graph, we adopt the risk ratio [17] as an indicator that we call *learning trouble ratio* which produces the ratio of the probability of the performance between two groups. We define that a student whose score is greater than or equal to the average in a concept as "Above-Average", otherwise they are "Below-Average". In this paper, the average score is equivalent to the mean score, the median may also be applied. There are "Above-Average" and "Below-Average" groups for each concept. We define a pair of concepts (C_{prev}, C_{next}) in the learning graph where C_{prev} is the prerequisite of C_{next}. The numerator on Equation 3 is interpreted as the probability of a student whose result of concept C_{next} is "Below-Average" given that the result of concept C_{prev} is "Below-Average" as well.

$$\text{Learning trouble ratio} = \frac{P(C_{next,below} | C_{prev,below})}{P(C_{next,below} | C_{prev,above})} \tag{3}$$

Students can be below average in a certain concept, however, if the numerator is larger than the denominator, we can infer that it is more likely that a student will have trouble when studying the concept. The higher the ratio, the more important it is that the prerequisite should be paid attention to.

For simplicity, we divide students into 4 groups in Table 3 where each group can easily be found by database query. The ratio is calculated by set operations. For example, $S_{prev,below}$ denotes the set of students who are below average in the preceding concept C_{prev}.

	pair	
	C_{prev}	C_{next}
Below	$S_{prev,below}$	$S_{next,below}$
Above	$S_{prev,above}$	$S_{next,above}$

Table 3: Four sets of students

$$\text{Learning trouble ratio} \\ = \frac{\frac{|S_{prev,below} \cap S_{next,below}|}{|S_{prev,below} \cap S_{next,below}| + |S_{prev,below} \cap S_{next,above}|}}{\frac{|S_{next,below} \cap S_{prev,above}|}{|S_{next,below} \cap S_{prev,above}| + |S_{prev,above} \cap S_{next,above}|}} \tag{4}$$

From Equation 4, we intersect the sets and compute the risk ratio by the cardinalities. For example, assume we have all the students' IDs in $S_{next,below} = \{2, 3, 4\}$ and $S_{prev,above} = \{3, 4, 5\}$. The number of students who are above average in C_{prev} but below average in C_{next}, i.e., $|S_{next,below} \cap S_{prev,above}|$ is 2.

The ratios can also be imported as the weights for each edge in the graph and can be dynamically updated based on the students' performance for an ongoing course (see Figure 1).

4.3 Personalized Path Generation

Our method is to enumerate all possible topological paths based on the given dependency graph $G = (V, E)$ and returns the top 10 paths evaluated by some fitness functions. We use the basic backtracking method to generate all possible arrangements in Algorithm 1. The algorithm begins with initializing all vertices as not visited. It then adds the vertex which is not visited and is of indegree 0 to the result followed by removing all edges pointing from that vertex. Lastly, it calls itself recursively and backtracks. Another efficient algorithm [8] can also be applied.

The fitness functions can be designed for different purposes. One fitness function we used is the ratio relative to the mean score. A concept should be learnt first because the score of the student in that concept is far from being an average student.

$$fit(p, t) = \sum_{i=1}^{|V|} \frac{1}{i} \frac{score_{mean}(p[i]) - score_t(p[i])}{score_{mean}(p[i])} \tag{5}$$

The above equation evaluates a path p for a student t, where $score_{mean}(\cdot)$ denotes the mean score of the concept in the course globally, and $score_t(\cdot)$ denotes the score of the student t in that concept. For example, given two paths.

`array` $\rightarrow$ `branch`

`branch` $\rightarrow$ `array`

A student whose score in `array` and `branch` are 0.6 and 0.5 respectively. From Table 2, we see that the mean score of `array` and `branch` are about 0.7 and 0.4 respectively. The first path gives the fitness $= \frac{1}{1}\frac{0.1}{0.7} + \frac{1}{2}\frac{-0.1}{0.6} \approx 0.05952$, while the second path gives $\frac{1}{1}\frac{-0.1}{0.6} + \frac{1}{2}\frac{0.1}{0.7} \approx -0.09523$. Therefore, the first path should be given higher priority than the recommendation.

Algorithm 1 All Topological Orders

```
Input: a directed graph $G = (V, E)$
Output: a list of paths $P$ in different orders
 1: procedure TOPOLOGICAL_ALL($V, E$)
 2:     $S \leftarrow \emptyset$                ▷ A set to store nodes are not visited
 3:     $P \leftarrow empty$                ▷ A list to store result paths
 4:     $p \leftarrow empty$                ▷ A temporary path
 5:     for each $v \in V$ do
 6:         $indegree[v] \leftarrow 0$
 7:         insert $v$ into S
 8:     for each $(u, v) \in E$ do
 9:         $indegree[v] \leftarrow indegree[v] + 1$
10:     Topological_Recursive($V, E, S, P, p$)
11:     return $P$
12: procedure TOPOLOGICAL_RECURSIVE($V, E, S, P, p$)
13:     if S is empty then
14:         insert $p$ into $P$
15:     else
16:         for each $v \in V$ do
17:             if $v \in S$ and $indegree[v] = 0$ then
18:                 for each $u$ such that $(v, u) \in E$ do
19:                     $indegree[u] = indegree[u] - 1$
20:                 push $v$ into $p$
21:                 remove $v$ from $S$
22:                 Topological_Recursive($V, E, S, P, p$)
23:                 insert $v$ into $S$
24:                 pop the last element from $p$
25:                 for each $u$ such that $(v, u) \in E$ do
26:                     $indegree[u] = indegree[u] + 1$
```

4.4 Visualizing Learning Graph

To draw the learning graph, we employ the web visualization framework *D3.js* [2]. The force simulation module in *D3* is a common feature of visualizing graphs. The module provides multiple forces in the simulation. For example, the centering force for positioning the nodes at the center of the visual space while the "collision" force adds a force to each node with a given radius as a repulsive force to prevent nodes from being close to each other. However, *D3* itself does not handle the crossed edge problem because using the collision force alone is insufficient. To address the problem, we first fix the x-coordinates of all nodes from left to right with the input order. At this time, there are many crossed edges. Next, we adopt an algorithm [5] for drawing directed graphs. It first finds the optimal rank assignments $\lambda(v)$ for each node v by solving the integer program in Equation 6 where $\lambda(u) - \lambda(v)$ represents the length between node v and u, $w(v, u)$ is the weight, and $\delta(v, u)$ is the minimum length constraint input by users.

$$\begin{aligned} \text{minimize} \quad & \sum_{(v,u)\in E} w(v,u)(\lambda(u) - \lambda(v)) \\ \text{subject to} \quad & \lambda(u) - \lambda(v) \geq \delta(v,u), \forall (v,u) \in E \end{aligned} \tag{6}$$

Two nodes with the same rank will have the same x-coordinate if the drawing is from left to right. Next, it uses a heuristic approach that iteratively finds the coordinates to minimize the number of crossed edges of the ranked graph until reaching the maximum number of iterations. This algorithm has been implemented by Dagre [14]. The final graph shows a clear layout of the input order without crossing edges (see Figure 4 in contrast to Figure 2).

5. IMPLEMENTATION

The system is built using two Web frameworks, Django and Vue.js, for backend and frontend, respectively. Django provides us with powerful Python libraries such as Numpy and Pandas for data processing that help to build a web application. The web interface also utilizes *D3.js* for visualization.

Our method has been employed on the system that can generate personalized topological order paths. First, we apply a diversified path algorithm to select the most representative paths [4]. These paths are static to all students. To make them personalized, we use some fitness functions to enhance the result. For example, one fitness function is based on performance. If the student's result is not good, the related concepts should be given a higher weighting. Finally, the pipeline produces the best 10 paths as a result.

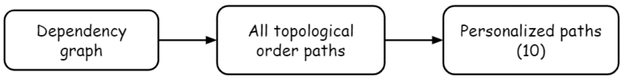


Figure 3: Flow of generating 10 personalized paths

The system has a component called "Recommended Study Paths" showing the 10 "best" paths, which are recommended based on the performance of learning concepts of a student and the dependency strength among the learning concepts. Each of this path is shown in two formats. The first format has a sequential order as shown at the bottom part of Figure 4 (which shows 2 paths, namely Pathway 1 and Pathway 2). The second format is a graph format where the leftmost node in the graph corresponds to the earliest learning concept to be learnt and so on. One path in the second format could be found at the top part of Figure 4 (which shows the second format of Pathway 2).

6. EXPERIMENT

In this section, our goal is to highlight the importance of each component including the learning dependency graph with learning trouble ratios, and the learning pathway.

6.1 Learning Dependency Graph and Learning Trouble Ratio

(C_{prev}, C_{next})	$\lvert S_{prev,below} \cap S_{next,below} \rvert$	$\lvert S_{prev,below} \cap S_{next,above} \rvert$	$\lvert S_{next,below} \cap S_{prev,above} \rvert$	$\lvert S_{prev,above} \cap S_{next,above} \rvert$	Learning Trouble Ratio
(`primitive_type`, `operator`)	1097	1811	115	2059	7.1
(`primitive_type`, `array`)	525	101	554	1342	2.9
(`primitive_type`, `variable`)	2523	94	888	1252	2.3
(`operator`, `branch`)	1206	72	1915	1914	1.9
(`branch`, `loop`)	893	68	639	1286	2.8
(`array`, `nd_array`)	555	520	**19**	1430	**39.4**
(`array`, `string`)	718	62	294	1154	4.5
(`variable`, `array`)	766	279	184	1149	5.3
(`variable`, `instance_variable`)	1123	514	256	1081	3.6
(`object_class`, `instance_variable`)	1024	125	514	1507	3.5
(`object_class`, `method`)	1343	384	76	1946	20.7
(`instance_variable`, `method`)	952	708	**23**	1609	**40.7**
(`method`, `recursion`)	229	20	410	1134	3.5

Table 4: Learning trouble ratios

Since both the learning dependency graph and the learning trouble ratio are presented in the same visualization, they are put together as a component. One might think that we can just make a notice for all students who are below average. However, one major issue in MOOCs is the high dropout rate of learners [12]. Most students treated as bad performers, in our terminology, are "below average" which is not enough to identify the students who really need helps. Owing to this, we focus on the students who are at least making an effort to do some of the exercises but end up achieving a low score. In short, they are the students who are "above" in C_{prev} and "below" in C_{next}, i.e., $S_{next,below} \cap S_{prev,above}$ in Equation 4.

According to Table 4, we can see two pairs (`array`, `nd_array`) and (`instance_variable`, `method`) having a high value of ratio. The trouble ratio not only indicates the strength of dependency but also makes the target students apparent among the crowd. Using the dependency graph with the edge values (learning trouble ratios), we can filter out these students easily, a detailed example is explained in the Case Study.

6.2 Learning Path

Without a suggested learning path, students may randomly pick a concept to study. We expect that this random approach would result in a bad performance, especially if a concept has prerequisites the student has not fulfilled. Therefore, we chose 3 (single-length path) pairs of concepts with the top 3 highest learning trouble ratio for illustration.

`instance_variable` → `method`

`object_class` → `method`

`array` → `nd_array`

We found that students perform badly if they only do the exercises for just 1 concept. From Table 5, we see that all 3 pairs: the score of C_{next} must be less than the that of the prerequisite C_{prev}.

Therefore, students should have better learning if they can follow a suggested path.

6.3 Case Study

concept	mean
`object_class`	0.21
`instance_variable`	0.31
`method`	0.03
`array`	0.18
`nd_array`	0.15

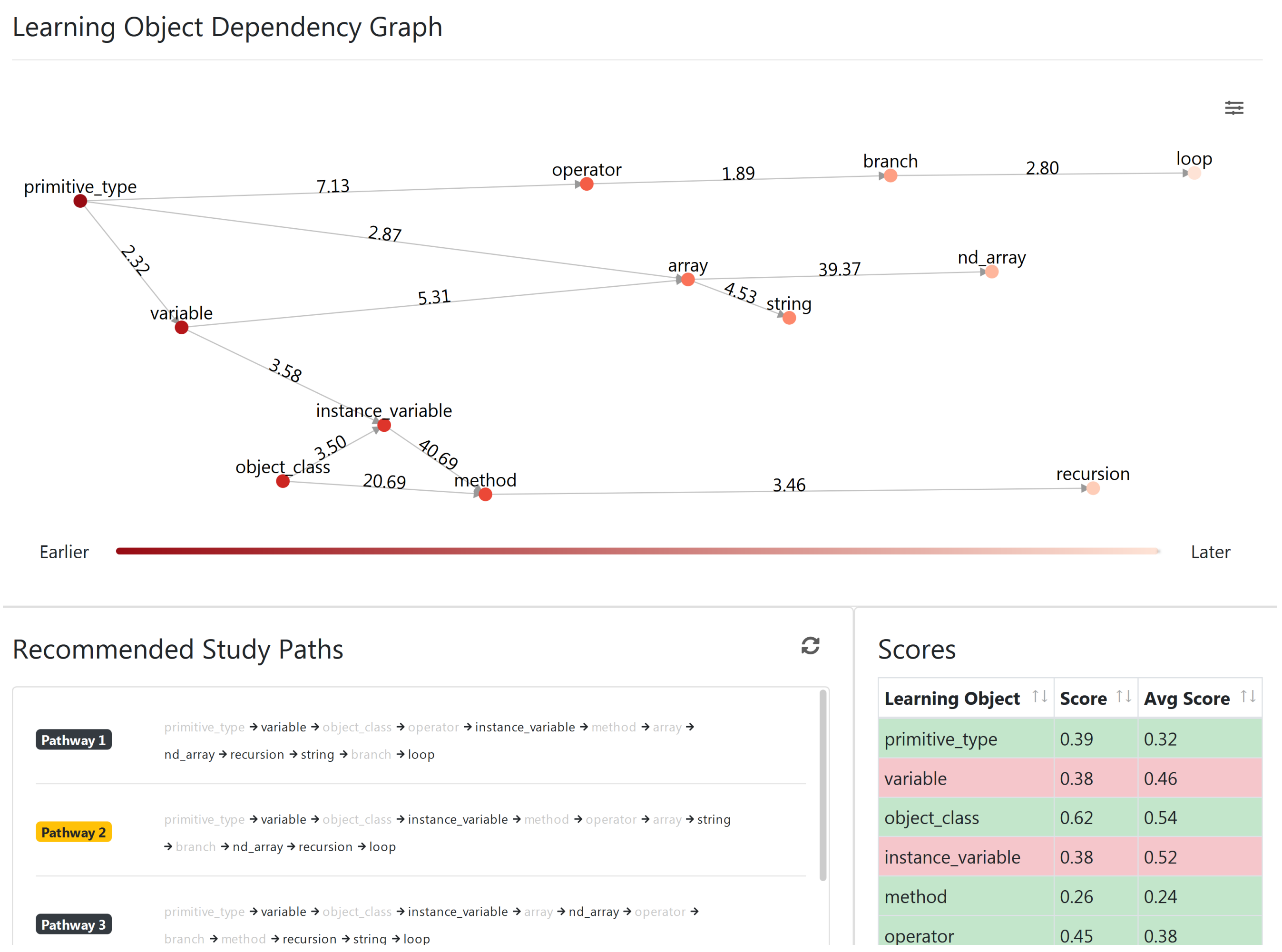
Table 5: Mean scores of students who only work on 1 concept (None for the others)

Since `array` and `nd_array` are very dependent (because the learning trouble ratio between `array` and `nd_array` is of value 39.37 which is high). Our system provides the parallel coordinates plot where users can select multiple score ranges to extract the students.

Figure 5 shows parallel coordinates, a lot of "blue" lines where each connected line corresponds to a student. It also shows a vertical black line for each learning concept. For example, `array` has a vertical line denoting the "score" of a learning concept obtained by a student. If a blue (horizontal) line has a value of 0.8 in the vertical line of `array` and a value of 0.2 in the vertical line of `nd_array`, this means that the corresponding student (for this blue line) has a score of 0.8 for array and a score of 0.2 for `nd_array`.

In this interface, users could also "highlight" some parts of the vertical lines to select all blue lines out. For example, in Figure 5, we tried to select the students whose score is above average in `array` (shown in the shaded region in the vertical line of `array` in the figure) and below average in `nd_array` (shown in the shaded region in the vertical line of `nd_array` in the figure) to check whether there are many such "unexpected" students (since we expected that students who performed well in `array` should perform well in `nd_array`).

Finally, we find 19 students (shown in Figure 6). Most students in this list completed nearly all learning concepts. This shows that they are very hard-working (since most students in MOOCs do not complete nearly all learning concepts). The instructor can suggest the learning paths like Figure 4 to these students for revision or other follow-ups to see how to help these students.



Recommended Study Paths

Pathway 1: primitive_type → variable → object_class → operator → instance_variable → method → array → nd_array → recursion → string → branch → loop

Pathway 2: primitive_type → variable → object_class → instance_variable → method → operator → array → string → branch → nd_array → recursion → loop

Pathway 3: primitive_type → variable → object_class → instance_variable → array → nd_array → operator → branch → method → recursion → string → loop

Scores

Learning Object	Score	Avg Score
primitive_type	0.39	0.32
variable	0.38	0.46
object_class	0.62	0.54
instance_variable	0.38	0.52
method	0.26	0.24
operator	0.45	0.38

Figure 4: The ordered graph is changed accordingly and showing the learning trouble ratios

7. CONCLUSION

We proposed the method to visualize the "weighted" and "ordered" learning graph that has been applied to the recommendation system. It is capable of extracting the data from the Open edX platform from time to time and performing various data mining and visualization techniques. In conclusion, we have made a prototype to achieve improved learning by personalized study plans alongside visualizing an ordered learning graph. We hope that our development can be further customized and integrated with most MOOCs.

ACKNOWLEDGEMENTS

The research is supported by ITS/227/17FP.

8. REFERENCES

[1] S. A. Adjei, A. F. Botelho, and N. T. Heffernan. Predicting student performance on post-requisite skills using prerequisite skill data: an alternative method for refining prerequisite skill structures. In *Proceedings of the sixth international conference on learning analytics & knowledge*, pages 469–473. ACM, 2016.

[2] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. 2011.

[3] C. De Medio, F. Gasparetti, C. Limongelli, F. Sciarrone, and M. Temperini. Automatic extraction of prerequisites among learning objects using wikipedia-based content analysis. In *International Conference on Intelligent Tutoring Systems*, pages 375–381. Springer, 2016.

[4] M. Drosou and E. Pitoura. Multiple radii disc diversity: Result diversification based on dissimilarity and coverage. *ACM Trans. Database Syst.*, 40:4:1–4:43, 2012.

[5] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Trans. Software Eng.*, 19:214–230, 1993.

[6] R. Grosse and C. Reed. Metacademy. `https://github.com/metacademy/metacademy-application`, 2013.

[7] T. Kanemitsu, Y. Higo, and S. Kusumoto. A visualization method of program dependency graph for identifying extract method opportunity. In *WRT@ICSE*, 2011.

[8] D. E. Knuth and J. L. Szwarcfiter. Erratum: A structured program to generate all topological sorting arrangements. *Inf. Process. Lett.*, 3:64, 1974.

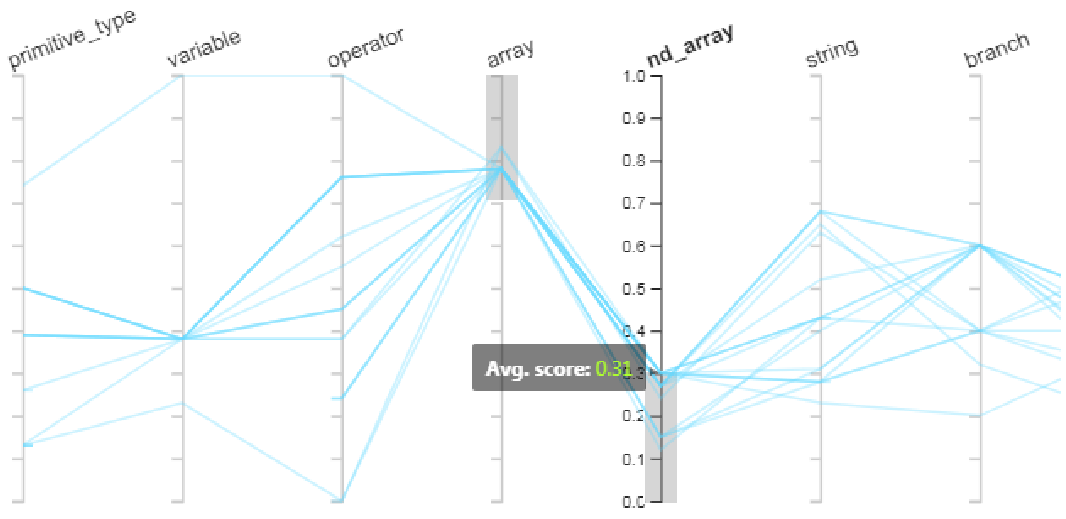
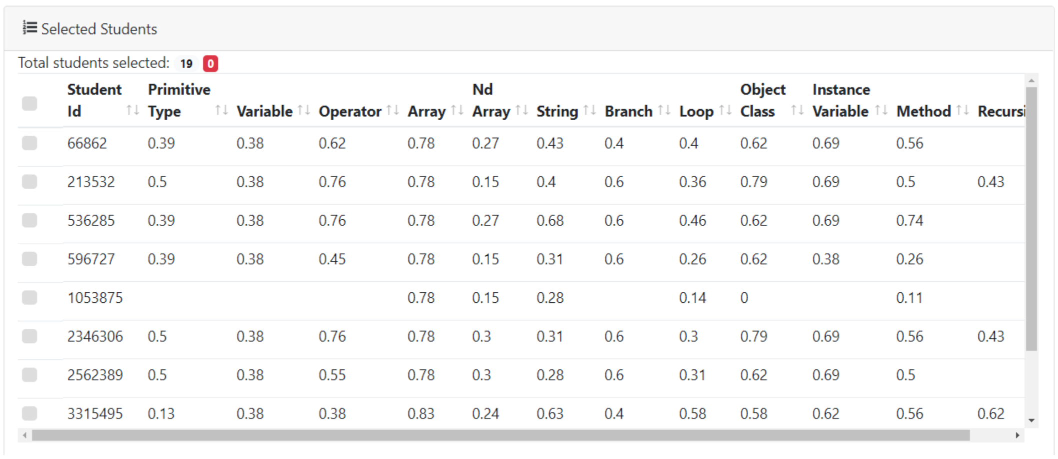


Figure 5: Parallel coordinates of scores


Selected Students

Total students selected: 19 0

Student Id	Primitive Type	Variable	Operator	Array	Nd Array	String	Branch	Loop	Object Class	Instance Variable	Method	Recursi
66862	0.39	0.38	0.62	0.78	0.27	0.43	0.4	0.4	0.62	0.69	0.56	
213532	0.5	0.38	0.76	0.78	0.15	0.4	0.6	0.36	0.79	0.69	0.5	0.43
536285	0.39	0.38	0.76	0.78	0.27	0.68	0.6	0.46	0.62	0.69	0.74	
596727	0.39	0.38	0.45	0.78	0.15	0.31	0.6	0.26	0.62	0.38	0.26	
1053875				0.78	0.15	0.28		0.14	0		0.11	
2346306	0.5	0.38	0.76	0.78	0.3	0.31	0.6	0.3	0.79	0.69	0.56	0.43
2562389	0.5	0.38	0.55	0.78	0.3	0.28	0.6	0.31	0.62	0.69	0.5	
3315495	0.13	0.38	0.38	0.83	0.24	0.63	0.4	0.58	0.58	0.62	0.56	0.62

Figure 6: The synchronized table shows 19 students are selected

[9] C. Liang, J. Ye, S. Wang, B. Pursel, and C. L. Giles. Investigating active learning for concept prerequisite learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[10] C. Liang, J. Ye, Z. Wu, B. Pursel, and C. L. Giles. Recovering concept prerequisite relations from university course dependencies. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[11] S. C. North and E. Koutsofios. Applications of graph visualization. 1999.

[12] D. F. Onah, J. Sinclair, and R. Boyatt. Dropout rates of massive open online courses: behavioural patterns. *EDULEARN14 proceedings*, 1:5825–5834, 2014.

[13] L. Pan, C. Li, J. Li, and J. Tang. Prerequisite relation learning for concepts in moocs. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1447–1456, 2017.

[14] C. Pettitt. Dagre. `https://github.com/dagrejs/dagre`, 2014.

[15] H. Qu. Visual analytics for mooc data. *IEEE Computer Graphics and Applications*, 35:69–75, 2015.

[16] C. Shi, S. Fu, and H. Qu. Vismooc: Visualizing video clickstream data from massive open online courses. *2014 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 277–278, 2014.

[17] C. L. Sistrom and C. W. Garvan. Proportions, odds, and risk. *Radiology*, 230 1:12–9, 2004.