Prerequisite-aware course ordering towards getting relevant job opportunities

AUTHOR(S):

Dai, Yiling; Yoshikawa, Masatoshi; Sugiyama, Kazunari

CITATION:

Dai, Yiling ...[et al]. Prerequisite-aware course ordering towards getting relevant job opportunities. Expert Systems with Applications 2021, 183: 115233.

ISSUE DATE:

2021-11-30

URL:

http://hdl.handle.net/2433/277446

Expert Systems With Applications 183 (2021) 115233

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Prerequisite-aware course ordering towards getting relevant job opportunities

Yiling Dai [*,1], Masatoshi Yoshikawa, Kazunari Sugiyama

Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan

ARTICLE INFO

Keywords:
Course ordering
Prerequisite
Job opportunity
Technical terminology
Markov decision process
Pedagogical perspectives

ABSTRACT

Adapting learning experience according to the rapidly-changing job market is essential for students to achieve fruitful learning and successful career development. As building blocks of potential job opportunities, we focus on "technical terminologies" which are frequently required in the job market. Given a technical terminology, we aim at identifying an order of courses which contributes to the acquisition of knowledge about the terminology and also follows the prerequisite relationships among courses. To solve the course ordering problem, we develop a two-step approach, in which course-terminology relatedness is first estimated and then courses are ordered based on the prerequisite relationships and the estimated relatedness. Focusing on the second step, we propose a method based on Markov decision process (MDPOrd) and compare it with three other methods: (1) a method that orders courses based on aggregated relatedness (AggRelOrd), (2) a method that topologically sorts the courses based on personalized PageRank values (PageRankTS), and (3) a method that greedily picks courses based on the average relatedness (GVPickings). In addition to evaluating how the order prioritizes the related courses, we also evaluate from pedagogical perspectives, namely, how the order prioritizes specifically/generally fundamental courses, and how it places courses close to their prerequisites. Experimental results on two course sets show that MDPOrd outperforms the other methods in prioritizing related courses. In addition, MDPOrd is effective in ordering courses close to their prerequisites, but does not work well in highly ranking fundamental courses in the order.

1. Introduction

Many college students experience the following when their graduations are approaching: flooded with a large amount of job postings, closely looking at their curriculum vitae (CV), and struggling to attract recruiters' attention among a bunch of competitors' applications. With the development of open and online education, we hold a vision that, in the near future, the students will be able to freely construct their own curriculum, which is not prescribed by the institutions. In addition, students are encouraged to accumulate working experience earlier as it is helpful for building the professional identity (Kapoor & Gardner-McCune, 2019). Consequently, it is necessary for students to keep adapting their learning experience to achieve fruitful learning and successful career development according to the frequent updates in the job market.

"Technical terminologies" are important building blocks of job opportunities. For instance, `scripting language` is one of such technical terminologies frequently required in IT job positions. In our work, we address technical terminologies as students' learning goals. Given `scripting language`, what is the best order to take courses for students? It is a difficult question even if the number of candidate courses is small. Table 1 lists an example of eight courses related to `scripting language`. According to the course title and the snippet of the course content, we find that courses c_{12} and c_{17} are helpful for learning some scripting languages such as Cascade Style Sheet, JavaScript, and Python. In addition, course c_{21} also addresses some scripting languages though they are not the main topics of the course. As a result, we recommend courses c_{12}, c_{17}, and c_{21} to students, in which priority is given to c_{12} and c_{17}. However, some of these courses are built on the basis of other courses. For example, before a student learns advanced knowledge about web design and programming in course c_{12}, the student should understand the basic knowledge of web design and development in course c_6. Given a course, some courses that students need to learn prior to it are called the *prerequisites* of it. Therefore, we need to not

* Corresponding author.
E-mail addresses: dai.yiling.4t@kyoto-u.ac.jp (Y. Dai), yoshikawa@i.kyoto-u.ac.jp (M. Yoshikawa), kaz.sugiyama@i.kyoto-u.ac.jp (K. Sugiyama).
[1] Currently at Academic Center for Computing and Media Studies, Kyoto University, Kyoto 606-8501, Japan.

https://doi.org/10.1016/j.eswa.2021.115233
Received 19 January 2021; Received in revised form 2 April 2021; Accepted 16 May 2021
Available online 15 June 2021

Table 1
Example of courses.

ID	Courses
c_1	Computer Programming 1
c_2	Computer Programming 2
c_4	Data Structures and Algorithm Analysis
c_6	Web Site Design and Development
c_{12}	Advanced Web Design and Programming
	(…Students examine advanced topics in Hyper Text Markup Language, **Cascade Style Sheet** and **JavaScript** for …)
c_{13}	Database Systems
c_{17}	Programming Methods
	(…Students learn a combination of visual programming using C# and scripting language using **Python** in this course. …)
c_{21}	Web-based Information Systems
	(…Students use a variety of web development tools and programming/ **scripting** languages. …)

only prioritize the courses related to the target terminology but also pay attention to the prerequisite relationships among courses when providing an order to take them. Actually, the number of candidate courses and the complexity of the course prerequisite relationships are expected to be much larger than the example shown in Table 1, which motivates this work.

Many researchers have been working on recommending learning materials using implicit (Srivastava, Palshikar, Chaurasia, & Dixit, 2018; Polyzou, Nikolakopoulos, & Karypis, 2019; Zhang et al., 2019; Pardos & Jiang, 2020) and explicit (Bridges et al., 2018; Jiang, Pardos, & Wei, 2019; Parameswaran & Garcia-Molina, 2009; Shi, Wang, Xing, & Xu, 2020; Xu, Xing, & Schaar, 2016; Zhao, Yang, Li, & Nie, 2020; Zhu et al., 2018; Ma et al., 2020) approaches. Our work is the first to explicitly combine two important factors behind the course selection behavior, namely, the course prerequisite relationships and the job-oriented learning goal. Other works aimed at recommending job opportunities to students (Almaleh, Aslam, Saeedi, & Aljohani, 2019; Guo, Alamudun, & Hammond, 2016; Qin et al., 2018; Jacobsen and Spanakis, 2019) or recommending learning materials for career demands (Srivastava et al., 2018; Wang et al., 2020). While the former works ignored the course prerequisite relationships and orders (Almaleh, Aslam, Saeedi, & Aljohani, 2019; Guo, Alamudun, & Hammond, 2016; Qin et al., 2018; Jacobsen and Spanakis, 2019), the latter ones focused on short-term learning scenarios in which only one course is recommended (Srivastava et al., 2018; Wang et al., 2020). In contrast, our work is helpful for planning long-term learning in which multiple courses are involved.

We develop a two-step approach to solve the course ordering problem: The first step estimates the relatedness of the courses to the technical terminology (hereafter, course-terminology relatedness). Then the second step determines the order of the courses based on the estimated relatedness and prerequisite relationships. We address the first step as a general task of relatedness estimation and put emphasis on the second step. Specifically, we then propose a method for ordering courses based on Markov decision process (Puterman, 1994) and conduct comparative experiments.

Furthermore, we explore whether the order is relevant from other pedagogical perspectives, such as whether the very basic course ranked first, the course is close to its prerequisites in the order, and so on. We conduct experiments on two different course sets and compare the strengths and weaknesses of the methods comprehensively.

The contributions of our work are summarized as follows:

- To the best of our knowledge, our work is the first that attempts to order the courses towards a job-oriented learning goal by following the prerequisite constraint.
- Our work is information retrieval-driven and education-aware one. In other words, we mainly serve the students whose information gain is maximized by following the order, and also explore whether the generated order is helpful for students or not from educational perspectives.
- We construct a fair-scale dataset annotated with three kinds of labels that denote relatedness between courses and technical terminologies.

The remaining of this paper is organized as follows: In Section 2, we review related works by highlighting the major differences between our work and them. In Section 3, we formulate our task. In Section 4, we detail our proposed and its comparative methods. In Section 5, we present our dataset, experimental results, and discuss them in detail. Finally, we conclude the work with a summary and directions for future work in Section 6.

2. Related work

2.1. Recommending learning materials

Students select learning materials (e.g., courses) based on various factors such as their interest toward the subject, the expectancy of achieving high grades, the alignment with their career plans, the social aspects, and the popularity of the materials (Ma et al., 2020). Therefore, to achieve better recommendation of learning materials, such factors should be taken into account.

Some works adopted implicit approaches where the reasons underlying the recommendations are hidden in the model and the training data. Srivastava et al. (2018) employed sequence mining techniques to identify the next training program a user is likely to take from the historical data. Polyzou et al. (2019) constructed a course transition graph from the course enrollment data and predicted the next course to take based on a random walk model. Zhang et al. (2019) built an attention-based recommendation model from the course enrollment data and proposed a hierarchical reinforcement learning algorithm to revise the course sequence and modify the recommendation model. Pardos and Jiang (2020) adopted skip-gram model to learn the course embeddings from the course enrollment data and recommend the most similar course for a given course. These works assume that the observed course taking behavior is the best answer we can expect. However, the observed course enrollment patterns are the consequences of complex decision making processes. The reasons that lead the students to take courses in those sequences have not been unveiled yet.

Other works included explicit rationales behind the student behaviors into their models. Parameswaran and Garcia-Molina (2009) aimed at finding the set of k items which has the maximum total score and meanwhile meets the prerequisite constraint.[2] Xu et al., 2016 developed a forward-search backward-induction algorithm to optimize course sequences with shorter time needed for graduation. Bridges et al. (2018) constructed a course transition graph from the student enrollment and grade data, in order to recommend the next course which is popular among the students and provides a grade improvement for the given student. Jiang, Pardos, and Wei (2019) used recurrent neural network to predict course grade from the student course enrollment and grade history. Zhao, Yang, Li, and Nie (2020) combined neural attention network and course prerequisite relation embeddings so that the recommended course is similar and understandable to the courses a student has taken. Ma et al. (2020) proposed a hybrid framework to recommend courses from three aspects, namely, the interest-based, timing-based, and grade-based scores. Zhu et al. (2018) and Shi, Wang, Xing, and Xu (2020) proposed a method for recommending learning paths from a knowledge map by meeting multiple constraints such as whether the paths contain unlearnt, important or popular knowledge nodes, and so

[2] Although the order is neglected in their problem, the algorithm can be adapted to solve our problem and the details are explained in Section 4.2.3.

on.

We attempt to identify a course order that meets the prerequisite relationships and the job-oriented goal. Compared with previous works, our work has the following advantages: (a) the model and the result are easy to explain, (b) we focus on the under-explored learning goal that is aligned with job opportunities, and (c) we aim at ordering multiple courses so that the information gain at each position is optimized.

2.2. Connecting academia and industries

To bridge the gap between the academia and the industries, many works attempted to recommend job opportunities to students or recommend learning materials for career demands.

A common approach can be observed in a category of research works in which the most "similar" job to the student's educational background is recommended (Almaleh, Aslam, Saeedi, & Aljohani, 2019; Guo, Alamudun, & Hammond, 2016; Qin et al., 2018; Jacobsen and Spanakis, 2019). Commonly, job postings and student CVs are utilized, and then various techniques, such as ontology-based skill similarity (Guo et al., 2016), recurrent neural network (Qin et al., 2018), latent Dirichlet allocation (Jacobsen & Spanakis, 2019), and naïve Bayes classifier (Almaleh et al., 2019), are adopted to estimate the similarity between them. These works treated the students' learning experience as a whole, namely, all the courses are completed and the students' acquisition level of those courses influences the recommendation performance. On the other hand, our work puts much emphasis on making a plan for learning, in which the course prerequisite relationships and orders are more important.

Another category of research works attempted to identify relevant courses or training programs for job positions or demands of career development (Srivastava et al., 2018; Wang et al., 2020). Srivastava et al. (2018) inferred a course which a student is most likely to take next by mining from a large-scale course enrollment data. Wang et al. (2020) recommended a course based on the employee's current competencies modelled from their skill profiles. For this reason, their works concentrated on the short-term learning needs, namely, the next one course is recommended for the student. Our work not only extracts the most necessary courses but also achieves prerequisite-aware course ordering for long-term learning needs.

2.3. Adapting Markov decision process

Markov decision process (MDP) is a stochastic sequential decision model, which has been widely applied to inventory management, equipment maintenance, communication systems, and so on (Puterman, 1994). The main idea behind MDP is to find the best set of decisions that optimizes the long-term reward.

Some researchers have adapted MDP to solve information retrieval and recommendation problems. Tavakol and Brefeld (2014) formulated the item view session of the user in an e-commerce system as MDP and estimated the parameters from a labeled dataset. They modeled the item as a disjoint set of attributes and recommended the item that best fits the estimated distribution over the attributes. Murray (2015) leveraged MDP to generate a summary from a corpus, where the next word in the sentence is selected based on its importance and the co-occurrence with the previous word. Srivastava et al. (2018) proposed a recommender system that provides a next training program from the historical training sequence data in a similar way of MDP. Xia et al. (2017) employed MDP to diversify a search result, where the perceived utility of the user is modeled as the state and optimized in the algorithm.

Although MDP has been applied to many tasks, our work is the first to adopt it for the course ordering problem. The difficulties are twofold:

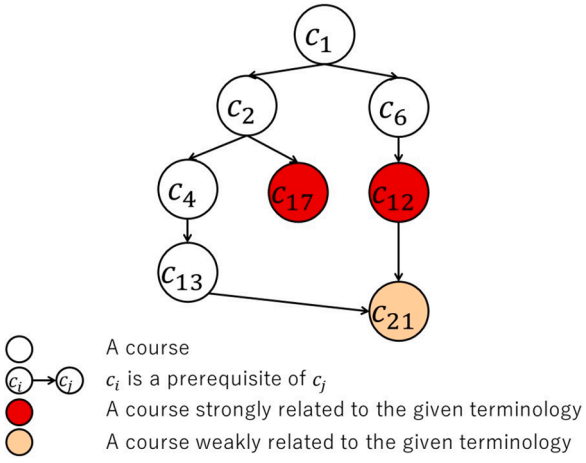


Fig. 1. Prerequisite relationships among courses.

(a) How can we model the problem under an MDP framework given the learning goal and the prerequisite constraint? (b) How can we estimate the parameters without relying on a labeled dataset? Section 4.1 describes the details.

3. Formulating our task

3.1. Awareness of prerequisite relationships

Generally, we need to acquire some knowledge before understanding more advanced knowledge. For instance, if we do not know "algorithms" at all, it is difficult to understand the "complexity of an algorithm". In taking courses, following the prerequisite relationships among courses is essential.

In traditional educational institutions, course prerequisite relationships are defined by the curriculum designers. In open and online education, massive learning materials are available. For the latter learning environment, research works focusing on the extraction of prerequisite relationships among concepts (Liang, Wu, Huang, & Giles, 2015; Gordon, Zhu, Galstyan, Natarajan, & Burns, 2016; Pan, Li, Li, & Tang, 2017; Sayyadiharikandeh, Gordon, Ambite, & Lerman, 2019) are helpful. Works on prerequisite extraction are orthogonal to our work and we assume the prerequisite relationships are given in our problem setting.

3.2. Definition of a "relevant" order

Given a technical terminology, we address how to effectively acquire the knowledge related to it. Fig. 1 shows the prerequisite relationships among the courses in Table 1 and how these courses are related to `scripting language`. Then, the relevant order is generated by highly ranking the courses which are related to the terminology. In other words, $c_1 \rightarrow c_6 \rightarrow c_{12} \rightarrow c_2 \rightarrow c_{17} \rightarrow c_4 \rightarrow c_{13} \rightarrow c_{21}$ (hereafter, Order 1) is a relevant order as all the related courses (i.e., c_{12}, c_{17}, and c_{21}) are ranked in the highest positions where they could appear. The goal of our work is to identify the most relevant order by taking the prerequisite constraint into account.

3.3. Observation from pedagogical perspectives

Depending on the complexity of the course network and the number of related courses, there may exist more than one relevant order of courses for a technical terminology. Even for the simple course relationships in Fig. 1, we can find another order, $c_1 \rightarrow c_2 \rightarrow c_{17} \rightarrow c_6 \rightarrow c_{12} \rightarrow$

$c_4 \rightarrow c_{13} \rightarrow c_{21}$ (hereafter, Order 2), which is equally relevant as Order 1 in terms of prioritizing related courses. However, Orders 1 and 2 give different effects on a student's learning experience. For example, Order 1 places c_{12} prior to c_{17}, which is desirable if the student prefers learning a basic course first and then other advanced courses. In contrast, Order 2, where c_{12} is placed closer to c_{21}, is more desirable if a student dislikes the time lag between a course and its prerequisites. Hence, we explore the relevancy of an order from the following pedagogical perspectives.

- **Specific fundamentality.** Specific fundamentality defines how likely a course will form the basis towards understanding the technical terminology. Prioritizing such courses makes a student's basic knowledge solid, helping the learning of more advanced knowledge.
- **General fundamentality.** In contrast to specific fundamentality above, general fundamentality defines how likely a course will form the basis towards understanding the whole domain. A general fundamentality-focused order is especially important in a learning context with high uncertainty. In other words, if students change their goals during the learning process, they can still benefit from the courses they have already completed since the courses also contribute to the understanding of other terminologies.
- **Local reference.** Local reference refers to placing the prerequisites of a course closer to it. From a cognitive point of view, to shorten the time lag between courses with dependency helps reduce the extra cognitive load in learning (Agrawal, Golshan, & Papalexakis, 2016; Paas, Renkl, & Sweller, 2003).

Theoretically, our concerns are various aspects that affect a student's learning experience. However, it is difficult to satisfy all types of students' learning preferences at the same time as some of them are inherently contradictory to each other. Our work mainly proposes a method for identifying "relevant" orders for understanding the terminology, and we explore whether the orders are relevant from the aforementioned pedagogical perspectives.

3.4. Problem definitions

A course set V and the prerequisite relationships $E = \{(c_i, c_j) | c_i, c_j \in V, \text{course } c_i \text{ is a prerequisite of course } c_j\}$ forms a course graph $G = \langle V, E \rangle$. In our work, when given a technical terminology t and a course graph G, we aim at identifying an order $Ord(V|t) = (c_i, \ldots, c_j)$ subject to the prerequisite relationships and prioritizing the courses closely related to t. In the following sections, we denote $pos(c, Ord)$ and $Ord[i]$ as the position of c in Ord and the course at the i^{th} position in Ord, respectively.

4. Methodology

To acquire the knowledge of a given technical terminology, we need to not only identify the related courses but also pay attention to the prerequisites when deciding the order to take them. Intuitively, our task consists of the following two steps:

(STP1) Estimation of how each course is related to the given technical terminology, namely, course-terminology relatedness, and
(STP2) Ordering the courses based on their course-terminology relatedness and prerequisite relationships.

In STP1, course-terminology relatedness can be estimated by matching the textual information of courses and terminologies. Relatedness estimation is an essential and ever-growing research domain in information retrieval, natural language processing, machine learning, and data mining. A variety of existing approaches can help realize it. In this work, we do not focus on the proposal or comparison of relatedness estimation methods. Instead, we adopt one of the simplest methods—TF-IDF scheme (Salton & Buckley, 1988) to obtain the course-terminology relatedness. Refer to the book (Manning, Raghavan, & Schütze, 2008) to understand relatedness estimation and the survey (Altinel and Ganiz, 2018) to learn recent advances. We then focus on STP2 to propose a method for ordering courses given course-terminology relatedness.

4.1. Proposed method— Markov Decision Process-Based Ordering (MDPOrd)

4.1.1. Intuition of adapting Markov decision process

To acquire the knowledge of a given technical terminology, deciding the order to take the courses involves a sophisticated judgement over multiple factors. Firstly, we need to identify the courses we are qualified to take based on the courses we have already taken. Secondly, we need to consider the instant knowledge acquisition by selecting a new course. Last but not least, we need to forecast the future gain of selecting a new course. In other words, we may choose a course that itself gives slight knowledge gain but is helpful for learning more advanced courses about the terminology. As a result, every step in the course ordering process should be based on the possible options and the expectation of the future gain by taking that step. MDP can model this process well and identify an order of courses that optimizes the information gain from a long-term perspective.

4.1.2. Formulation of MDPOrd

A general model of MDP consists of decision epochs T, states S, actions A, rewards R, transition probabilities P, discount factor γ, and policies Π, namely, denoted as $\{T, S, A, R, P, \gamma, \Pi\}$. In this framework, a decision maker follows a policy to take an action at each epoch, receives a reward and transits between states. The main goal is to find a policy which leads to a state with optimal discounted future gain. We explain how each components are modeled in the following:

Decision Epochs. Let T be the discrete time steps in the system. In our work, we adopt infinite horizon $T \equiv \{1, 2, \ldots\}$ to find the best policy, which assumes the system does not know how long the process will last at any time step. Infinite horizon assures a stationary policy for any state regardless at what time step it reaches the state. The practical meaning of infinite horizon in our work can be explained as follows: the system will suggest the same course order whenever the students start learning. This is reasonable if we assume the students always have sufficient time to learn the courses.

States. Let S be the set of possible states the system occupies at a time step. In our work, a state s is a set of courses a student has completed. As our work needs to follow prerequisite relationships, for any state s, if it includes a course c, it must include the prerequisites of c. Therefore, we denote a possible $s = \{c_j \in V | \exists (c_i, c_j) \in E \Rightarrow c_i \in s\}$. It is time-consuming to exhaust all the possible states, especially when a course is allowed to have more than one prerequisite in the graph. Table 2 demonstrates the process of generating valid states for the course graph in Fig. 1 and the followings detail the process:

1. We add a dummy node to the graph and connect it to the original root nodes (the value of its in-degree is 0) in the graph. The addition of a dummy node assures that
   (a) all the sub graphs are linked as one graph if more than one root exist, and

Table 2
Process of generating valid states for the course graph in Fig. 1.

Step	Node	From	Parents visited	Generated states
1	*dummy*	–	True	∅
2	c_1	*dummy*	True	$\{c_1\}$
3	c_2	c_1	True	$\{c_1, c_2\}$
4	c_4	c_2	True	$\{c_1, c_2, c_4\}$
5	c_{13}	c_4	True	$\{c_1, c_2, c_4, c_{13}\}$
6	c_{21}	c_{13}	False	–
7	c_{17}	c_2	True	$\{c_1, c_2, c_{17}\}$
				$\{c_1, c_2, c_4, c_{17}\}$
				$\{c_1, c_2, c_4, c_{13}, c_{17}\}$
8	c_6	c_1	True	$\{c_1, c_6\}$
				$\{c_1, c_2, c_6\}$
				$\{c_1, c_2, c_4, c_6\}$
				$\{c_1, c_2, c_4, c_{13}, c_6\}$
				$\{c_1, c_2, c_{17}, c_6\}$
				$\{c_1, c_2, c_4, c_{17}, c_6\}$
				$\{c_1, c_2, c_4, c_{13}, c_{17}, c_6\}$
9	c_{12}	c_6	True	$\{c_1, c_6, c_{12}\}$
				$\{c_1, c_2, c_6, c_{12}\}$
				$\{c_1, c_2, c_4, c_6, c_{12}\}$
				$\{c_1, c_2, c_4, c_{13}, c_6, c_{12}\}$
				$\{c_1, c_2, c_{17}, c_6, c_{12}\}$
				$\{c_1, c_2, c_4, c_{17}, c_6, c_{12}\}$
				$\{c_1, c_2, c_4, c_{13}, c_{17}, c_6, c_{12}\}$
10	c_{21}	c_{12}	True	$\{c_1, c_2, c_4, c_{13}, c_6, c_{12}, c_{21}\}$
				$\{c_1, c_2, c_4, c_{13}, c_{17}, c_6, c_{12}, c_{21}\}$

(b) the generated states will always include an empty set, which is the start state to find an order of courses.

2. We traverse the graph in a depth-first way from the dummy node. Whenever we explore a new node, we generate new states by adding this node to the current states based on the following two rules:

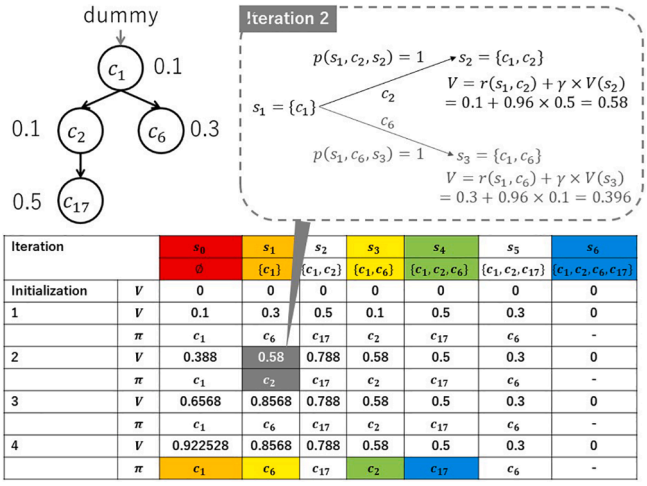
(RL1) Given a node at exploration, new states must be generated from the states generated from its parent and other explored descendants. For example, when we are exploring c_{17}, its parent c_2 and other descendants of c_2 (c_4 and c_{13}) are already explored. We generate new states by adding c_{17} to all the generated states from c_2, c_4, and c_{13} (Steps 3 to 5).

(RL2) Given nodes with multiple parents, we only generate new states at the last time we explore it, and new states must contain all of its parents. For example, when we first explore c_{21} in Step 6 where the other parent c_{12} has not been explored yet, it is skipped. The last time we explore c_{21} when all of its parents have been explored, we can generate new states by adding c_{21} to the states in Step 9 which include both of its parents c_{12} and c_{13}.

Finally, we take the union of all the states generated alongside the traverse as S.

Actions. A denotes the union of actions that the decision maker is allowed to take at each state. In this case, an action is the behavior to take a new course at the current state. Subject to the constraint, the valid action for a state is any course whose prerequisites (if any exists) are already included in the state. For the example in Fig. 2, given $s_1 = \{c_1\}$, the valid actions are c_2 and c_6.

Rewards. $R = S \times A$ denotes the immediate reward obtained from taking a course c at state s. In our work, the reward comes from the information gain towards understanding the technical terminology by taking a course. To this end, we use course-terminology relatedness $rel(c, t)$ to represent the immediate reward. Eq. (1) defines how an immediate reward $r(s, c)$ from taking c at s is computed:



Iteration		s_0 ∅	s_1 $\{c_1\}$	s_2 $\{c_1, c_2\}$	s_3 $\{c_1, c_6\}$	s_4 $\{c_1, c_2, c_6\}$	s_5 $\{c_1, c_2, c_{17}\}$	s_6 $\{c_1, c_2, c_6, c_{17}\}$
Initialization	V	0	0	0	0	0	0	0
1	V	0.1	0.3	0.5	0.1	0.5	0.3	0
	π	c_1	c_6	c_{17}	c_2	c_{17}	c_6	-
2	V	0.388	0.58	0.788	0.58	0.5	0.3	0
	π	c_1	c_2	c_{17}	c_2	c_{17}	c_6	-
3	V	0.6568	0.8568	0.788	0.58	0.5	0.3	0
	π	c_1	c_6	c_{17}	c_2	c_{17}	c_6	-
4	V	0.922528	0.8568	0.788	0.58	0.5	0.3	0
	π	c_1	c_6	c_{17}	c_2	c_{17}	c_6	-

Fig. 2. An example of how to order courses in MDP framework. For simplicity, we only include four nodes in the course graph. The value beside the node indicates the course-terminology relatedness.

$$r\left(s, c\right) = \begin{cases} rel(c, t) & c \notin s, s \cup \{c\} \in S \\ 0 & otherwise \end{cases} \tag{1}$$

In Fig. 2, the reward of taking c_2 from s_1 is 0.1.

Transition Probabilities. $P = S \times A \times S$ denotes the distributions of the probabilities that the system transits from states to states by taking courses. $p(s, c, s')$ denotes the probability that the system transits from s to s' by taking c. Note that $\sum_{s'} p(s,c,s') = 1$. In our work, we simplify the transition probability by considering the system will keep transiting to the next state if some courses are taken. If the action is invalid, the state remains unchanged. Eq. (2) defines this:

$$p\left(s, c, s'\right) = \begin{cases} 1 & s \cup \{c\} = s' \\ 0 & otherwise \end{cases} \tag{2}$$

As shown in Fig. 2, the probability of transiting from s_1 to s_2 by taking c_2 is 1.

Discount Factor. γ denotes a parameter ranged in $0 \leqslant \gamma < 1$ modifying how much of the future gain values currently. This parameter is mainly set for a mathematical reason, allowing the decision maker to find an optimal policy. In our experiment, we set γ to 0.96, which is a commonly used value in MDP.

Policy. $\Pi = S \rightarrow A$ denotes the moving pattern of a decision maker at each state. In MDP model, a policy that maximizes the expected value of all the states is defined as an optimal policy π^*. The expected value of a state $ExpVal(s)$ comes from the immediate reward of taking an action and the expected value of the next state. The best we can expect from a state $ExpVal^*(s)$ is the maximum value of all policies and the optimal policy for s is the action that maximizes $ExpVal(s)$, defined by Eqs. (3) and (4), respectively:

$$ExpVal^*(s) = \max_c (r(s, c) + \gamma \sum_{s'} pr(s, c, s') ExpVal(s')), \tag{3}$$

$$\pi^*(s) = \text{argmax}_c(r(s,c) + \gamma \sum_{s'} pr(s,c,s')ExpVal(s')). \quad (4)$$

Eqs. (3) and (4) can be solved using dynamic programming algorithms which find an approximation of the optimal policy until it converges. The table in Fig. 2 illustrates the value iteration process using the toy example. The dotted frame shows how the state value of s_1 and its best action are calculated in the second iteration. At s_1, c_2 is chosen as the expected value of taking c_2 is greater than the one of taking c_6. In this example, the iteration stops when all the state values are finally converged.

Ordering. In this step, we utilize the output of MDP model to provide relevant course ordering. Once the optimal policy π^* is found, we start with the empty set and follow the best actions to move from one state to another state. If all the courses have a positive reward value, following the optimal policy leads us to a state in which all courses are included. As shown in Fig. 2, $c_1 \rightarrow c_6 \rightarrow c_2 \rightarrow c_{17}$ is identified as the order to take the courses. If not, the state remains unchanged at some point where no more reward can be obtained by taking a new course. In this case, we sort the remaining non-rewarded courses topologically and append them to the order.

4.2. Comparative methods

We compare MDPOrd with the following three different types of methods:

- Aggregated-Relatedness-Based Ordering (AggRelOrd),
- Personalized-PageRank-Based Topological Sorting (PageRankTS), and
- Greedy-Value-Pickings-Based Ordering (GVPickings).

The intuition behind AggRelOrd and PageRankTS is that the priority of a course in the order is determined by its direct relatedness to the terminology and how likely it will be the prerequisite of other related courses. On the other hand, GVPickings selects the courses as a set of paths, where a highly related course having many prerequisites still have a chance to be selected at an earlier stage. We compare these three methods to verify whether our proposed approach works better in ordering the courses. In the following sections, we explain each of the three comparative methods.

4.2.1. Aggregated-Relatedness-Based Ordering (AggRelOrd)

As described above, a course should be prioritized as it is related to the terminology itself and it is the basis of other courses. This method simply treats the maximum relatedness among all the descendants of a course as an aggregated score to indicate its priority. Let $D(c)$ be the set of courses that have a path from c in the course graph, namely, the descendants of c in the graph. Let *aggRel* denote the aggregated relatedness of a course. Then *aggRel* is computed as:

$$aggRel(c|t,V) = (\max_{c' \in c \cup D(c)} rel(c',t)) + |D(c)|\delta, \quad (5)$$

where δ is a very small value added to assure that the score a course gets is always larger than its descendants, thus subject to the prerequisite constraint. At last, the courses are sorted in the order of their *aggRel* scores.

4.2.2. Personalized-PageRank-Based Topological Sorting (PageRankTS)

PageRank (Page, Brin, Motwani, & Winograd, 1999) estimates the probability of a "random walker" who ultimately stops at each node in the network if it follows the links. Therefore, the probability distribution shows the linkage of the network. The more incoming links, the higher probability a node gets. Furthermore, a personalized PageRank (Page et al., 1999; Jeh & Widom, 2003) enables the model to estimate a mixed probability distribution of following both the network linkage and a personalized preference over the nodes. As described at the beginning of Section 4.2, the position of a course in the order should be determined by the relatedness to the terminology and how likely the course will be the prerequisite of other related courses. Here, the direct relatedness between the course and the terminology can be represented in the personalized preference in the PageRank model. In addition, the likelihood of being a prerequisite of other related courses can be captured in the network links part in PageRank model.

We follow the matrix–vector notation in Jeh and Widom's work (Jeh & Widom, 2003) to explain how the PageRank-based score of each course is computed. Let $\vec{v}$ be the PageRank scores over the courses in the graph, and $\vec{u}$ be the course-terminology relatedness. Let $\vec{M}$ be the transition matrix of the graph where $M_{ij} = \frac{1}{|indegree(c_j)|}$ if c_i is prerequisite of c_j and $M_{ij} = 0$ otherwise. $\beta \in [0,1]$ is a teleportation constant and modifies the probability that the "random walker" follows the personalized preference over the nodes.[3] Solving Eq. (6) gives a PageRank-based score $\vec{v}$ for each course.

$$\vec{v} = (1-\beta)\vec{M}\vec{v} + \beta\vec{u} \quad (6)$$

Algorithm 1: PageRank-based topological sorting (PageRankTS)

Input : $G =< V, E >$: the course graph, V: the set of courses, $E = \{(c_i, c_j) \mid \text{if } c_i \text{ is prerequisite of } c_j\}$: the set of course prerequisite relationships, pr_i: the PageRank-based score of c_i
Output: Ord: a queue of courses

```
1 Ord ⟵ ∅
2 while G not empty do
3     Root ⟵ the courses {c} whose indegree is 0
4     Add c to Ord if c ∈ Root and c has the largest pr
5     V ⟵ V \ c
6     E' ⟵ {(c, c') | (c, c') ∈ E}
7     E ⟵ E'
```

Note that the score provided by the PageRank model is not necessarily subject to our prerequisite constraint. In other words, it is possible that a course which has many outgoing edges gets a higher score than its parent course who has no other children. To solve this problem, we rely on topological sorting to reorder the courses based on their PageRank-based scores. As shown in Algorithm 1, line 2–6 is the basic process of topological sorting with a modification in line 4, where the course with the highest PageRank-based score and no any prerequisites is always selected first.

4.2.3. Greedy-Value-Pickings-Based Ordering (GVPickings)

As described in Section 2.1, Parameswaran and Garcia-Molina (2009) proposed Greedy Value Pickings which recommends the best set of k items with the maximum total score and meets the prerequisite constraint. However, their work did not consider the order of items in the set even if the total score is maximized. While Greedy Value Pickings is not designed for ordering the items at the first place, we adapt it to our work. We enhance Greedy Value Pickings by utilizing the order that k

[3] We only report the results obtained by $\beta = 0.2$ in Section 5 as it gives the best performance.

items are added to the final set as the order required in our work, which is shown in Algorithm 2. The basic idea behind this algorithm is to always add the path of courses with the largest average score to the order. As our course graph allows multiple parents for a node, all the paths towards a node should be included if we add a node (lines 4 and 5). Once a path (or paths) of courses is added to the final order, the average scores of the remaining paths are recomputed and the picking is conducted again. The process of computation and picking is repeated until all the courses are correctly ordered.

Algorithm 2. Greedy-Value-Pickings-Based Ordering (GVPickings)

Input : $G =< V, E >$: the course graph, V: the set of courses, $E = \{(c_i, c_j) \mid \text{if } c_i \text{ is prerequisite of } c_j\}$: the set of course prerequisite relationships
Output: Ord: the queue of courses

1 $Ord \longleftarrow \emptyset$
2 **while** G not empty **do**
3 **for** $c \in V$ **do**
4 $Anc(c) \longleftarrow$ ancestors of c
5 $Anc'(c) \longleftarrow Anc(c) \cup \{c\}$
6 $value(c) = \sum_{a \in Anc'(c)} rel(a)/|Anc'(c)|$
7 $\hat{c} \longleftarrow \arg\max_{c \in V} value(c)$
8 Add a topological order of $Anc'(\hat{c})$ to Ord
9 $V \longleftarrow V \setminus Anc'(c)$
10 $E \longleftarrow E \setminus \{(u, v) \mid u \in Anc'(c), (u, v) \in E\}$

5. Experiments

5.1. Dataset

For the convenience of data collection and analysis, we select computer science domain to verify the effectiveness of our proposed methods.

5.1.1. Course graph

We collect course syllabi and prerequisite relationships from the bachelor curricula of computer science in Thompson Rivers University[4] and Rutgers University[5]. We select these curricula as we can access sufficient course information and prerequisite relationships. As a result, we obtain two course graphs— Course Graph 1 with 24 courses and 30 prerequisite relationships, and Course Graph 2 with 20 courses and 25 prerequisite relationships (see Tables A.1, A.2, and Fig. A.1 for further details). In Section 5.4, we report the results for the two course graphs separately.

5.1.2. Technical terminology

We extract a set of technical terminologies, which are frequently required by the job postings in a kaggle dataset.[6] The dataset consists of 19,000 job postings collected from the Armenian human resource portal CareerCenter during 2004 to 2015. We use 3,759 of the them which are labelled as IT-related. We first remove unnecessary sections such as "Company location", "Salary", "Application procedure", and so on, by utilizing some basic text processing techniques. We then extract the technical terminologies from the pre-processed texts by applying entity extraction tool Wikifier, which works well in detecting abstractive concepts (e.g., "software development") and is linked to Wikipedia.

[4] https://www.tru.ca/science/programs/compsci/programs/cs_bachelor_of_computing_science.html, last accessed on January 18, 2021.
[5] https://www.cs.rutgers.edu/academics/undergraduate/course-synopses/articles, last accessed on March 19, 2021.
[6] https://www.kaggle.com/madhab/jobposts/, last accessed on January 18, 2021.

An order of items

Position	Item	Gain	Discounted gain
1	a	3	3.00
2	b	2	1.26
3	c	1	0.50
4	d	2	0.86
5	e	5	1.93
6	f	0	0.00
7	g	1	0.33
Cumulative discounted gain			7.89

Ideal order of items

Position	Item	Gain	Discounted gain
1	e	5	5.00
2	a	3	1.89
3	b	3	1.00
4	d	2	0.86
5	c	2	0.39
6	g	1	0.36
7	f	0	0.00
Cumulative discounted gain			9.50

nDCG=7.89/9.50=0.83

Fig. 3. An example of how to compute nDCG.

More specifically, we employ TAGME (Ferragina & Scaiella, 2010) and DBpedia Spotlight (Daiber, Jakob, Hokamp, & Mendes, 2013) and include the terminologies identified by any of them in the terminology set. While we address the job postings in IT-related domain only, the extracted terminologies still include general ones such as "knowledge", "communication", and so on. As each of the terminologies has its associated Wikipedia page, we filter out irrelevant terminologies that is greater than six hops distant from the Wikipedia category "Computing". Finally, we collect 3,803 terminologies and select the 100 most frequent terminologies in the job postings among them.

5.1.3. Ground truth

It is beyond the cognitive capacity even for an expert to decide the order of taking the courses with complex prerequisite relationships. In addition, it is difficult to integrate several orders into one. To avoid this, we first ask several experts what courses are necessary to take for a given technical terminology without directly collecting the order of courses from them. We then evaluate the relatedness of an order with some traditional information retrieval metrics. We discuss the details in Section 5.3.

We invite five domain experts to construct our ground truth. Given a technical terminology, we first provide them with the graph and the syllabi of the courses, and then ask them to annotate which courses are necessary or preferable to take for better understanding the terminology. Note that "necessary" is a higher level of relatedness than "preferable". In this annotation task, each pair of a course and a terminology can be regarded as an item, and the domain experts give either of the following three labels: "necessary", "preferable", or "unnecessary" to the pair. Some technical terminologies such as `computer programming` and `productivity software` are too general or unrelated to the computer science domain. It is difficult for the domain experts to annotate a proper label for any of the courses for them. We call this type of terminology "irrelevant" and exclude the terminologies viewed as irrelevant by at least one domain expert. Finally, we obtain 67 terminologies in the dataset (see Table B.3 for further details).

We evaluate the agreement among five domain experts with Fleiss kappa coefficient (Fleiss, 1971), resulting in moderate agreement of 0.405 and 0.308 for Course Graph 1 and Course Graph 2, respectively. In a majority-vote method, the five domain experts and the three labels sometimes make it difficult to determine the final label. Therefore, we adopt DS algorithm (Dawid & Skene, 1979), which is one of stochastic approaches, to estimate the probabilities of which label is likely to be given to the pair of course and terminology. Then, we choose the label with the highest probability as the ground truth.

5.2. Estimation of course-terminology relatedness

As described at the beginning of Section 4, our work does not mainly focus on estimating course-terminology relatedness. Using different

techniques in relatedness estimation and then comparing their impact on the course ordering performance is beyond the scope of this work. Instead, we use the course-terminology relatedness annotated by the domain experts in STP1 to explore the best performance that STP2 can achieve. We discuss the details in Section 5.5.1. Here, we adopt a simple yet effective method, TF-IDF scheme (Salton & Buckley, 1988) to estimate course-terminology relatedness. More specifically, we use course syllabus and the leading section in Wikipedia page for the term as the course and terminology corpus. We first compute TF-IDF score from both of the corpus and then take the cosine similarity of the course and terminology vectors as the relatedness score.

5.3. Evaluation metrics

We adopt normalized discounted cumulative gain (nDCG) (Järvelin & Kekäläinen, 2002) to measure the relatedness, specific fundamentality, and general fundamentality of an order of courses. In addition, we propose a distance metric to measure the degree of local reference of an order of courses.

5.3.1. General model of nDCG

nDCG is a widely used metric in information retrieval, which measures how an order of items prioritizes the ones with a higher information gain. Fig. 3 illustrates a simple example of how to compute nDCG. Firstly, the information gain of each item is discounted according to its position. The lower in the order, the more information gain is discounted. Secondly, the cumulative discounted gain is computed by summing up the discounted gain of all the items. At the same time, we reorder the items based on their information gain to obtain an "ideal" order of these items. Lastly, we normalize the cumulative discounted gain of the original order using the one of the ideal order.

nDCG meets our need to measure how an order prioritizes items that provide information of interest such as relatedness, specific fundamentality, and general fundamentality. Let $nDCG@k(Ord)$ be the normalized discounted cumulative gain of an order Ord for items at position k. $nDCG@k(Ord)$ is computed as follows:

$$nDCG@k(Ord) = \frac{DCG@k(Ord)}{DCG@k(Ord_{ideal})}, \tag{7}$$

$$DCG@k(Ord) = \sum_{i=1}^{k} \frac{gain(Ord[i])}{\log_2(i+1)}, \tag{8}$$

where $DCG@k(Ord)$ is the discounted cumulative gain of Ord at position k, Ord_{ideal} is the ideal order ranked according to the information gain of the items, and $gain(Ord[i])$ is the information gain of the i^{th} item in Ord. By replacing $gain(\cdot)$ with the information to be explored, we can adapt nDCG to measure the relatedness, specific fundamentality or general fundamentality of an order.

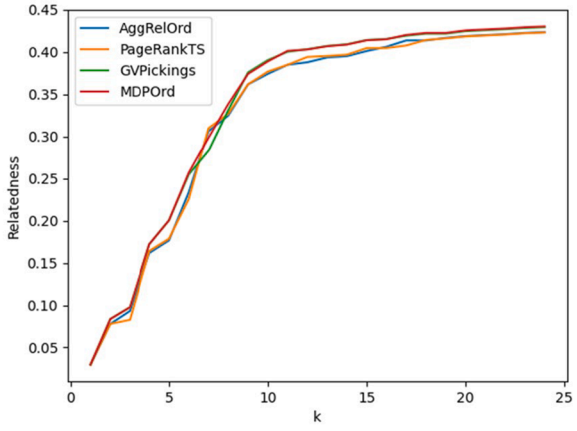
5.3.2. Relatedness of an order

We denote $Relatedness@k$ to measure how an order prioritizes related courses to the terminology. Let $rel^{gt}(c,t)$ be the score defined by the ground truth of course-terminology relatedness, which is transformed from the domain experts' annotated labels as follows:

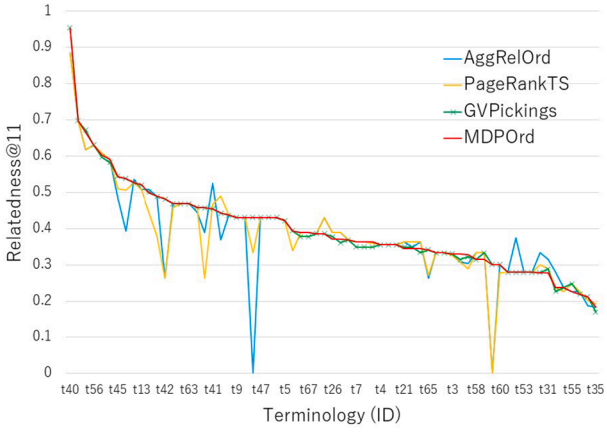
$$rel^{gt}(c,t) = \begin{cases} \alpha & \text{if the label for } c \text{ is "necessary"} \\ 1-\alpha & \text{if the label for } c \text{ is "preferable"} \\ 0 & \text{if the label for } c \text{ is "unnecessary"}, \end{cases} \tag{9}$$

where α ($0 \leqslant \alpha \leqslant 1$) is a parameter to tune the level of relatedness.[7] Then $Relatedness@k$ is computed by replacing $gain(\cdot)$ in Eqs. (7) and (8) with $rel^{gt}(c,t)$.

[7] We set $\alpha = 0.7$ in this work.



(a) Average $Relatedness@k$, varying position k.



(b) $Relatedness@11$ over the terminologies. The terminologies are sorted in descending order based on MDPOrd scores.

Fig. 4. Relatedness obtained by MDPOrd, AggRelOrd, PageRankTS, and GVPickings for Course Graph 1.

5.3.3. Specific fundamentality of an order

We denote $SpecFdm@k$ to measure how an order prioritizes specifically fundamental courses to the terminology. Let $SpecFdm(c|t)$ be the specific fundamentality of a course for a terminology in the graph. We then define $SpecFdm(c|t)$ as follows:

$$SpecFdm(c|t) = \frac{\sum_{c' \in D(c)} rel^{gt}(c',t)}{\sum_{c' \in V \setminus c} rel^{gt}(c',t)}, \tag{10}$$

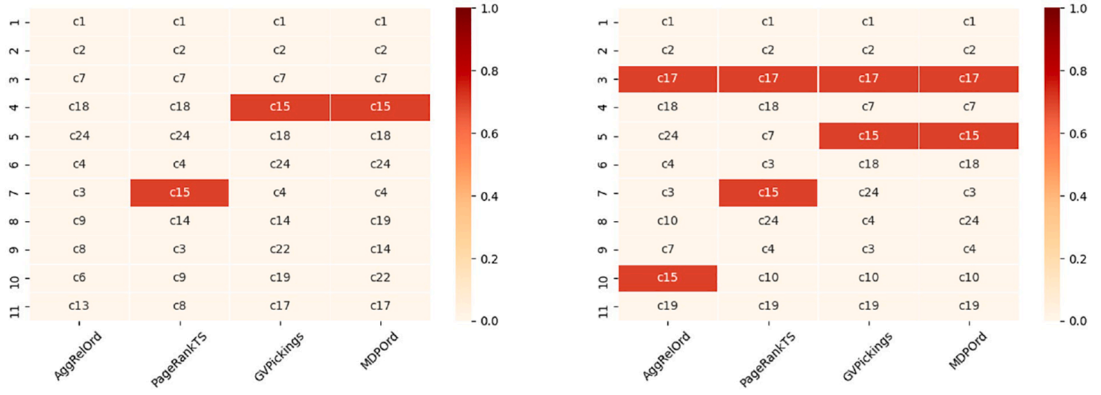
where $D(c)$ is the set of descendants of c in the graph. $SpecFdm(c|t)$ reflects the proportion of the related knowledge based on c to the total amount of related knowledge for the terminology. Then $SpecFdm@k$ is computed by replacing $gain(\cdot)$ in Eqs. (7) and (8) with $SpecFdm(c|t)$.

5.3.4. General fundamentality of an order

We denote $GenFdm@k$ to measure how an order prioritizes generally fundamental courses in the domain. Let $GenFdm(c)$ be the general fundamentality of a course in the graph. Then it is computed as:

$$GenFdm(c) = \frac{|D(c)|}{|V|-1}. \tag{11}$$

$GenFdm(c)$ reflects the proportion of the knowledge based on c to the



(a) `formal specification`. *Relatedness*@11 obtained by AggRelOrd, PageRankTS, GVPickings, and MDPOrd for this terminology is 0, 0.333, 0.431, 0.431, respectively.

(b) `debugging`. *Relatedness*@11 obtained by AggRelOrd, PageRankTS, GVPickings, and MDPOrd for this terminology is 0.483, 0.511, 0.543, 0.543, respectively.

Fig. 5. Examples of orders generated by our proposed methods for Course Graph 1. In each sub-figure, the orders for one terminology are presented. The row and cell color denote the position in the order and the course-term relatedness annotated by the domain experts, respectively. Only the top 11 courses in the orders are demonstrated to compactly show the results.

total knowledge in the domain. Then *GenFdm*@k is computed by replacing $gain(\cdot)$ in Eqs. (7) and (8) with $GenFdm(c)$.

5.3.5. Local reference of an order

We propose "Average Reference Distance (ARD)" to inversely measure how likely an order will place courses close to their prerequisites. We denote *ARD*@k as the average distance between the courses and their prerequisites of an order at position k. Let $RelDis(c|Ord)$ be the reference distance of c in *Ord*. We then define $RelDis(c|Ord)$ as follows:

$$RelDis(c|Ord) = \begin{cases} 0 & Pre(c) = \varnothing \\ \dfrac{\sum_{c' \in Pre(c)} pos(c, Ord) - pos(c', Ord)}{|Pre(c)|} & \text{otherwise} \end{cases}, \quad (12)$$

where $Pre(c)$ is the set of direct prerequisites of c in $G = \langle V, E \rangle$, i.e., $Pre(c) = \{c' | (c', c) \in E\}$. Then *ARD*@$k$ is computed as:

$$ARD@k = \frac{\sum_{i=1}^{k} RelDis(Ord[i])}{k - \sum_{i=1}^{k} 1\{Pre(Ord[i]) = \varnothing\}}. \quad (13)$$

Here, $1\{\cdot\}$ is an indicator function which equals 1 if $\{\cdot\}$ is true, 0 otherwise.

In summary, *Relatedness*@k, *SpecFdm*@k, and *GenFdm*@k range in $[0, 1]$. In addition, the larger the scores, the better the performance. On the other hand, *ARD*@k ranges from 0 to some positive value based on the size and structure of a graph. The smaller *ARD*@k is, the higher the degree of local reference, indicating a better performance.

5.4. Experimental results

5.4.1. Course Graph 1

Relatedness. Fig. 4a shows the average nDCG-based relatedness of the orders of Course Graph 1 obtained by the four methods (MDPOrd, AggRelOrd, PageRankTS, and GVPickings) described in Section 4 by varying k.

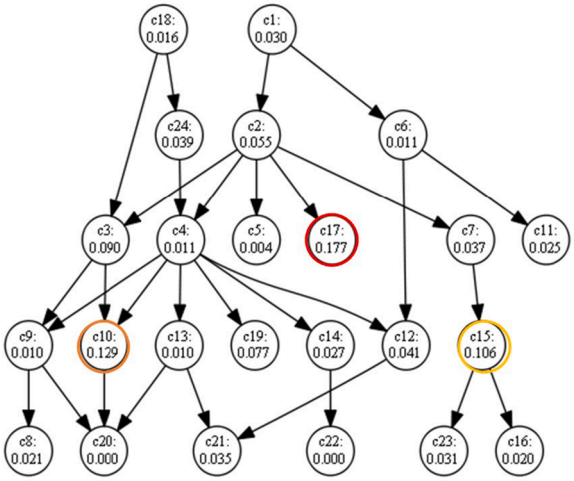
Overall, the *Relatedness* value ranges from 0 to 0.45 as the value of k increases. We observe that GVPickings and MPDOrd slightly outperform AggRelOrd and PageRankTS, especially when k is larger than 9. To further observe the performance over different terminologies, Fig. 4b shows *Relatedness*@11 scores for each terminologies. According to Fig. 4b GVPickings and MDPOrd show a similar relatedness score distribution over the terminologies while AggRelOrd and PageRankTS show some significant performance drop for some of the terminologies.

What do these scores indicate in the orders? We pick two technical terminologies to demonstrate the relation between the *Relatedness* difference and the order difference. Fig. 5 shows the orders generated by the four methods. In `formal specification`, c_{15} (Software Engineering) is considered necessary to take and it is not ordered within the top 11 courses by AggRelOrd, which leads to a *Relatedness* score of 0. Meanwhile, three lower positions of c_{15} in PageRankTS results in around 0.1 drop in the *Relatedness* compared with GVPickings and MDPOrd. In `debugging`, the domain experts annotate both c_{15} (Software Engineering) and c_{17} (Programming Methods) as necessary. All the methods successfully rank c_{17} in its optimal position while AggRelOrd and PageRankTS place c_{15} five and two lower positions than GVPickings and MDPOrd, respectively. The two lower positions for c_{15} result in a decrease of 0.03 in the *Relatedness* of the order. The drop of 0.03 is small in the evaluation metric. However, the order forces the students to learn two more courses before they can reach the one which is helpful for the acquisition of knowledge on the terminology.

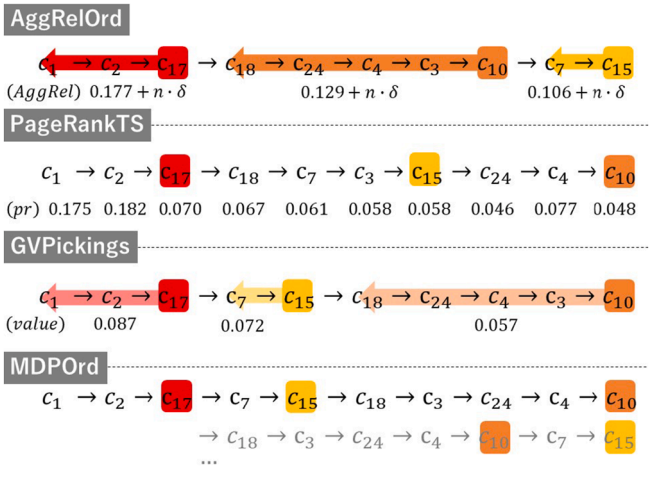
Comparison between the two terminologies indicates that the positioning of the first related course substantially influences the final *Relatedness* score. While following related courses in the order do not give a large impact on the *Relatedness* score, the wrong positioning of them results in extra learning cost in actual cases. In summary, we believe that GVPickings and MDPOrd work well.

To further discuss why GVPickings and MDPOrd work better than AggRelOrd and PageRankTS, we analyze the frameworks of four methods for the terminology `debugging`. Fig. 6a shows the estimated course-terminology relatedness scores, in which c_{17} (Programming Methods), c_{10} (Operating Systems), and c_{15} (Software Engineering) give the top three scores 0.177, 0.129, and 0.106, respectively. However, only c_{17} and c_{15} are annotated as necessary by the domain experts. Thus, the relatedness of c_{10} is overestimated in STP1. Fig. 6b illustrates how the order is determined in each method and how c_{10} affects the ordering process.

AggRelOrd aggregates the score of a course based on its most related descendant. As shown in Fig. 6b, c_1 and c_2 obtain their scores

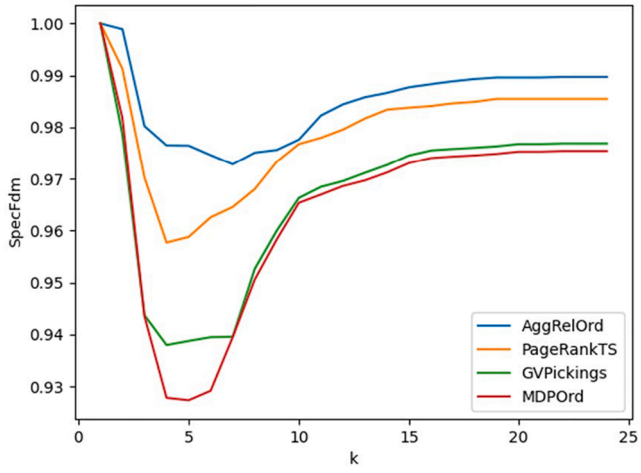


(a) Course Graph 1 with the estimated course-terminology relatedness scores.

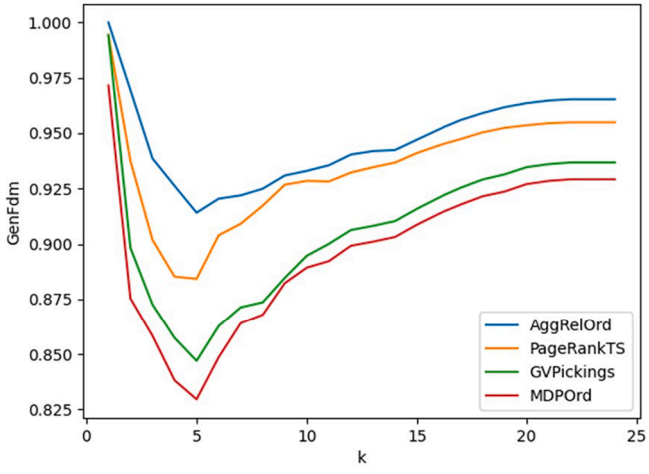


(b) Frameworks of the four methods determining the orders.

Fig. 6. Analysis on the frameworks of AggRelOrd, PageRankTS, GVPickings, and MDPOrd for the terminology `debugging` and Course Graph 1.



(a) *SpecFdm*.



(b) *GenFdm*.

Fig. 7. *SpecFdm* and *GenFdm* obtained by AggRelOrd, PageRankTS, GVPickings, and MDPOrd, varying position k for Course Graph 1.

(0.177+n·δ) from the relatedness of c_{17} with their corresponding small value δ. As a result, the order is determined by ranking the most related courses: c_{17}, c_{10}, and c_{15} with their ancestors ({c_1 and c_2}, {c_{18}, c_{24}, c_4, and c_3}, and {c_7}, respectively). PageRankTS computes the score of each course from the scores of their children recursively. As shown in Fig. 6b, the relatedness of c_{17}, c_{10}, and c_{15} is transferred to other courses. Note that the PageRank scores of c_{15} and c_{10} are partially propagated to their parents and also partially propagated from their children. In this case, c_{10} only has a non-related child c_{20}, resulting in a lower score (0.048) than that of c_{15} (0.058). GVPickings picks paths based on their average relatedness. As shown in Fig. 6b, the path from c_7 to c_{15} gets a higher average value (0.072) than those from c_{18} to c_{10} (0.057), resulting in lower rank for c_{10} while it has a higher relatedness than c_{15}. The decision maker in MDPOrd compares the expected values among multiple candidate orders. For example, placing c_{10} prior to c_{15} results in an increase of expected value from c_{10} but a larger decrease of that from c_{15}. Therefore, the upper order is finally selected.

In summary, AggRelOrd works better in prioritizing a highly-related course no matter how deep it locates in the graph. Consequently, it cannot globally optimize relatedness of the order and wrongly estimated course often generates wrong order. PageRankTS does keep a balance

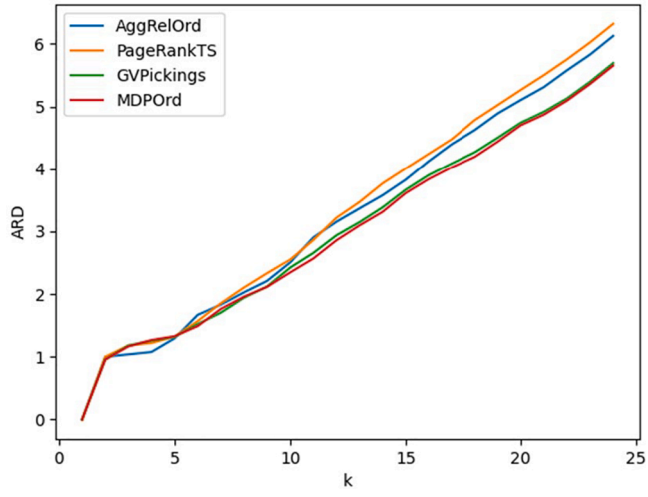
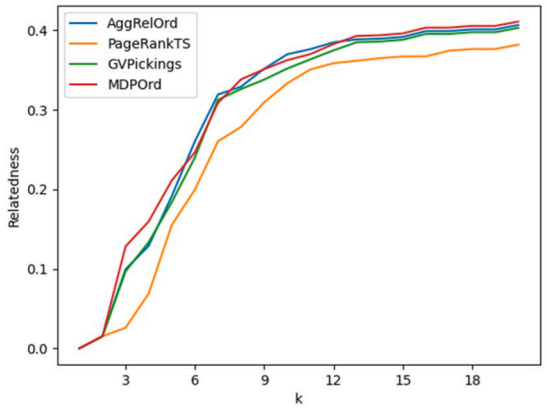
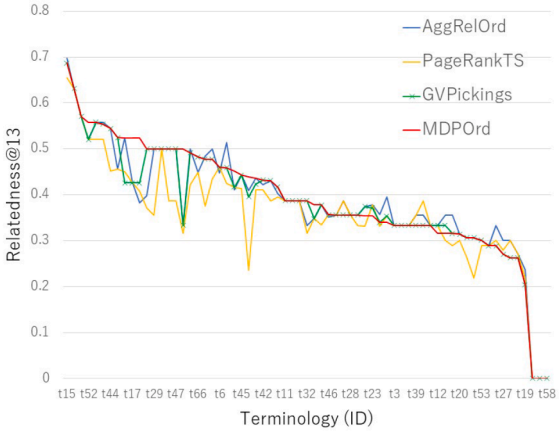


Fig. 8. *ARD* obtained by AggRelOrd, PageRankTS, GVPickings, and MDPOrd, varying position k for Course Graph 1.

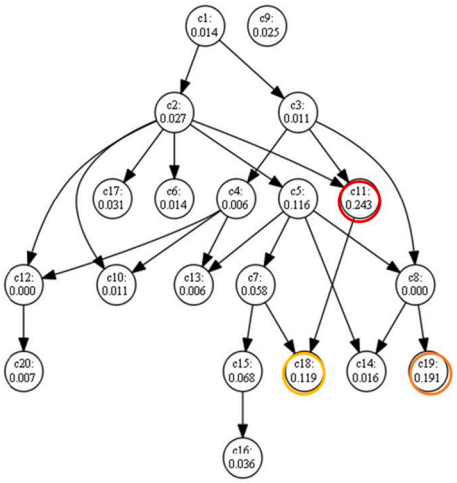


(a) Average *Relatedness*@k, varying position k.

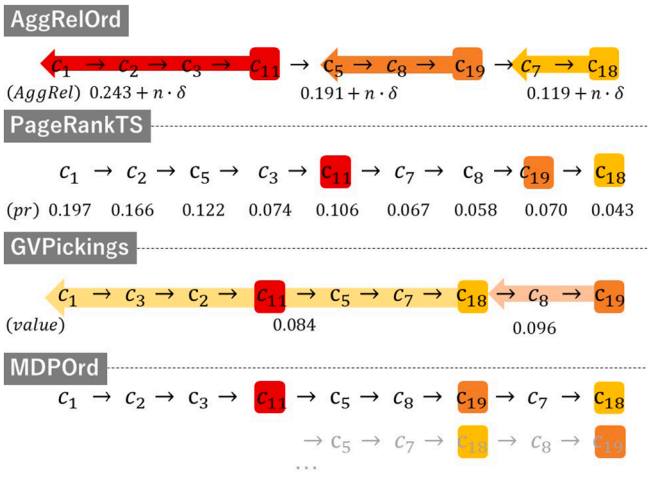


(b) *Relatedness*@13 over the terminologies. The terminologies are sorted in descending order based on MDPOrd scores.

Fig. 9. Relatedness obtained by MDPOrd, AggRelOrd, PageRankTS, and GVPickings for Course Graph 2.



(a) Course Graph 2 with the estimated course-terminology relatedness scores.



(b) Frameworks of the four methods determining the orders.

Fig. 10. Analysis on the frameworks of AggRelOrd, PageRankTS, GVPickings, and MDPOrd for the terminology `knowledge base` and Course Graph 2.

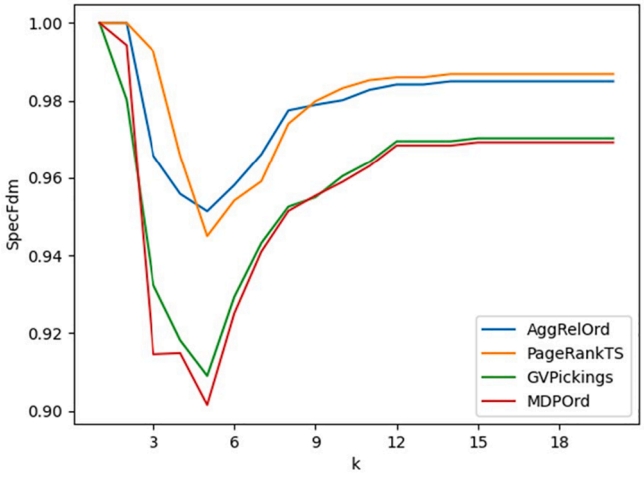
between the relatedness of a course and the relatedness of its descendants. However, the courses with more children tend to be overestimated and the courses without any child tend to be underestimated. In contrast, GVPickings and MDPOrd consider the impact of selecting a course from a long-term perspective. Therefore, GVPickings and MDPOrd are more robust for wrongly estimated course-terminology relatedness and demonstrate more stable performances than AggRelOrd and PageRankTS.

Pedagogical metrics. Figs. 7 and 8 show the *SpecFdm*, *GenFdm*, and *ARD* scores for the orders of Course Graph 1 generated by the four methods, AggRelOrd, PageRankTS, GVPickings, and MDPOrd. We observe similar trends for *SpecFdm* and *GenFdm*. Note that *SpecFdm* and *GenFdm* scores for the orders obtained by the four methods are relatively close to 1, which is due to the prerequisite relationships. In other words, a course that is prerequisite for more courses is usually ordered in a higher position, resulting in better fundamentality score. Among the four methods, AggRelOrd and PageRankTS outperform GVPickings and MDPOrd in both of the fundamentality scores, which can be inferred from the frameworks of AggRelOrd and PageRankTS. AggRelOrd chooses the maximum value among the course and its descendants as the aggregated value of this course. As a result, a course with more descendants, which is more fundamental one, is ordered higher in most cases. In PageRankTS, 80% of the priority of a course comes from the outgoing edges, which shows the fundamentality of the course.

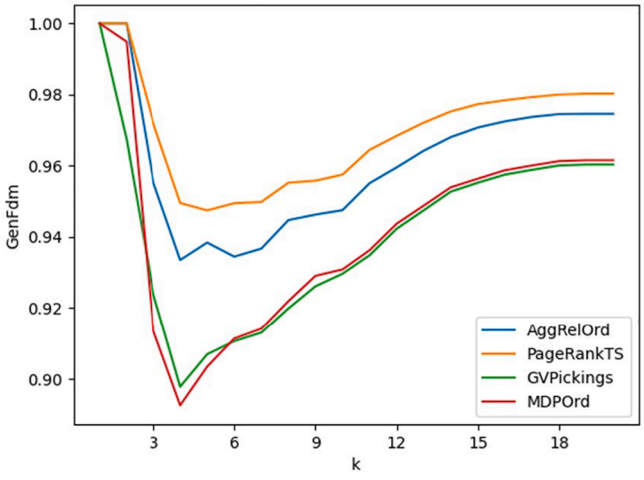
On the other hand, in *ARD*, we observe different trends as shown in Fig. 8. The difference between (AggRelOrd, PageRankTS) and (GVPickings, MDPOrd) is getting larger as the value of k increases. GVPickings explores the courses in the unit of paths, which makes it easier to sequentially order a path of courses. In MDPOrd, if the first course of a path is not so closely related, the courses on the path tend to be ordered lower. In other words, the decision maker in MDPOrd tends to choose courses in the current path to starting a new path. In contrast, AggRelOrd and PageRankTS tend to prioritize courses with more descendants in the graph, resulting in more jumps in the order from one course to another without any prerequisite relationships.

5.4.2. Course Graph 2

Relatedness. Fig. 9a shows the average nDCG-based relatedness of the orders of Course Graph 2 obtained by the four methods (MDPOrd, AggRelOrd, PageRankTS, and GVPickings) described in Section 4 by varying k. Overall, the *Relatedness* of AggRelOrd, GVPickings, and MDPOrd is greater than the one of PageRankTS. When $k \leqslant 5$ and $k \geqslant 13$, MDPOrd outperforms the other methods. Although slight improvement is obtained at the average *Relatedness*@13 of MDPOrd over AggRelOrd and GVPickings, the specific *Relatedness*@13 of different terminologies



(a) *SpecFdm*.



(b) *GenFdm*.

Fig. 11. *SpecFdm* and *GenFdm* obtained by AggRelOrd, PageRankTS, GVPickings, and MDPOrd, varying position k for Course Graph 2.

varies. As shown in Fig. 9b, MDPOrd gives significantly better results over the other methods on some of the terminologies (e.g., t_{17}, t_{29}, t_{47}, etc.)

We take the terminology `knowledge base` as an example to further discuss how the methods perform on Course Graph 2. For `knowledge base`, c_{19} is annotated as necessary and $\{c_{11}, c_{18}\}$ are annotated as preferable courses. As shown in Fig. 10a, c_{11}, c_{19}, and c_{18} obtain the highest relatedness scores 0.243,0.191, and 0.119 in STP1, respectively. Given the course graph and the estimated relatedness, we observe that AggRelOrd, GVPickings, and MDPOrd rank c_{11} at a higher position while PageRankTS places it one position lower. In PageRankTS framework, c_5 has more descendants than c_{11} so that it gets a higher score and takes over the positions of c_3 and c_{11}. AggRelOrd, GVPickings, and MDPOrd generate different orders from c_{11}. GVPickings places c_{18} higher than a more related course c_{19}. As shown in Fig. 10b, the paths from c_1 to c_{18} are selected at once since they have a larger average relatedness. It is not desirable for GVPickings to take such a large step of seven courses as it may ignore other potential courses. AggRelOrd ranks $\{c_5, c_8, c_{19}\}$ and $\{c_7, c_{18}\}$ according to the relatedness of c_{19} and c_{18}. AggRelOrd performs well when the highly-related courses are located deep in the graph or

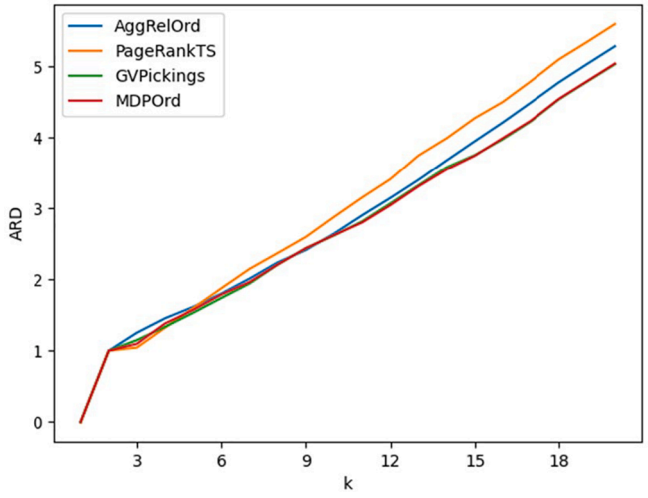


Fig. 12. *ARD* obtained by AggRelOrd, PageRankTS, GVPickings, and MDPOrd, varying position k for Course Graph 2.

when their prerequisites have low relatedness. MDPOrd chooses to place c_{19} higher than c_{18} based on a comparison of other possible orders. In summary, AggRelOrd and MDPOrd succeed to rank related courses at higher positions even if they have many prerequisites.

Pedagogical metrics. Figs. 11 and 12 show the *SpecFdm*, *GenFdm*, and *ARD* scores for the orders of Course Graph 2 generated by the four methods, AggRelOrd, PageRankTS, GVPickings, and MDPOrd. We observe similar results with Course Graph 1— AggRelOrd and PageRankTS outperform GVPickings and MDPOrd in both of the fundamentality scores while GVPickings and MDPOrd work well in ordering courses at a shorter reference distance.[8]

5.5. Discussion

5.5.1. Analysis on performance obtained by STP1 and STP2

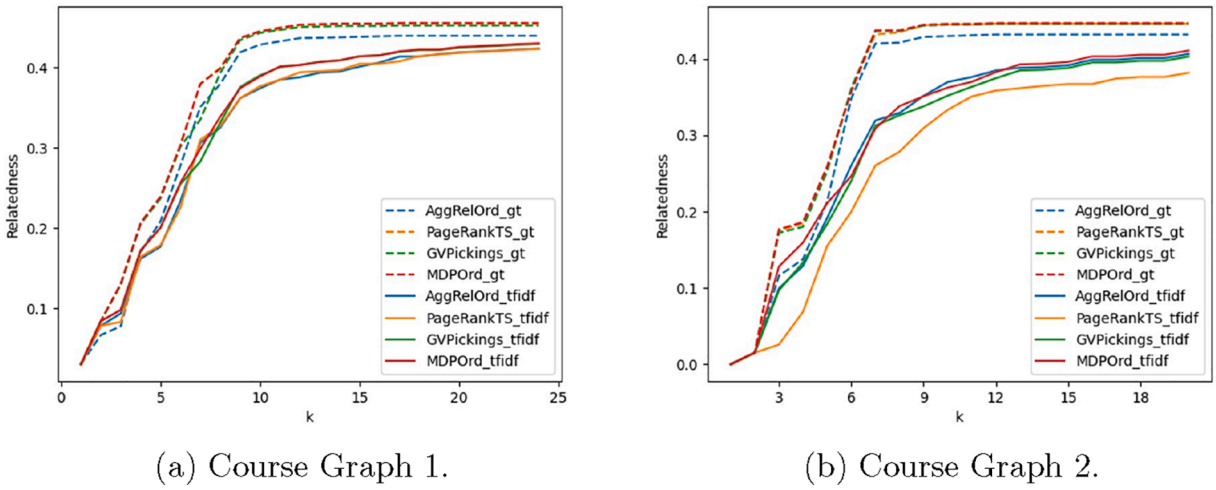
In this work, we put much emphasis on STP2 to identify an order of related courses for the technical terminology and adopt TF-IDF scheme to estimate course-terminology relatedness. As described in Section 5.2, we use the course-terminology relatedness annotated by the domain experts as the input of STP2 to explore the upper bounds of the performance of STP2.

Fig. 13 shows that using the ground truth in STP1 achieves better *Relatedness* than using TF-IDF scheme-based course-terminology relatedness. As described in Section Relatedness, the relatedness of `debugging` and c_{10} (Operating Systems) is overestimated by TF-IDF scheme. This is because the terms such as "system" and "memory" show relatively higher frequency scores in c_{10}, and `debugging` includes these terms. However, if these terms are addressed with more textual information such as "operating system", "main memory", "memory cache", and so on in c_{10}, the relatedness score for `debugging` would be reasonable one. Therefore, improving relatedness estimation is one of our future works.

5.5.2. Comparison of performance on different course graphs

In this work, we conduct experiments on two different course graphs. As described in Section 5.4, we observe that MDPOrd works well in prioritizing related courses for both course graphs. The performance of the other methods such as GVPickings and AggRelOrd varies with the structures of course graph and the locations of highly-related courses. From pedagogical perspectives, AggRelOrd and PageRankTS are effective in prioritizing specifically and generally fundamental courses while

[8] We omit detailed analysis as it is discussed in Section 5.4.1.



(a) Course Graph 1. (b) Course Graph 2.

Fig. 13. *Relatedness* obtained by AggRelOrd, PageRankTS, GVPickings, and MDPOrd by using ground truth and TF-IDF based relatedness in STP1, varying position *k*.

GVPickings and MDPOrd work well in ordering courses at a shorter reference distance.

Compared with MDPOrd, AggRelOrd and GVPickings demonstrate competitive performances in prioritizing related courses. We use a simple example to compare how the course order is determined in AggRelOrd, GVPickings, and MDPOrd frameworks. As shown in the top left of Fig. 14, suppose that we have a course graph of five courses with their course-terminology relatedness.

AggRelOrd prioritizes the path toward the most related course *e*. As a result, the secondly related course *c* is set aside and placed at the last position. AggRelOrd has a strength of spotting a highly-related course especially when it has many prerequisites. However, AggRelOrd is short-sighted and permits no other plans once the path toward the high-related course is detected.

In contrast, GVPickings examines more possible paths as shown in the top right of Fig. 14. Specifically, it selects $a \rightarrow b \rightarrow d \rightarrow e$ over $a \rightarrow b \rightarrow c$ in the first round of picking as it has the largest average relatedness (0.175). It is reasonable that GVPickings takes the cost of prerequisite courses into consideration. However, there remains a concern that whether the duplicated part of different paths should be counted as the

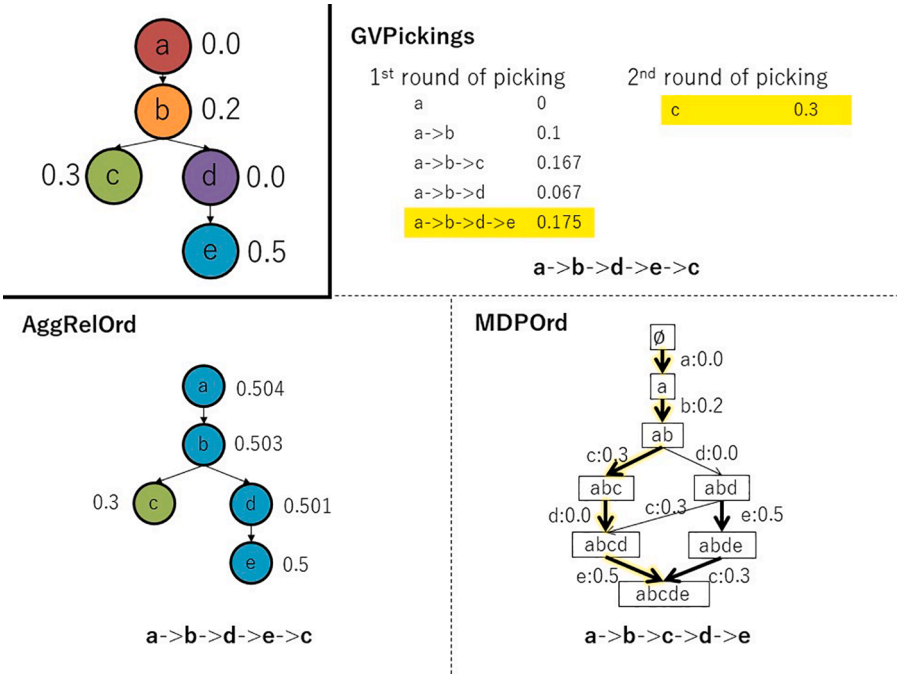


Fig. 14. Comparison among AggRelOrd, GVPickings, and MDPOrd frameworks using a simple example.

cost. For example, if $a \rightarrow b$ is included into the order, it is better to take c next to get a relatedness gain of 0.3 rather than to take $d \rightarrow e$ to get an average relatedness gain of 0.25. Therefore, when GVPickings picks a long path of courses, it needs a more sophisticated way to compute the average relatedness.

On the other hand, MDPOrd works from a future-oriented perspective. The bottom right of Fig. 14 shows the state-action transition pattern in the MDP model, where a rectangle represents a state, an arrow represents an action, and a bold arrow indicates the best policy for a state. For each state, the best policy is chosen based on the reward and the expected value of the future states. At state $\{a, b\}$, taking c, d, and e ensures a higher expected value than any other plans. In other words, MDPOrd chooses to take c first when given the case where it always needs to take a non-reward course d before being able to take the most related course e.

In summary, AggRelOrd prioritizes the most related course without considering the cost and other possibilities. GVPickings examines the cost of taking a related course but sometimes overestimates the influence of prerequisite courses. MDPOrd orders courses from a future-based perspective and keeps a balance among multiple related courses. Actually, the strengths of AggRelOrd and GVPickings are the weaknesses of MDPOrd: (a) MDPOrd does not necessarily rank the highly-related courses if they have many unrelated prerequisites; (b) MDPOrd makes decisions independent of the past, which means the cost of taking previous courses is ignored.

6. Conclusion and future work

In this work, we have addressed the problem of ordering the related courses for a given technical terminology while following the prerequisite relationships among courses. We proposed a two-step approach where the course-terminology relatedness is first estimated and then courses are ordered based on the prerequisite relationships and the estimated relatedness. Putting much emphasis on the second step, we proposed a Markov Decision Process-Based Ordering (MDPOrd) method and compared it with the other three methods— AggRelOrd, PageRankTS, and GVPickings. Except for evaluating how an order prioritizes related courses for a terminology, we also evaluated from pedagogical perspectives, namely, how the order prioritizes specifically/generally fundamental courses, and how it places courses close to their prerequisites.

We conducted the experiments on two course graphs with different courses and prerequisite relationships. The results showed that MDPOrd achieved the best performance on prioritizing related courses when we observe more than 11 and 13 positions of the order for Course Graphs 1 and 2, respectively. For pedagogical metrics, we observed that AggRelOrd and PageRankTS worked well in prioritizing fundamental courses while GVPickings and MDPOrd were effective in ordering courses at a shorter reference distance. Furthermore, we discussed how the course order is generated in different methods with specific examples. We concluded that MDPOrd orders courses from a future-based perspective and keeps a balance among multiple related courses. However, MDPOrd tends to be influenced by the unrelated prerequisites of highly-related courses and ignores the cost of taking previous courses.

In future work, we plan to address the followings: We used courses collected from two curricula in which prerequisite information is available. Thus, we plan to explore whether we can apply our method to order courses in more learning resources such as Massive Open Online Courses (MOOC). In addition, we only evaluated the orders generated by our methods from a pedagogical point of view. It would be more useful if we could use other pedagogical perspectives to further improve the orders. As pointed out as a limitation, current MDPOrd ignores the cost of previous courses when determining the order. It could be a promising direction to integrate the cost into the schemes of reward and transition probability.

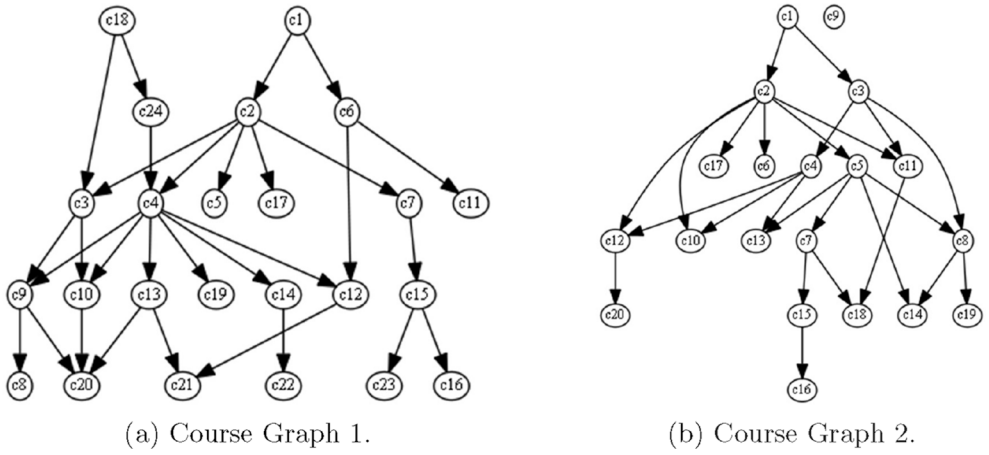
CRediT authorship contribution statement

Yiling Dai: Conceptualization, Methodology, Software, Writing - original draft. **Masatoshi Yoshikawa:** Writing - review & editing, Supervision. **Kazunari Sugiyama:** Writing - review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Course information



(a) Course Graph 1. (b) Course Graph 2.

Fig. A.1. Course graph structures in our experiments.

Table A.1
Course information in Course Graph 1.

ID	Course	Prerequisite ID
c_1	Computer Programming 1	–
c_2	Computer Programming 2	c_1
c_3	Introduction to Computer Systems	c_2, c_{18}
c_4	Data Structures and Algorithm Analysis	c_2, c_{24}
c_5	Mobile App Development	c_2
c_6	Web Site Design and Development	c_1
c_7	Software Architecture & Design	c_2
c_8	Computer Network Security	c_9
c_9	Computer Networks	c_3, c_4
c_{10}	Operating Systems	c_3, c_4
c_{11}	Human Computer Interaction Design	c_6
c_{12}	Advanced Web Design and Programming	c_4, c_6
c_{13}	Database Systems	c_4
c_{14}	Applied Artificial Intelligence	c_4
c_{15}	Software Engineering	c_7
c_{16}	Computing Science Project	c_{15}
c_{17}	Programming Methods	c_2
c_{18}	Discrete Structure 1 for Computer Science	–
c_{19}	Algorithm Design and Analysis	c_4
c_{20}	Distributed Systems	c_9, c_{10}, c_{13}
c_{21}	Web-based Information Systems	c_{12}, c_{13}
c_{22}	Expert Systems	c_{14}
c_{23}	Systems Software Design	c_{15}
c_{24}	Discrete Structure 2 for Computer Science	c_{18}

Table A.2
Course information in Course Graph 2.

ID	Course	Prerequisite ID
c_1	Introduction to Computer Science	–
c_2	Data Structures	c_1
c_3	Introduction to Discrete Structures I	c_1
c_4	Introduction to Discrete Structures II	c_3
c_5	Computer Architecture	c_2
c_6	Software Methodology	c_2
c_7	Systems Programming	c_5
c_8	Principles of Programming Languages	c_3, c_5
c_9	Numerical Analysis and Computing	–
c_{10}	Introduction to Imaging and Multimedia	c_2, c_4
c_{11}	Principles of Information and Data Management	c_2, c_3
c_{12}	Design and Analysis of Computer Algorithms	c_2, c_4
c_{13}	Internet Technology	c_4, c_5
c_{14}	Compilers	c_5, c_8
c_{15}	Operating Systems Design	c_7
c_{16}	Distributed Systems: Concepts and Design	c_{15}
c_{17}	Introduction to Computer Graphics	c_2
c_{18}	Database Systems Implementation	c_7, c_{11}
c_{19}	Introduction to Artificial Intelligence	c_8
c_{20}	Formal Languages and Automata	c_{12}

Appendix B. Technical terminology information

Table B.3
Technical terminologies used in our experiments.

ID	Technical terminology	ID	Technical terminology
t_1	Web application	t_{35}	Computer program
t_2	Object-oriented programming	t_{36}	Web service
t_3	Database	t_{37}	World Wide Web
t_4	SQL	t_{38}	Computer hardware
t_5	Knowledge representation and reasoning	t_{39}	Machine learning
t_6	Programming language	t_{40}	Subroutine
t_7	JavaScript	t_{41}	Graphical user interface
t_8	Software development process	t_{42}	Software bug
t_9	Software testing	t_{43}	Knowledge base
t_{10}	HTML	t_{44}	Unit testing
t_{11}	Operating system	t_{45}	Debugging
t_{12}	Microsoft SQL Server	t_{46}	Data management
t_{13}	Java virtual machine	t_{47}	Agile software development
t_{14}	Software engineering	t_{48}	Modular programming
t_{15}	Java (programming language)	t_{49}	JavaServer Pages
t_{16}	Cascading Style Sheets	t_{50}	Java Platform, Enterprise Edition
t_{17}	Software maintenance	t_{51}	Data structure
t_{18}	MySQL	t_{52}	Test case
t_{19}	PHP	t_{53}	Stored procedure
t_{20}	Formal specification	t_{54}	Algorithmic efficiency
t_{21}	Active Server Pages	t_{55}	Transact-SQL
t_{22}	Web server	t_{56}	Hypertext Transfer Protocol
t_{23}	Computer network	t_{57}	Relational database
t_{24}	Scripting language	t_{58}	HTML5
t_{25}	Unix	t_{59}	Test automation
t_{26}	ASP.NET	t_{60}	Client-server model
t_{27}	Ajax (programming)	t_{61}	Software deployment
t_{28}	Computational problem	t_{62}	Internet protocol suite
t_{29}	Software design pattern	t_{63}	Software project management
t_{30}	Algorithm	t_{64}	Software documentation
t_{31}	Digital signature	t_{65}	Apache HTTP Server
t_{32}	User interface	t_{66}	Version control
t_{33}	Web development	t_{67}	Java servlet
t_{34}	JQuery		

References

Agrawal, R., Golshan, B., & Papalexakis, E. (2016). Toward data-driven design of educational courses: A feasibility study. *Journal of Educational Data Mining (JEDM), 8*(1), 1–21.

Almaleh, A., Aslam, M. A., Saeedi, K., & Aljohani, N. R. (2019). Align my curriculum: A framework to bridge the gap between acquired university curriculum and required market skills. *Sustainability, 11*(9).

Altinel, B., & Ganiz, M. C. (2018). Semantic text classification: A survey of past and recent advances. *Information Processing & Management, 54*(6), 1129–1153.

Bridges, C., Jared, J., Weissmann, J., Montanez-Garay, A., Spencer, J., & Brinton, C. G. (2018). Course recommendation as graphical analysis. In *Proceedings of the 52nd Annual Conference on Information Sciences and Systems (CISS)* (pp. 1–6).

Daiber, J., Jakob, M., Hokamp, C., & Mendes, P. N. (2013). Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th International Conference on Semantic Systems (I-SEMANTICS'13)* (pp. 121–124). Association for Computing Machinery.

Dawid, A. P., & Skene, A. M. (1979). Maximum likelihood estimation of observer error-rates using the EM algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics), 28*(1), 20–28.

Ferragina, P., & Scaiella, U. (2010). TAGME: On-the-fly annotation of short text fragments (by Wikipedia Entities). In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM'10)* (pp. 1625–1628).

Fleiss, J. L. (1971). Measuring nominal scale agreement among many raters. *Psychological Bulletin, 76*(5), 378–382.

Gordon, J., Zhu, L., Galstyan, A., Natarajan, P., & Burns, G. (2016). Modeling Concept Dependencies in a Scientific Corpus. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016) (pp. 866–875). volume 2.

Guo, S., Alamudun, F., & Hammond, T. (2016). RésuMatcher: A Personalized Résumé-Job Matching System. *Expert Systems with Applications, 60*, 169–182.

Jacobsen, A., & Spanakis, G. (2019). It's a Match! Reciprocal Recommender System for Graduating Students and Jobs. In Proceedings of the 12th International Conference on Educational Data Mining (EDM 2019) (pp. 580–583).

Järvelin, K., & Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS), 20*(4), 422–446.

Jeh, G., & Widom, J. (2003). Scaling Personalized web search. In *n Proceedings of the 12th International Conference on World Wide Web (WWW'03)* (pp. 271–279).

Jiang, W., Pardos, Z. A., & Wei, Q. (2019). Goal-based course recommendation. In *Proceedings of the 9th International Conference on Learning Analytics & Knowledge (LAK'19)* (pp. 36–45).

Kapoor, A., & Gardner-McCune, C. (2019). Understanding CS undergraduate students' professional identity through the lens of their professional development. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'19)* (pp. 9–15).

Liang, C., Wu, Z., Huang, W., & Giles, C.L. (2015). Measuring Prerequisite Relations among Concepts. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2015) (pp. 1668–1674).

Ma, B., Taniguchi, Y., & Konomi, S. (2020). Course Recommendation for University Environments. In Proceedings of the 13th International Conference on Educational Data Mining (EDM 2020) (pp. 460–466).

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval.* Cambridge University Press.

Murray, G. (2015). Abstractive Meeting Summarization as a Markov Decision Process. In Proceedings of the 28th Canadian Conference on Artificial Intelligence (Canadian AI 2015) (pp. 212–219).

Paas, F., Renkl, A., & Sweller, J. (2003). Cognitive load theory and instructional design: recent developments. *Educational Psychologist, 38*(1), 1–4.

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank Citation Ranking: Bringing Order to the Web. Technical Report SIDL-WP-1999-0120 Stanford Digital Library Technologies Project.

Pan, L., Li, C., Li, J., & Tang, J. (2017). Prerequisite Relation Learning for Concepts in MOOCs. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL 2017) (pp. 1447–1456). volume 1.

Parameswaran, A. G., & Garcia-Molina, H. (2009). Recommendations with prerequisites. In *Proceedings of the 3rd ACM Conference on Recommender Systems (RecSys'09)* (pp. 353–356).

Pardos, Z. A., & Jiang, W. (2020). Designing for serendipity in a university course recommendation system. In *Proceedings of the 10th International Conference on Learning Analytics & Knowledge (LAK'20)* (pp. 350–359).

Polyzou, A., Nikolakopoulos, A.N., & Karypis, G. (2019). Scholars Walk: A Markov Chain Framework for Course Recommendation. In Proceedings of the 12th International Conference on Educational Data Mining (EDM 2019) (pp. 396–401).

Puterman, M. L. (1994). *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons Inc.

Qin, C., Zhu, H., Xu, T., Zhu, C., Jiang, L., Chen, E., & Xiong, H. (2018). Enhancing person-job fit for talent recruitment: An ability-aware neural network approach. In *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'18)* (pp. 25–34).

Salton, G., & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information Processing & Management, 24*(5), 513–523.

Sayyadiharikandeh, M., Gordon, J., Ambite, J.-L., & Lerman, K. (2019). Finding prerequisite relations using the wikipedia clickstream. In *Companion Proceedings of the 2019 World Wide Web Conference (WWW'19 Companion)* (pp. 1240–1247).

Shi, D., Wang, T., Xing, H., & Xu, H. (2020). A learning path recommendation model based on a multidimensional knowledge graph framework for E-learning. *Knowledge-Based Systems (KBS), 195*, Article 105618.

Srivastava, R., Palshikar, G. K., Chaurasia, S., & Dixit, A. (2018). What's next? A recommendation system for industrial training. *Data Science and Engineering (DSE), 3*(3), 232–247.

Tavakol, M., & Brefeld, U. (2014). Factored MDPs for detecting topics of user sessions. In *Proceedings of the 8th ACM Conference on Recommender Systems (RecSys'14)* (pp. 33–40).

Wang, C., Zhu, H., Zhu, C., Zhang, X., Chen, E., & Xiong, H. (2020). Personalized Employee Training Course Recommendation with Career Development Awareness. In Proceedings of the Web Conference 2020 (WWW'20) (pp. 1648–1659).

Xia, L., Xu, J., Lan, Y., Guo, J., Zeng, W., & Cheng, X. (2017). Adapting markov decision process for search result diversification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'17)* (pp. 535–544).

Xu, J., Xing, T., & Schaar, M.v.d. (2016). Personalized course sequence recommendations. *IEEE Transactions on Signal Processing (TSP), 64*(20), 5340–5352.

Zhang, J., Hao, B., Chen, B., Li, C., Chen, H., & Sun, J. (2019). Hierarchical reinforcement learning for course recommendation in MOOCs. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI-19)* (pp. 435–442).

Zhao, Z., Yang, Y., Li, C., & Nie, L. (2020). GuessUNeed: Recommending courses via neural attention network and course prerequisite relation embeddings. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 16*(4), 132:1–132:17.

Zhu, H., Tian, F., Wu, K., Shah, N., Chen, Y., Ni, Y., Zhang, X., Chao, K.-M., & Zheng, Q. (2018). A multi-constraint learning path recommendation algorithm based on knowledge map. *Knowledge-Based Systems (KBS), 143*, 102–114.