Sequence-based Approaches to Course Recommender Systems

by

Ren Wang

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

A curriculum is a planned sequence of instructions or a view of the student's experiences in terms of the educator's or school's instructional goals. However, the guidance provided by the curriculum is limited and both student and course counsellor struggle with the question of choosing a suitable course at a proper time. Many researchers have focused on making course recommendations with traditional data mining technologies, yet they failed to take a student's history path of taking courses into consideration. In this thesis, we study sequence-based approaches for the course recommender system. First, we implement a course recommender system based on three different sequence related approaches: process mining, dependency graph and sequential pattern mining. Then, we evaluate the impact of the recommender system on undergraduate students in higher education. The result shows that all three methods can improve the performance of students in a certain scale while the approach based on dependency graph contributes most.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Glossary

Apriori-based Approach A SPM approach based on the fact that if a sequence is not frequent, then none of the super-sequences of it is frequent.

Business Process Management (BPM) A discipline that combines knowledge from information technology and knowledge from management sciences and applies this to operational business processes.

Casual Footprint Approach A conformance checking technique in PM that compares two processes with footprint tables.

Collaborative Filtering Approach Recommending a user the items that other users with similar tastes liked in the past.

Confidence An indication of how often the rule has been found to be true in the database.

Content-based Approach Recommending an item that is similar to another item a user or a customer has already taken.

Course History The sequence of a student taking each course including course information and the student's performance on that course.

Course Recommender System (CRS) A recommender system that seeks to recommend the "ideal" courses for a student.

Curriculum The lessons and academic content taught in a school or in a specific course or program. It often refers specifically to a planned sequence of instruction, or to a view of the student's experiences in terms of the educator's or school's instructional goals.

Data Mining The computational process of discovering valuable information, patterns mostly, in large data sets involving methods including artificial intelligence, machine learning, statistics, and database systems.

Dependency Graph (DG) One of the notations to represent a workflow model. It is the most widely used notation because of its simplicity.

Dependency Graph Algorithm (DGA) An algorithm that recommends learning resources based on DG.

Footprint Table A table that records the relations of different activities in process models or event logs.

Fuzzy Mining An process discovery algorithm in PM that can handle the problems of incomplete data and outliers.

GUI (Graphic User Interface) A visual way of interacting with a computer using items such as windows, icons, and menus, used by most modern operating systems.

Item A general term used to denote what the system recommends to users based on users' preference and information.

Key Courses Courses that must be taken in order to graduate for students. Key courses may not show up in the mandatory list of a department's curriculum guideline.

Learning Objectives Brief statements that describe what students will be expected to learn by the end of the study.

Learning Outcomes The actual outcomes of students learning by the end of they study.

Petri Net One of the representations of process model which is consisted by transitions, places and tokens.

PrefixSpan Algorithm The most widely used SPM algorithm which is based on building projected databases.

Process Mining (PM) An emerging technique that can discover the real sequence of various activities from an event log, compare different processes and ultimately identify the bottleneck of an existing process and hence improve it.

Process Model A sequence of operations composed by related, structured activities or tasks to produce a specific service or product or to fulfill a particular goal.

Program of Study An academic plan developed by the school to help students move towards the learning objectives.

Recommender System (RS) A technique and a tool providing suggestions for items valuable to users.

Sequential Pattern A frequent sequence whose statistical significance in the database is above a user-specified threshold.

Sequential Pattern Mining (SPM) A data mining technique of finding all the sequential patterns given a database.

Support An indication of how frequently the itemset appears in the database.

The Alpha Algorithm The first process discovery algorithm in PM but it cannot handle the problems of incomplete data and outliers properly.

Workflow Model See process model.

Chapter 1

Introduction

The term program of study is defined as an academic plan developed by the school to help students move towards the learning objectives, which are brief statements that describe what students will be expected to learn by the end of the study. The term curriculum is defined as the lessons and academic content taught in a school or program. More often, it refers specifically to a planned sequence of instructions, or to a view of the student's experiences and this definition is similar with the program of study. But, the learning outcomes, the actual outcomes of students' learning, may not match the learning objectives. To bridge the gap, some constraints are imposed in the program of study or the narrow definition of the curriculum. Examples are what courses to take at a specific time and prerequisite course relations. However, the guidance provided by the curriculum is very limited. It only tells students some restrictions they must follow. Students are often confused by liberal course options that are not strictly imposed by the curriculum. They can consult their course counsellor with regard to the course selection, but even for course counsellors, they may not have a clear clue of which course is proper for a specific student. Moreover, such advice is time-consuming to be performed manually. Due to this predicament, both students and course counsellors, or educators in a broad view, would appreciate a course recommender system (CRS) to help students achieve their goals and exert their potentials by recommending "ideal" courses for students.

A recommendation of learning resources relies on the recommender system

technique in essence. A recommender system (RS) is a technique and software tool providing suggestions of items valuable for users. The typical approaches to build a recommendation system are content-based approaches and collaborative filtering approaches. A content-based approach is to recommend an item that is similar to another item a user or a customer has already taken or purchased. But, how do we know two items are similar? On one hand, we can train the recommender system with past data to build some feature vectors of all items. On the other hand, we can provide annotation manually which is adding some structure or metadata to learning objects, in our case, courses, to make both human and machine understand. Nonetheless, both methods have their limitations. Normally, we do not have that much training data compared with the products sold online to make feature vectors. Furthermore, annotating each course manually requires significant human efforts, and standards may vary from person to person. Moreover, recommending a course simply because it is "similar" to other courses taken by the student may not be the right thing to do, even if it makes sense for purchased items. Another traditional recommendation approach is collaborative filtering which relies on both previous user ratings and ratings by others. This method is often used in websites or software to provide product recommendations, such as movies, books, etc.

However, both traditional recommendation approaches are not very appropriate for recommending courses due to one unique characteristic: the order of students taking each course. It is not a very beneficial for student to learn a more difficult course without the basic one first. Neglecting the order, there can be a huge gap between the student's course history and the course a recommender system recommends even if the system has found some similarities by other approaches. In addition, some questions that students and educators are eager to know can only be answered if we dive into the sequence of taking courses by students. For example, an educator may want to know what is the "real" academic curriculum, if there are paths seldom used, if current prerequisites make sense, if the particular curriculum constraints are obeyed, or how courses from other departments affect courses inside the department.

The questions students may ask include but are not restricted to: how can I finish my study as soon as possible? Is it more advantageous to take course A before B or B before A? What is the best course for me to take this semester? Will it improve my GPA if I take this course? Answering such questions to both educators and students can greatly enhance the educational experience and process. However, very few course recommender systems currently take advantage of this unique sequence characteristic.

On one hand, students and course counsellors desperately need a course recommender system that fully utilizes the sequence of a student's course history. On the other hand, there is not an available tool to achieve that. This missing link is the motivation of our work. The goal of our thesis is to prove that sequence-based course recommender system is possible and specifically we study three sequence-based approaches to build this recommender system.

Our thesis statements are as follows. First, a course recommender system can be built. Second, such a recommender system can improve students' performance. Third, recommender systems based on different approaches can improve students' performance in various scales. Fourth, this course recommender system can also help course counsellors and educators to gain an insight of the curriculum. My work and contribution include: 1. Exploiting the possible solution to recommend courses with three sequence-based approaches, 2. Building a recommender system based on these three approaches, 3. Implementing a simulator that mimics the course history of undergraduate students in higher education which we cannot access due to ethical approval problems and finally, 4. Comparing and analyzing the results of the recommender system.

The feature that three approaches share is that all of them utilize the sequence of taking courses by students. The first approach our recommender employs is process mining. Process mining is an emerging technique that can discover the real sequence of various activities from an event log, compare different processes and ultimately identify the bottleneck of an existing process and hence improve it. Utilizing the power of process mining, we recommend courses to a student that successful students who have a similar course path

have taken. The next approach is based on the dependency graph. In [10], the authors try to find dependencies among learning resources by the ratings from students. We modify and improve that method to find the real prerequisite relations. Then we recommend courses to students based on what prerequisite courses students have taken. The last approach our recommender relies on is sequential pattern mining. Sequential pattern mining is to find the frequent patterns that are above a certain threshold. By discovering the frequent patterns that successful students have, we can lead the students who need recommendation to greater success.

We conduct several experiments to evaluate our course recommender systems and to find the best recommendation approach. All three approaches can improve students performance in different scales. The best recommendation method is based on the dependency graph, and the number of recommended courses accepted by students have a positive correlation with the performance. Moreover, the course recommender system we build can speed up students' graduation if set properly, and provide some useful insights for educators and course counsellors.

In the next chapter we introduce the related research on recommender systems both in general and in the field of education. For the next three chapters, each will contain its own literature review in regard to the techniques utilized. In Chapter 3, we give a background of process mining techniques then we define our approach of building a course recommender system based on process mining. Chapter 4 gives an introduction of the dependency graph algorithm and then explains the recommender system built on that. A review of sequential pattern mining is presented in Chapter 5 and then a recommender system built by such method is exhibited. Chapter 6 discusses some other aspects of our course recommender systems such as helping students graduate quickly. Chapter 7 is about the course history simulator we implement, the experiments we conduct and the result analysis of those experiments. In the last chapter we conclude our thesis and indicate future work we need to do to further enhance our course recommender system.

Chapter 2

Recommender System

2.1 Review of Recommender System

In this chapter we introduce the background information of data mining, especially in the area of recommender systems. Then we look at the role of recommender systems in education.

The expanding capabilities of information systems and other systems that depend on computing have grown greatly in recent years. Yet most of the data stored is unstructured and organizations have problems dealing with this data. Data mining is expected to resolve this issue in this information age. Data mining is an interdisciplinary subfield of computer science. It is the computational process of discovering valuable information, patterns mostly, in large data sets involving methods including artificial intelligence, machine learning, statistics, and database systems [9].

Among the various topics of data mining, recommender systems or recommendation systems are a major aspect on which this manuscript is based. A recommender system (RS) is both a technique and a tool providing suggestions for items valuable to users [40]. The suggestions relate to various decision-making processes like what items to buy. "Item" is a general term used to denote what the system recommends to users based on users' preference and information. Such needs become imperative especially in this digital age with the explosive growth and variety of information available almost anywhere. By using an RS, a user can shun such overloaded information and choose the most fitting item in ranking in most situations. One example of

why service providers may want to exploit this technique is boosting item sales, which is arguably the most important function of an RS. A customer will buy an item with a higher probability if he or she sees it. Another function of RS is increasing users' satisfaction. An RS can help customers find what they want more easily and people like readily obtainable information. In our case, building a course recommender system can not only improve students' satisfaction and make them study with more initiative, but also benefit the school, counsellors and educators overall.

The necessity of an RS is apparently indisputable, but how does an RS work? According to [8], the taxonomy or the techniques of RS are as follows. The two most frequently used methods are content-based approaches and collaborative filtering approaches. The content-based RS learns to recommend items that are similar with the ones that the user liked in the past. The similarity of items is calculated based on the features associated with the compared items. For instance, if a user has positively rated a movie that belongs to the comedy genre, then the system can learn to recommend other movies from this genre. Collaborative filtering [44] approach is to recommend an active user the items that other users having similar tastes liked in the past. The similarity between the tastes of two users is calculated based on the similarity in the rating history of users. This is the reason why [45] refers to collaborative filtering as "people-to-people correlation". Suppose people who rate bicycles as good also tend to rate safe helmets as good, then it is logical for the system to recommend a safe helmet to a new customer who just bought a bicycle rated it as good. Collaborative filtering is considered to be the most popular and widely implemented technique in RS.

Demographic profile of users can also be valuable to build an RS. For example, users can be dispatched to particular websites based on their language, country or even age [25]. Knowledge-based systems recommend items based on specific domain knowledge about how certain item features meet users' needs and preferences and, ultimately, how the item is useful for the user. A community-based system [22] recommends items based on the preferences of users' friends. This technique follows the epigram "Tell me who your friends

are, and I will tell you who you are" [7]. This technique is very effective in the rise of social network. If an RS is based on the combination of the above mentioned techniques, it is considered to be a hybrid system. It can take the advantages of technique A to fix the disadvantages of technique B.

RS also plays a significant role in education. Not long ago, RS was widely used in commercial web system but was rarely deployed in the learning system. The possibility of applying RS in education especially in the e-learning system is brought in by [63]. Afterwards, many researchers have devoted to this area and details can be found in [26]. The overall goal of most RS in education is to improve students' performance. This goal can be achieved in diverse ways by recommending various things [50]. A common idea is to recommend learning materials and resources such as papers, books and hyperlinks [48] [15] [24]. Enrollment courses can also be recommended [35] [14]. However, most RS only apply the techniques we introduced, i.e., content-based approaches or collaborative filtering approaches. Utilizing sequences to improve students' performance is seldom attempted. Nonetheless, the orders of how students take courses often have a great impact on students' understanding and overall performance. This missing link is what this thesis tries to connect and fill.

2.2 Course Recommender System Structure

As stated, what we build is a course recommender system (CRS) that seeks to recommend the "ideal" courses for students. The overall structure of all entities' interactions in our CRS is shown in Figure 2.1.

Interactions among entities can be categorized as follows:

- Curriculum and Students: Students are free to choose courses they like. The curriculum can help students choose courses from the vast course options they have. The curriculum also indicates some compulsory courses students must take in order to graduate.

- Students and Course history logs: Each course students take are recorded in the course history log. Valuable information includes when students

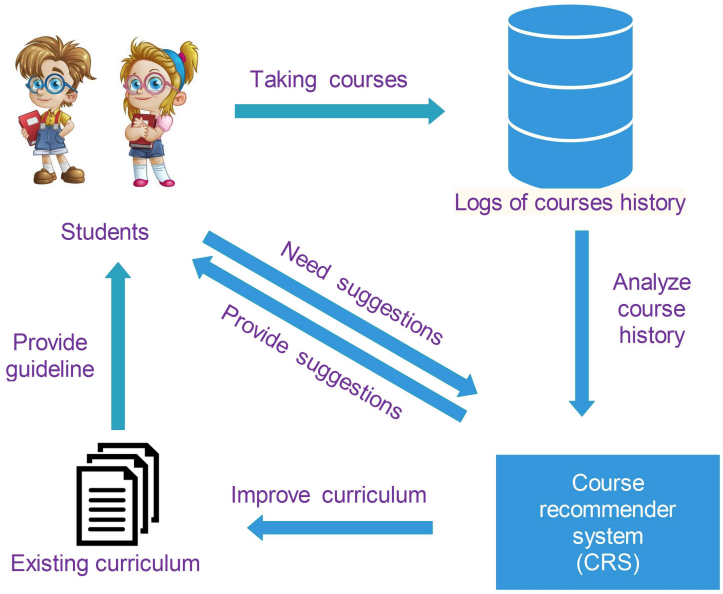


Figure 2.1: The overall structure of all entities' interactions in our CRS

take this course, how students perform and the order in which students take courses.

- Course history logs and CRS: CRS extracts worthy information from the course history log and analyzes it. CRS can summarize some underlying patterns of successful students beneath the vast amount of data.

- Students and CRS: Students may still have questions about which course to take since the guideline a curriculum can provide is very limited. Students can now ask CRS by providing their course history, and CRS will feedback a personalized course plan.

- CRS and Curriculum: The patterns mined by CRS can also be helpful to educators. Educators can improve the curriculum by digging into the patterns found by CRS.

In this thesis, we implement a CRS that can choose the "right" course for each individual undergraduate students in higher education at a specific time based on their past course history. The three sequence-based approaches our CRS relies on are process mining, dependency graph and sequential pattern mining respectively, which will be presented in detail in the following chapters.

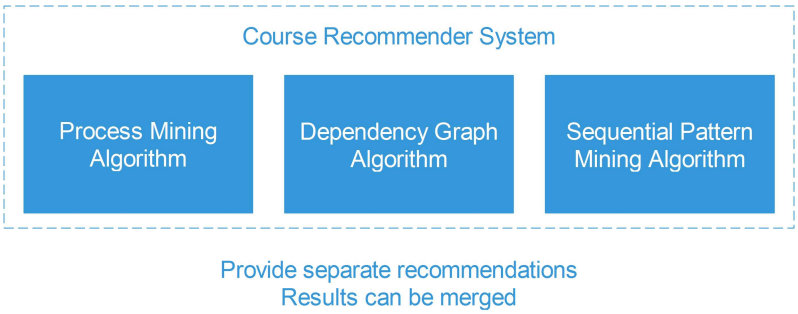


Figure 2.2: Three main approaches inside our CRS

Their structure inside our CRS is shown in Figure 2.2. Each approach provides course recommendations separately. In addition, another algorithm to provide recommendations by merging their results is also implemented.

Measuring students' competencies is a hard problem that many scholars have debated. A model of dimensions of learning is built to assess student learning achievements in [32]. Various scoring rubrics of student's performance criteria are discussed in [5]. The limitations of testing in an assessment system are researched in [62]. Merits and weaknesses of using numerical marks to assess students' performance are presented in [11]. Finally, authors in [43] analyzed interpretations of criteria-based assessment and grading in higher education. As these papers studied, the assessment criteria of students' competencies differ from teachers to teachers, students to students, school to job market. Here we do not intend to argue what is best to gauge students. Although numerous disadvantages in the grading system, there are no simple solutions to replace it. Other student performance criteria are hard to measure especially for a recommender system based on numbers like our CRS and they are universal among different education institutions either. That is the reason we utilize the grading system as a proxy for students' competencies to check whether our CRS works or not. We also implement an algorithm to speed up students graduation by recommending key courses in the early terms which is introduced in Chapter 6. In the future we can add more measurements into our system.

Chapter 3

Course Recommender System based on Process Mining

In this chapter we first present a general view of process mining which includes discovery, conformance and enhancement. Then we go through how we utilize the power of process mining to recommend courses to students in detail. Although we present three sequence methods to improve students' performance in this thesis, they are all related to process mining in some way. We put this chapter first and introduce it in more detail compared to others in order to have a full background knowledge.

3.1 Review of Process Mining

The concept of data mining can be applied to many areas. One area could be Business Process Management (BPM). BPM is a discipline that combines knowledge from information technology and knowledge from management sciences and applies this to operational business processes[52]. Much attention has been drawn to this area due to the significance of its role in increasing productivity and saving costs. For instance, by modelling a business process and analyzing it using simulation, managers may get ideas on how to reduce costs while improving service level. The core of BPM is business process model or workflow model.

A process model is a sequence of operations composed by related, structured activities or tasks to produce a specific service or product, or to fulfill

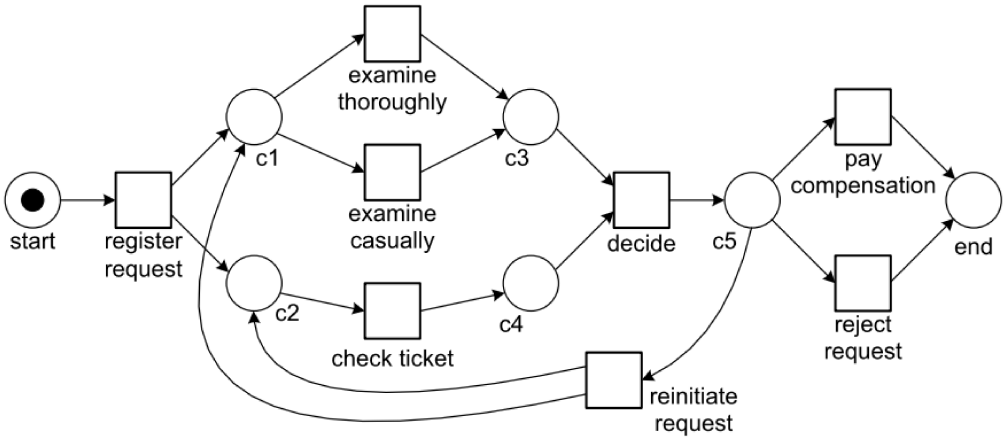


Figure 3.1: A Petri net modelling the handling of compensation requests

a particular goal. An example of process model is shown in Figure 3.1 [53] in terms of a Petri net [39]. The model describes handling a request for compensation of an airline. A Petri net is one of the representations of a process model. Activities in a Petri net are modelled by transitions and are represented by a square. Transitions are connected through places that model possible states of a process. Each place is represented by a circle. In a Petri net a transition is enabled, i.e., the corresponding activity can occur, if all input places hold a token. When the activity executes, also referred as firing, the transition consumes one token from each of its input places and produces one token for each of its output places. Tokens are shown as black dots.

However, a typical BPM workflow model has some serious problems. Firstly, a typical model describes an idealized reality. When modelling processes, the designer tends to concentrate on the "normal" or "desirable" behaviour. For example, the model may only cover 80% of the cases assuming that these are representative. However, this is not the case since the other 20% may cause 80% of the problems. Designers and managers may not be aware of these deviations. Moreover, people may have biased perceptions, depending on their roles in the organization. What is more, hand-made models tend to be subjective. Secondly, it cannot adequately capture human behaviours. For example, a worker who is involved in multiple processes needs to be paid more atten-

tion. This makes it difficult to model one process in isolation. Also, workers do not work at constant speed. The third problem is the abstraction level of the model. Depending on the input data and the questions that need to be answered, a suitable abstraction level is needed.

Thanks to the digital age when nearly all activities of a business are stored in computers, one solution to the problems is to find out what really happened from the massive data. Just like we utilize data mining to obtain the "real" information hidden beneath the vast amount of data, process mining is what is applied in this business process or workflow area. Process mining (PM) is an emerging technique that can discover the real sequence of various activities from an event log, compare different processes and ultimately find the bottleneck of an existing process and hence improve it [53]. To be specific, PM consists of extracting knowledge from event logs recorded by an information system and discovering business process from these event logs, comparing processes, and providing suggestions for improvements in these processes. PM techniques are often used when there is not a formal description of the process and it often can provide a visualization with a flowchart as a sequence of activities. To conclude, the three main tasks of PM are discovery, conformance and enhancement. Figure 3.2 shows the positions of three main types of PM: discovery, conformance, and enhancement. PM can be deemed as part of data mining, a specific direction that targets the process model, normally in the business areas.

Getting and cleaning data, which are called event logs in PM, are always the first step of PM. It is significant for three reasons. First, most of the time we cannot directly handle the event logs because they are often messily stored in the database. Second, there are some data that need to be filtered or treated specially. Third, a universal form of event logs is needed if we are going to apply standard PM approaches or software. Until recently, the de facto standard for storing and exchanging events logs was MXML (Mining eXtensible Markup Language). MXML emerged in 2003 and was later adopted by the PM tool ProM [60]. It uses an XML-based syntax. The current standard of event logs in PM field is XES which is the successor of MXML [19]. Based

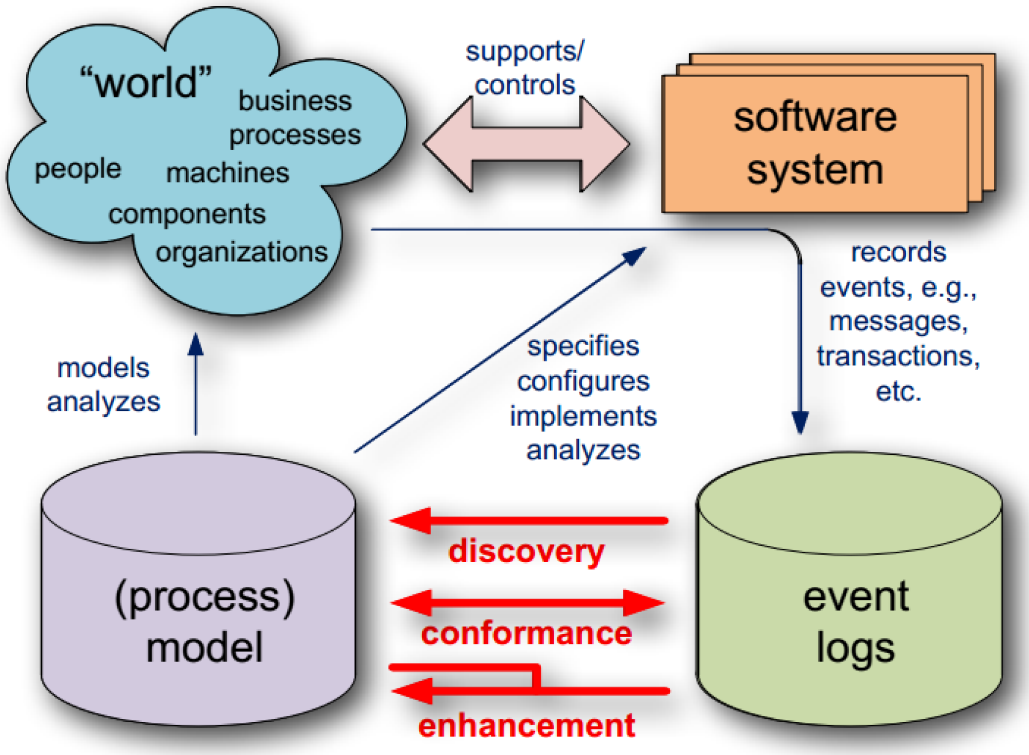


Figure 3.2: Positions of three main types of PM: discovery, conformance, and enhancement [53]

on many practical experiences with MXML, the XES format has been made less restrictive and truly extensible. Figure 3.3 shows an example log in XES format. In this example, we can see some label names like trace and event. If we have a student who takes a sequence of courses, then each sequence is a case, called a trace in XES. An event is what we know as an action or an activity. In the student instance we mentioned, an event is taking a course. Certainly, trace and event can have their own properties or attributes. The attributes used most are the names (course name), who performed the events (professors), the time of the event (term), what resources are needed etc.

The next step is process discovery which is to find the real workflow, instead of an ideal written description, behind a process from event logs. Although it is often performed before the other two tasks, that is not always the case. We will discuss the numerous discovery techniques proposed so far in the next chapter since the discovery part is more related to the dependency graph algorithm.

```
<trace>
   <string key="concept:name" value="1"/>
   <event>
      <string key="concept:name" value="register request"/>
      <string key="org:resource" value="Pete"/>
      <date key="time:timestamp" value="2010-12-30T11:02:00.000+01:00"/>
      <string key="Event_ID" value="35654423"/>
      <string key="Costs" value="50"/>
   </event>
   <event>
      <string key="concept:name" value="examine thoroughly"/>
      <string key="org:resource" value="Sue"/>
      <date key="time:timestamp" value="2010-12-31T10:06:00.000+01:00"/>
      <string key="Event_ID" value="35654424"/>
      <string key="Costs" value="400"/>
   </event>
   <event>
      <string key="concept:name" value="check ticket"/>
      <string key="org:resource" value="Mike"/>
      <date key="time:timestamp" value="2011-01-05T15:12:00.000+01:00"/>
      <string key="Event_ID" value="35654425"/>
      <string key="Costs" value="100"/>
   </event>
```

Figure 3.3: An example log in XES format

Conformance in the context of PM is finding the discrepancy between different logs and models, comparing the observed behaviour with modelled behaviour. It can help answer questions like "Did the reality follow the guideline we wanted". For the comparison, it can be between the found model and the ideal model. It also can be performing a "replay": replaying the event logs on the ideal process model. With these techniques, we can highlight the exact places where the model and the log are inconsistent. An illustration is shown in Figure 3.4. When deviations are detected, we can view them in two angles. If these deviations occurred in the event logs are correct, then there must be an error with the model, whether it is not including all scenarios or it simply is not correct. The other possibility is that the deviations in the log should not happen. In the reality it is due to incorrect operation of human.

Enhancement is identifying the bottleneck of a particular process and providing recommendations to improve the performance based on the discovered processes and their comparisons. This often includes viewing the process model in several perspectives. The first perspective is the organizational per-

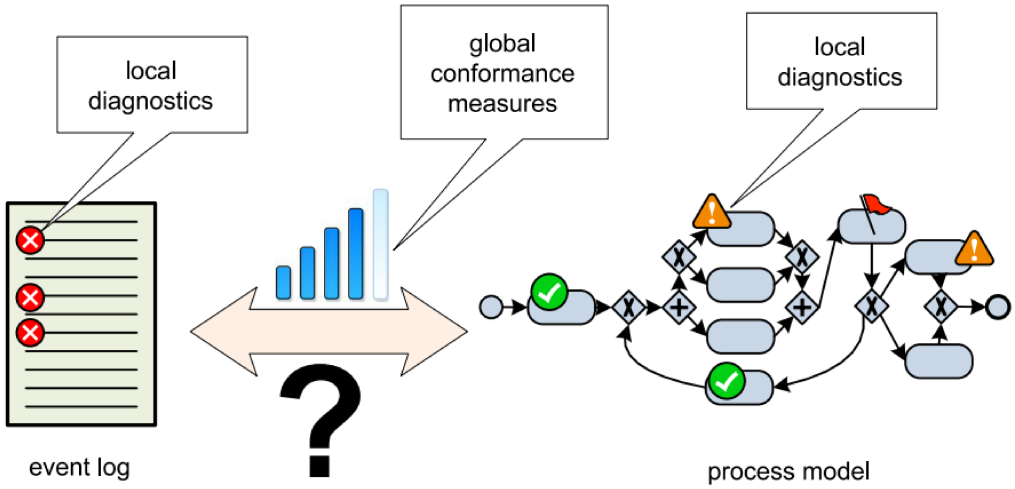


Figure 3.4: PM conformance illustration [53]

spective which focuses on how the work is handed among different resources like workers and how the resources resemble with each other. The second perspective is time and probability which include waiting time, frequency events and bottleneck identification. The third perspective is decision mining: how the decision is made when facing crossroads. Gaining these perspectives can provide two important operational support. One is prediction: "What is the remaining time?", "What is the possibility that activity A will occur?". The other operational support is making recommendations to minimize or maximize something for benefits. That is what we are trying to do for students in this manuscript.

PM can be applied to many areas such as education and medical areas. We will introduce some of them briefly in the following parts, especially in education.

Some attempts have already been made to exploit the power of PM in curriculum data particularly. For instance, authors of one chapter in [41] give a broad introduction of PM and indicate that it can be used in educational data. However, the description is too general and not enough examples are given. The first paper that proposes to utilize PM on curriculum data is [51]. The authors point out the significant benefit in combining educational data with PM. The main idea is to model a curriculum as a coloured Petri net

using some standard patterns. However, most of the contribution is plain theory and no real experiment is conducted. [36] directly targets curriculum data and brings up a notion called curriculum mining. Similar with the three components of PM, it clearly defines three main tasks of curriculum mining, which are curriculum model discovery, curriculum model conformance checking and curriculum model extensions. The authors explain vividly how curriculum mining can answer some of the questions that teachers and administrators may ask.

Researchers have also focused on educational data with PM, though not particularly on curriculum data. One example is [6] in which the authors present a platform, SoftLearn, that is able to discover complete, precise and simple learning paths from event logs. The paths they find are the sequences of a student's behaviour during a specific week of one course, while our emphasis is curriculum mining which is the sequence of a student taking different courses. Assessment data of online multiple choices is analyzed in [37]. Still, it only focuses on events related to the flexibility of navigation feedback requests for the question items. The modelling approach does not explain how the navigation through different questions impact test results. Another example is [46]. The authors try to explore the relationship between the students' performance and regulatory activity during a computer-supported collaborative learning task using PM. The result is that no major differences are found in process models between high-achieving and low-achieving students. Again, these papers among others are just showing the possibility to apply PM to educational data.

Another area that PM is used is health care data. The process of patients going through different departments of a hospital and doing different kinds of examinations and treatments can be analyzed through PM. PM in this area in general, especially the data type and some frequently asked questions, is discussed in [29]. Simulated data of patients that require different specialties (multidisciplinary patient) are analyzed in order to automatically discover the workflow of the hospital in [30]. Similarly, [28] also aims to obtain meaningful knowledge about "careflows", e.g., to discover typical paths followed by par-

ticular groups of patients. Finally, some real stroke patients cases are dealt with PM in [27] bearing the objective to discover the procedures for treating them in different hospitals, and it demonstrates the applicability of PM in the health-care domain.

Besides in education and hospital areas, scholars also attempt to apply PM in many other areas. Authors discuss the possibilities of employing PM to the human interaction in public works department with a goal of productivity improvement [55]. Authors in [17] brought PM to machines instead of human, i.e., remote servicing network of Philips Healthcare (formerly known as Philips Medical Systems). They find some surprising discrepancies and improve test scripts by analyzing deployed application usage data. Another example of PM in machine is [42]. It is reducing the test period of manufacturers of chip-making equipment by examining the unnecessary loops. PM can also be helpful to many other areas such as web service [57] [54] and supply chain [31].

Currently, there are some tools available to help dive into PM immediately. The most famous one is ProM[59]. It is a free software for academia mostly. Another commercial PM tool is Disco [16], which has a very nice interface and the analysis of it is straightforward. These tools shed light into data with PM perspective. However, the insights they provide are too general and that is the reason we have to build a tool targeting students' data by ourselves.

3.2 Implementation of Course Recommender System based on Process Mining

In this section we introduce our CRS based on PM, i.e., we recommend courses to a student that successful students who have a similar course path have taken. As briefly mentioned in the last section, PM is an emerging technique that can discover the real sequence of various activities from an event log, compare different processes and ultimately find the bottleneck of an existing process and hence improve it. According to this definition, there are three classes of PM techniques: discovery, conformance checking and enhancement. In the discovery part, a PM discovery algorithm will attempt to identify the

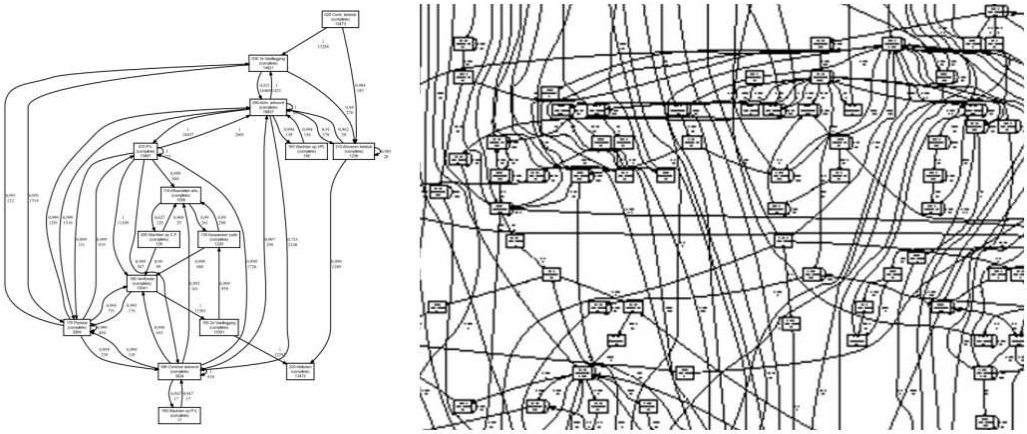


Figure 3.5: An example of two types of processes

real orders of activities. However, our course data are different from typical PM data at least in the following three aspects.

The first aspect is the order of the activities. Generally, in PM data, the order of activities is relatively determined. One example is software development. When we develop a software, the steps we often take are demands analysis, system structure design, detail design, coding, testing, delivery and maintenance. We cannot skip to testing without coding. Certainly, the process in the real world is more complicated. We may skip or change the order of some activities in reality, but there is a general order overall. However, this is not the case for our data, the history of students taking courses. Students are quite free to take the courses they like and they do not follow a specific order. Granted that there are restrictions such as prerequisite courses or the courses we need to take in order to graduate, these dependencies are relatively rare compared with the number of courses available. To make it clearer, a normal process would be like the left one of Figure 3.5, a graph with many branches yet there is a relatively clear path. However, our data looks like the figure on the right, a messy graph with no strict order, which is vividly called the spaghetti process due to its intertwined property. The PM application papers we mentioned before often face this kind of graph. Yet their solutions to this problem are more from a statistical view rather the messy graph itself.

The second aspect is the dependency length. In the example of software

development, coding must be done before testing. If we are at the testing phase, we may safely conclude that the design phase is already done. It is because testing depends not only on coding but also on design since design is before coding. This is an intuition of knowing which phase we are at in the process. However, in the course history data, we do not have such a long dependency. We may have a prerequisite requirement, e.g., we must take CMPUT 174 and CMPUT 204 first in order to take CMPUT 304, but such dependency is very short. We may conclude the phase of our study by the starting number of the course, e.g., taking CMPUT 304 indicates the third year, but such information is not general and accurate.

The third difference is the type of activities in the sequence. Data from typical PM problem are the sequence of single activities, while in our case they are the sequence of sets. Students can take several courses in the same term, which makes it more difficult to represent in a graph.

For the reasons above, we do not attempt to build a dependency graph in our PM method and we directly apply the second class of PM, namely conformance checking, to our CRS. Our CRS can also be considered as the enhancement class of PM since the goal of our CRS is to recommend courses that can boost students performance.

The intuition behind our algorithm is to recommend the path that successful students take, i.e., to recommend courses taken by the students who are both successful and similar to our students who need help. This rationale is shown in Figure 3.6. We achieve this by the steps in Algorithm 1.

Here is the explanation of the above algorithm. Suppose we have a log of finished students and a student who needs recommendations. In the program, we find all course histories of finished students whose GPA is above a certain threshold. We define them as successful students and we do not take other students, less successful students, into consideration. Next, we further filter the results from successful students and only keep successful students who are similar to this specific student based on a similarity metric. Now that we have students who are both successful and similar to this student, we build a list of all the courses those students take next. We call these courses candidate

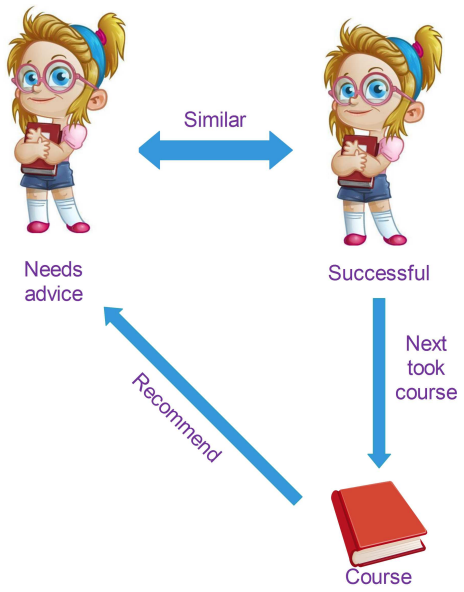


Figure 3.6: The rationale behind our PM algorithm

courses. There are probably many candidate courses while only a limited number of recommendations is needed. The last step is to rank them based on some metrics and choose the top courses to recommend. The metrics can be course GPA contribution, whether the course is a key course or not (courses students must take) or a hybrid method. Details of such metrics are explained in Chapter 6.

The method we use to compute the similarity between two students is an improved version of the casual footprint approach. The casual footprint approach is a conformance checking technique in PM that compares two processes. In the original casual footprint approach, it first builds two footprint tables that record the relations of different activities based on two process models. Then it computes the number of differences between two footprint tables. Our approach improves the original one in the following two aspects. Firstly, we modify the approach to suit our data: the original approach is used on two process models while our method applies to data directly which is the sequence of sets. Secondly, we define some new relations among activities, courses in our case, due to the special attributes of course history and the sequence of set.

We define the following relations between two courses:

Algorithm 1 Algorithm of CRS based on PM

Input :

Logs L of finished students course history

Student stu who needs course recommendations

Execute :

1: Find all high GPA students from L as HS
2: Set candidate courses $CC = \emptyset$
3: **for all** $stuHGPA$ in HS **do**
4: Apply Algorithm 2 to compute the similarity sim between stu and $stuHGPA$
5: **if** sim is greater than a certain threshold **then**
6: Add courses that $stuHGPA$ take next to CC
7: **end if**
8: **end for**
9: Rank CC based on selected metrics
10: Recommend the top courses from CC to stu

Table 3.1: Course histories of *studentA* and *studentB*

Student	Term 1	Term 2	Term 3	Term 4
A	174	175	201, 208	
B	174	175	204, 208	304

- Direct succession: $x \rightarrow y$ iff x is directly followed by y
- Indirect succession: $x \rightarrow\!\!\rightarrow y$ iff x is indirectly followed by y
- Reverse direct succession: $x \leftarrow y$ iff y is directly followed by x
- Reverse indirect succession: $x \leftarrow\!\!\leftarrow y$ iff y is directly followed by x
- Same term: $x \parallel y$ iff x and y are in the same term
- Other: $x\#y$ for Initialization or if x and y have the same name

The example below illustrates these relations. Suppose we want to compare the similarity between *studentA* and *studentB* whose course histories are shown in Table 3.1. Then the relations between courses of *studentA* include but are not limited to: 174#204 (204 is not in the course history of *studentA*), $174 \rightarrow 175$ (174 is immediately followed by 175), $174 \rightarrow\!\!\rightarrow 201$ (174 is followed but not directly by 201), $201 \leftarrow\!\!\leftarrow 174$.

With the relation terms defined, we can proceed to our improved version of the footprint algorithm which computes the similarity of two course history sequences.

Algorithm 2 Algorithm of computing the similarity of two course history sequences

Input :
Course history sequence of the first student s_1
Course history sequence of the first student s_2
Output :
1: Truncate the longer sequence to the same length with the shorter sequence
2: Build two blank footprint tables that map between s_1 and s_2
3: Fill out two footprint tables based on s_1 and s_2
4: Calculate the total elements and the number of elements that are different
5: Compute the similarity
6: Return the similarity of s_1 and s_2

In most cases, finished students' course histories are much longer than the current students'. To eliminate this difference we truncate the longer sequence to the same length of the shorter sequence. The next step is to build a one-to-one mapping of all courses in both sequences. Our CRS computes the above defined relations based on the two sequences and fills the relations in the footprint table separately. Lastly, our CRS calculates $differenceCount$ which is the number of elements in footprint tables that s_1 differs from s_2, and $totalCount$ which is the total number of elements in one footprint table. With those numbers, $similarity$ is

$$similarity = 1 - \frac{differenceCount}{totalCount}.$$

Figure 3.7 is the footprint tables of $studentA$ and $studentB$. The table does not contain 304 since it has been truncated in the first step. $differenceCount$ is 12 and $totalCount$ is 25 if we check the cells in both tables. Therefore, the similarity between the two sequences is:

$$similarity = 1 - \frac{12}{25} = 0.52$$

Figure 3.8 shows the whole algorithm workflow of our CRS based on PM. When we were conducting the experiments, it took far too long to recommend

	174	175	201	204	208
174	#	→	→→	#	→→
175	←	#	→	#	→
201	←←	←	#	#	||
204	#	#	#	#	#
208	←←	←	||	#	#

Footprint table of student A

	174	175	201	204	208
174	#	→	#	→→	→→
175	←	#	#	→	→
201	#	#	#	#	#
204	←←	←	#	#	||
208	←←	←	#	||	#

Footprint table of student B

Figure 3.7: Footprint tables of *studentA* and *studentB*

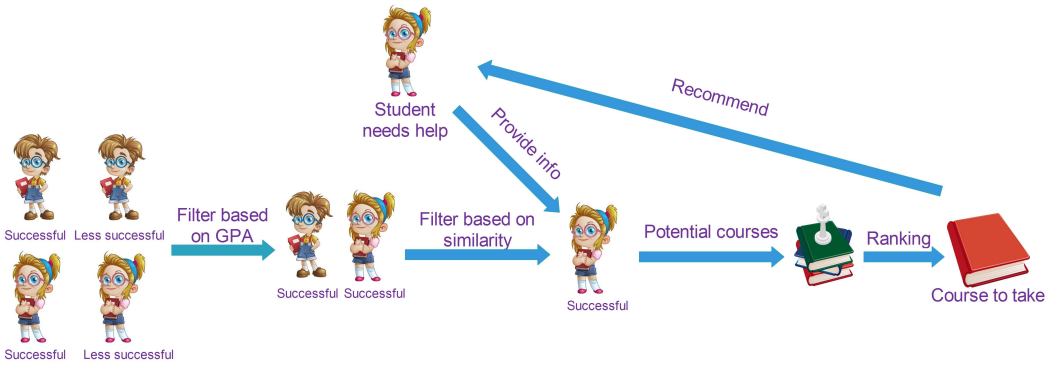


Figure 3.8: The overall workflow of our CRS based on PM algorithm

courses to students, so we made some modifications to speed up the recommendation process of this approach. Firstly, we used to recommend one course to a student at a time even if several courses are needed in one term. The number of repeating processes even in one term to one student would take about four times. Now instead, we recommend the courses in one term together. Secondly, if we need to recommend courses to students who we have recommended to before, we do not need to compare them to all other successful students again. We only compare them to students who are similar to them in the previous recommendation round. Because if a path is not similar with another path before, it is not very possible to become similar by adding another activity, provided that the path is already long enough. This is also for the sake of less computation.

Chapter 4

Course Recommender System based on Dependency Graph

In this section we look at our CRS based on the dependency graph. Some algorithms of finding the dependency graph are reviewed. Broadly speaking, we can categorize the dependency graph algorithm into the discovery part of PM. Therefore, we introduce the discovery approaches in this chapter. We choose and improve one of them to build our CRS.

4.1 Review of Dependency Graph

How to discover a process, which we did not go into detail in the last chapter, is more closely related to this section, the dependency graph algorithm. By examining the data, the result from process discovery is a workflow model represented in a graph. The basic elements of this graph are sequences and choices. Sequences define the order of each activity in the work and the choices define in certain cases we choose one way over another. As long as a graph model possesses these two elements, it can be a result of a process discovery, just different notations. Examples of such graph models include Transition System [56], Petri net [39], YAWL [49], Buisiness Process Modelling Notation (BPMN) [13], Event-Driven Process Chains (EPC) [34] and Dependency Graph (DG). Surely, each notation has its own advantages and limitations. The key things to be considered for a process notation are whether it can represent concurrency, soundness and patterns (AND, OR, XOR etc.). These

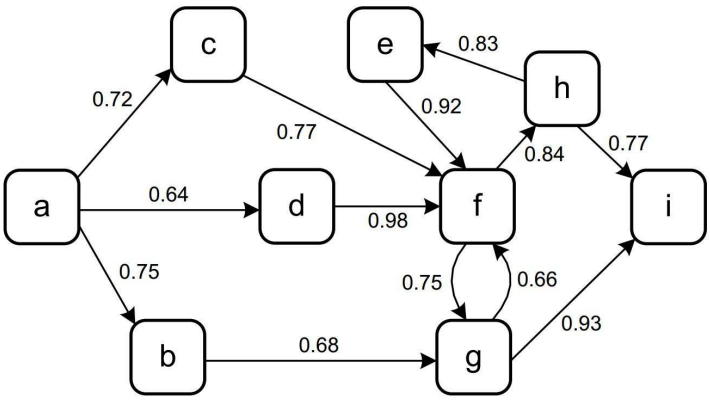


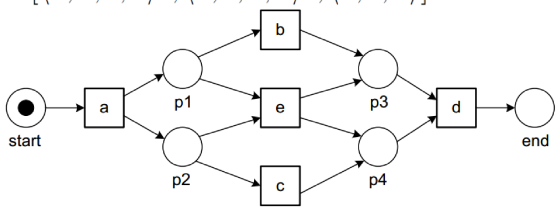
Figure 4.1: An example of DG

functions are not strictly required since the best model is a model that is suitable for its task. As one of the notations, the dependency graph (DG) is the simplest yet most widely used. Figure 4.1 is a DG example from with various activities from a to i and edges with weights. The edges do not necessarily have weights, but the key in a DG is edges connecting different activities to demonstrates dependencies. Normally, the representation of a workflow model does not affect how we analyze the problem too much and different notations can be converted. Even though the following discovery methods introduced often have a preferred result notation, it is not a big problem.

A primitive method to utilize data to discover DG is stated in [2]. It is an algorithm that, given a log of unstructured executions of a process, generates a graph model of the process. The dependency relation is based on the intuition that for two activities A and B, if B follows A but A does not follow B, then B is dependent on A. If they both follow each other in the data, they are independent. This simple intuitive idea lays the foundation for many following PM algorithms though this paper [2] itself leaves many complicated questions unanswered.

In some situations, a DG is not enough to deal with the problems. A notation that is able to reveal concurrency and different choice features is needed, and the Petri net [39] satisfies this criteria. To correspond with the Petri net, a more complicated mining method is proposed: the Alpha Algorithm [58]. The Alpha Algorithm is a widely known discovery approach that is represented in

$L_1 = [\langle a,b,c,d\rangle^3, \langle a,c,b,d\rangle^2, \langle a,e,d\rangle]$



	a	b	c	d	e
a	$\#_{L_1}$	$\rightarrow_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$	$\rightarrow_{L_1}$
b	$\leftarrow_{L_1}$	$\#_{L_1}$	$\|\|_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$
c	$\leftarrow_{L_1}$	$\|\|_{L_1}$	$\#_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$
d	$\#_{L_1}$	$\leftarrow_{L_1}$	$\leftarrow_{L_1}$	$\#_{L_1}$	$\leftarrow_{L_1}$
e	$\leftarrow_{L_1}$	$\#_{L_1}$	$\#_{L_1}$	$\rightarrow_{L_1}$	$\#_{L_1}$

Figure 4.2: Relations of event logs, the Petri net and the foortprint table

a Petri net. This algorithm can be viewed as a milestone of PM because it is the first algorithm that deals with concurrency in PM and it sparked the whole research area of PM. The Alpha Algorithm first finds four log-based ordering relations (Precisely one of these relations holds for any pair of activities):

Let L be an event log over activities A. Let a, $b \in A$:

- $a > b$ if and only if there is a trace $\sigma = \langle t_1, t_2, t_3, ..., t_n\rangle$ and $i \in \{1, ..., n1\}$ such that $\sigma \in L$ and $t_i = a$ and $t_{i+1} = b$
- $a \rightarrow b$ if and only if $a > b$ and $b \not> a$
- $a \# b$ if and only if $a \not> b$ and $b \not> a$
- $a \parallel b$ if and only if $a > b$ and $b > a$

Based on the relations of all activities in the log (an example is shown in Figure 4.2), one can find typical process patterns which are shown in Figure 4.3. These choice patterns indicate how a case deals with choices if it reaches this point. For instance, an XOR-split pattern means after activity a only one of activity b and c can be carried out while an AND-split indicates both b and c can occur after a (the order between b and c is not guaranteed). We can inference that an XOR-split pattern of a, b, c three activities is equal to the relation that $a \rightarrow b$, $a \rightarrow c$ and $b\#c$. Other patterns can be deducted

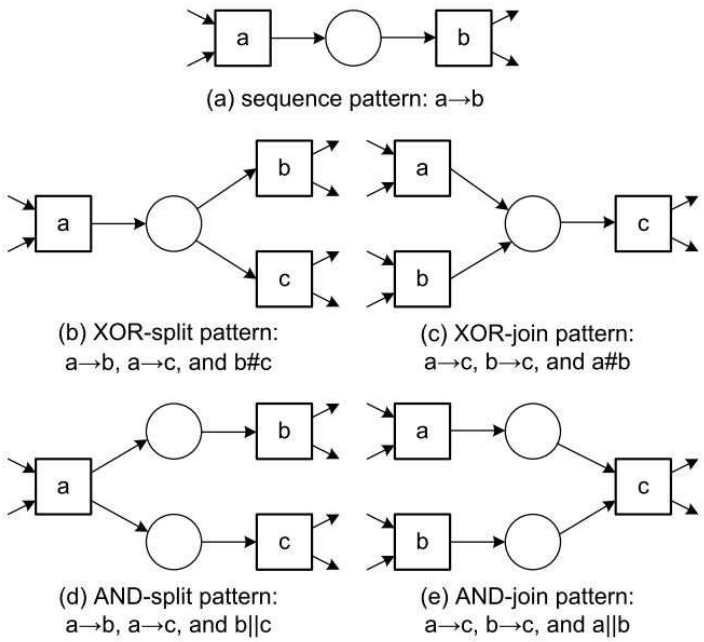


Figure 4.3: Typical process choice patterns and corresponding footprints

similarly. Utilizing this principle, one can build a whole Petri net from the footprint table that records the relations found in data.

With so many benefits being mentioned, the Alpha-algorithm has serious problems with incomplete data or outliers. Due to such limitations, the Alpha-algorithm is seldom applied with real data. In the real world, data tend to be chaotic and disordered. Researchers in PM field vividly call the two different kinds of data as "Lasagna" and "Spaghetti" respectively. The Lasagna process is structured, regular, controllable and repetitive. The opposite is the Spaghetti process which represents unstructured, irregular, flexible and variable. Most data in the real world belong to the latter genre which requires an approach tolerating the messy situations. That is why in reality the heuristic mining approach [61] and the fuzzy mining approach [18] are the most widely used discovery techniques. They are resilient with incompleteness and outliers. Fuzzy mining is an improved version of heuristic mining and its idea is as following:

1. Highly significant behaviour is preserved, i.e., contained in the simplified model.

2. Less significant but highly correlated behaviour is aggregated, i.e., hidden in clusters within the simplified model.

3. Less significant and lowly correlated behaviour is abstracted from, i.e., removed from the simplified model.

We can imagine the fuzzy mining algorithm as a map and the type of map is subject to our need. We may need a bicycle map instead of a driving map when we are riding a bicycle. Similarly, a general overview of countries is preferred instead of street details when we want to figure out the location structure of countries. The word "fuzzy" means to omit certain details when they are not important. That is exactly how "spaghetti" process should be dealt with. If activities are very important, they should be preserved. If activities are not so important but are highly correlated, they should be aggregated as one activity. If activities are neither important or correlated, they should be ignored. These rules are also applicable to the edges.

Other than the Alpha Algorithm and the fuzzy mining algorithm, some researchers use genetic PM technique [12] and region based mining. Genetic PM technique adopts an approach that mimics the process of natural evolution: treating process models as individuals and improving their quality by combining them iteratively.

The approach we adopt is none of the above. Certainly, these approaches have their advantages, but they are not quite suitable for our task. Our method is based on [10]. We explain the limitations of approaches above, why we adopt [10] and how we modify it to suit our case in the next section.

4.2 Implementation of Course Recommender System based on Dependency Graph

In this section we introduce our CRS based on the dependency graph algorithm (DGA). As we briefly mentioned in Chatper 2, one could provide recommendations by two major approaches, namely, the content-based approach and the collaborative filtering approach. However, in education these methods cannot

be applied well due to three limitations. One is the order. There is no point to learn the advanced course without the basic one first. The second limitation is the number of resources which is relatively finite in education. Compared with the vast potential items like movies or products that we need to scan, the learning resources, courses in our case, are limited in number. The last method limitation is that recommending a course simply because it is "similar" to other courses taken by the student may not be the right thing to do, even if it makes senses for purchased items.

The method which our CRS adopts is described in [10]. The authors developed an approach of recommending of learning resources for users based on users' previous feedback. It learns DG by users ratings. This method can be applied to other resources. The goal of [10] is to resolve both limitations. Specifically, it achieves this in three steps. The first step is growing the pool of resources: gathering data. Users can take learning resources from the database at will. The second step is obtaining the feedback. Learners are required to give a rating or usefulness of the resources they used. The database evolves by filtering learning objects with low ratings as time goes by. The third step is mining the dependencies based on ratings. This mining is inspired by association rule mining and sequential pattern analysis and we will introduce it later in this section. The last step is generating intelligent suggestions based on resources learners have seen, resources other learners have liked.

Consider an example of how to recommend resources to learners based on what learners have seen. For this sample scenario we use the dependency graph shown in Figure 4.4. The pool of resources contains eight resources which are labeled A to H. Each edge in this graph represents a dependency among resources. For example, E depends on either D or C. A special case is the dependency for C, which depends on both A and B, but not in any particular order. Each dependency represents what a learner should have seen in order to find the current resource useful. All of these resources are targeted at a particular topic. Now a learner who wishes to learn logs into the system for the first time. Initially, the learner has not seen any resources, so the system will select suitable resources for the learner to begin with. The dependency

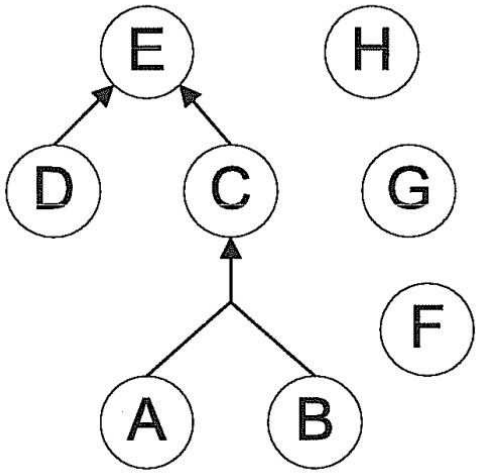


Figure 4.4: An example of course DG

Seen	$\emptyset$
Suggested	$A, B, D\ (C, E)$
Other	F, G, H

(a) Iteration 1

Seen	B
Suggested	$A\ (C, E)$
Other	D, F, G, H

(b) Iteration 2

Seen	B, G
Suggested	$A\ (C, E)$
Other	D, F, H

(c) Iteration 3

Seen	B, G, A
Suggested	$C\ (E)$
Other	F, H

(d) Iteration 4

Seen	B, G, A, C
Suggested	E
Other	F, H

(e) Iteration 5

Seen	B, G, A, C, E
Suggested	$\emptyset$
Other	F, H

(f) Iteration 6

Figure 4.5: A learner's interaction with DG learning system

graph is consulted and resources at the leaves, which are A, B and D, are suggested to the learner. There are resources that the learner should not see at this point as well, which are C and E, written next to these suggestions in brackets. Finally, any resource that is not connected to the graph is not hidden from the user and is added to the "other" list. Figure 4.5 (a) shows the current state of the suggestions for the user. The learner selects resource B out of the suggested resources. As the dependency graph contains a dependency { A, B} $\rightarrow$ C , it will suggest A next since B is considered to complement A. This is reflected in Figure 4.5 (b). Figure 4.5 (c) shows that the learner selected G, which does not change the suggestions. Once the learner has seen the suggested resource A (Figure 4.5(d)), C is selected, shown in Figure 4.5 (e) and finally, E in Figure 4.5 (f).

The above analysis presumes that we have a DG prior to recommending learning resources, yet this DG needs to be built. The following is the rationale

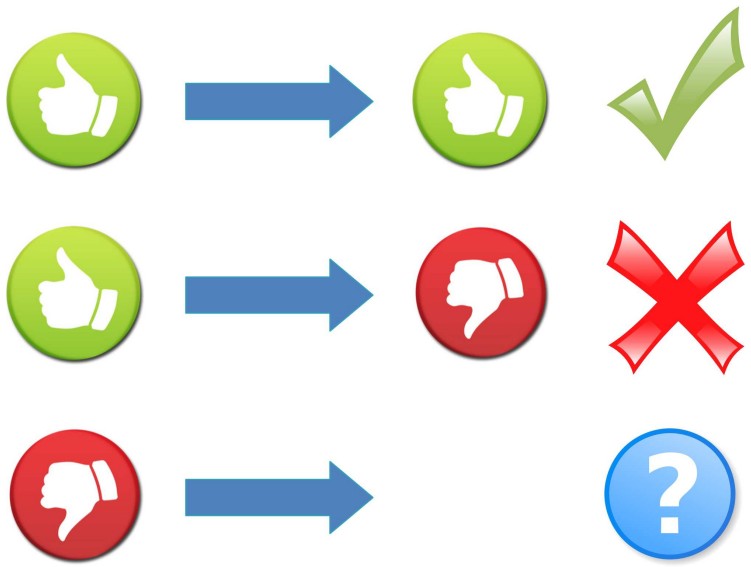


Figure 4.6: The rationale behind the original DG algorithm

behind the algorithm of mining a DG algorithm proposed by the authors and it is illustrated in Figure 4.6.

1. Any positively rated item j that appears before a positively rated i implies j is a dependency for i.

2. Any positively rated item j that appears before a negatively rated i implies j is not a dependency for i.

3. Any negatively rated item j before i is ignored.

Based on these rationales, the specific steps of discovering the DG are described as Algorithm 3.

Now we apply the steps above to a simple dataset shown in Table 4.1. The first step is to take each item and find the items with a positive rating that preceded them in each record. This is done for both positively and negatively rated items. For example, the first record contains a positive B and there is only one positive item, A, preceding it. Thus, considering only record 1, $P_{B+} = \{ A \}$. This relationship can be used to imply a dependency between A and B, $(A \rightarrow B)$. Note that if in a subsequent record, we also find a $(B \rightarrow A)$ dependency, then this will cancel out the previous one, resulting in no

Algorithm 3 Algorithm of original DG discovery

Input :

Course material item set I

Execute :

1: **for all** item i from I **do**

2: Create two projected datasets P_{i+} and P_{i-} for the positively and negatively rated item i_+ and i_- respectively. A projected dataset for item i is the set of positive items occur before i and in each sequence of the source dataset.

3: Remove from P_{i+} item sets in P_{i-} as they do not impact the ratings of i, creating the potential dependency set D_i

4: Find the smallest set of item sets from D_i which describes the dependencies.

5: **end for**

dependency between A and B. The project datasets for the three items, A, B and C are shown in Figure 4.7.

Next, we work out D_i which is defined as P_{i+} with all itemsets of P_{i-} removed. As i is not dependent on itemsets in P_{i-}, then by removing these from P_{i+} we only consider the items that may be dependent on i. For example, if $P_{A+} = \{ B, C, D \}$, $P_{A-} = \{ B, C \}$, then $D_A = \{ D \}$ as this is the only set of items that matters when A is not dependent on $\{ B, C \}$.

After that we take the smallest subset of the potential dependencies, D_i. This involves finding the smallest set of items which cover the full dependencies for the given item. Given any sequence X and Y if X is contained within Y, we can remove Y. For example, given that the current item is A, and $X = B$, C and $Y = \{ B, C, D \}$ where $\{ X, Y \}$ belongs to D_A, we can remove Y as X is contained within Y. This means that A will be positive if positive items B and C exist before it in sequence. As this subsequence exists in both X and Y, we simply keep the smaller one of the two and discard the other. Finally, we build the DG. Every item i is dependent on each item within Y, given Y belongs to D_i. For our example, $A \rightarrow B$, $B \rightarrow C$. Thus, the DG is $A \rightarrow B \rightarrow C$ for our example.

Admittedly, there are several drawbacks of this method. Firstly, the dependency structures authors tested are simply linear, bottom-up and bottom-down. Both may lead to an over-estimated result. Secondly, authors do not

Table 4.1: A sample dataset of the original DG discovery approach

ID	Feedback
1	A_+, B_+, C_+
2	A_+, C_-, B_+
3	B_-, A_+, C_-
4	B_-, C_-, A_+
5	C_-, A_+, B_+
6	C_-, B_+, A_+

ID	P_{A+}	P_{A-}
1	$\emptyset$	
2	$\emptyset$	
3	$\emptyset$	
4	$\emptyset$	
5	$\emptyset$	
6	$\emptyset$	

(a) Projected datasets for A

ID	P_{B+}	P_{B-}
1	A	
2	A	
3		$\emptyset$
4		$\emptyset$
5	A	
6		$\emptyset$

(b) Projected datasets for B

ID	P_{C+}	P_{C-}
1	$\{A, B\}$	
2		A
3		A
4		$\emptyset$
5		$\emptyset$
6		$\emptyset$

(c) Projected datasets for C

Figure 4.7: Projected datasets of the sample dataset

utilize much of the context information, considering only ratings from learners. Last but not least, they do not mention how to eliminate the noise. For example, they eliminate all the items P_{i-} from P_{i+}, but in real data there will definitely be some noise.

Regardless of these drawbacks, the algorithm is considered innovative. We propose an improved version of it as one of our approaches to recommend courses. We choose this method for the reason that there are just too few recommendation methods utilizing the sequence, especially in the education field. The following aspects are changed to make the approach more suitable to our case.

The first modification is rating. We cannot ask students to rate all the courses they have taken. Besides, these ratings are not very reliable for building dependencies since the rating may come more from a preference mostly instead of a relation. The indicator we built our dependencies upon is the mark obtained by students in courses since mark is what how we evaluate our CRS performance and we have explained this in Chapter 2. A good mark for course i before a good mark of course j often implies course i is the prerequisite

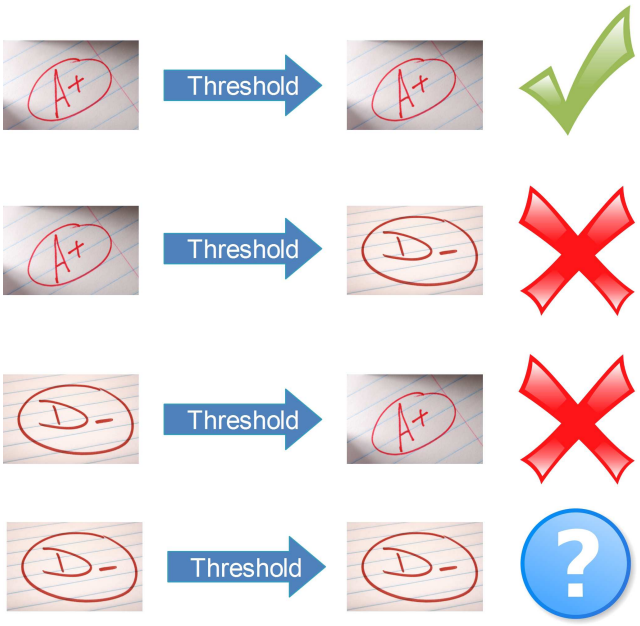


Figure 4.8: The rationale behind our DG algorithm

of course j. This rationale is similar to the approach mentioned above. In addition to the original algorithm rationale, we also add a small variation in P_{i-}. Once we decide to use the mark as a evaluation criteria, the next question will be how good is positive and how bad is negative. Instead of using a universal criteria, a more personalized conversion may be better. For instance, B+ may be a good mark in common sense, but for a successful student whose mark is A on average, B+ is not that good. Meanwhile, for a less successful student whose average mark is only B, B+ is indeed a good mark. This characteristic of students must be considered. Another improvement is support and confidence thresholds. These two terms are used in association rule learning to find interesting relations. The support is an indication of how frequently the itemset appears in the database while the confidence is an indication of how often the rule has been found to be true. One obvious problem of the original algorithm is that the noise cannot be dealt with properly and thresholds can solve it. When we are building the projected dataset a item is added into P_{i+} only if the occurrence number satisfy the support and the confidence threshold we set. After these modifications, the rationale behind our DG algorithm is shown in Figure 4.8.

Algorithm 4 Algorithm of CRS based on DG

Input :

Logs L of finished students course history

Student stu who needs course recommendations

Execute :

1: Convert all marks of courses from L to positive or negative signs. The standard may differ based on GPA
2: Build the projected dataset using Algorithm 3 with new modifications applied.
3: Set candidate courses $CC = \emptyset$
4: Add courses whose prerequisites are finished to CC
5: Rank CC based on selected metrics
6: Recommend the top courses from CC to stu

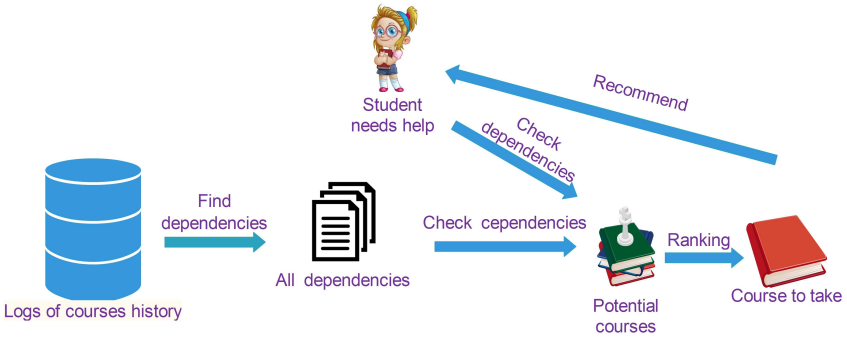


Figure 4.9: The overall workflow of our CRS based on DG algorithm

Based on these rationales, we outline our algorithm of our CRS based on DG in Algorithm 4. The CRS first learns dependencies from the finished students' course history. For a student who needs recommendations, the CRS will check the previous course history of this student and compares his history with the dependencies the CRS has learned. At the end, the CRS summarizes the candidate courses that will be potentially recommended, ranks them, and recommends the final result to this student. The overall workflow of our CRS based on DG algorithm is shown in Figure 4.9.

Chapter 5

Course Recommender System based on Sequential Pattern Mining

Sequential pattern mining is an important data mining problem with broad applications. It focuses mainly on sequence data. In the following section we give a brief introduction of sequential pattern mining including some basic definitions and various approaches. Then we explain how we apply this sequence technique in our CRS.

5.1 Review of Sequential Pattern Mining

Sequential pattern mining (SPM) deals with data represented as sequences (a sequence contains sorted sets of items) [33]. In [4] the authors calls a database a base of data sequences if it has the characteristics: sequence-id, transaction-time and the item involved in the transaction. More precisely, each transaction is a set of items (itemset or element) and each sequence is a list of transactions ordered by transaction time. A sequential pattern is a frequent sequence whose statistical significance in the database is above a user-specified threshold. SPM is finding all the frequent sequences in the database. However, finding all the patterns from huge data sets is a time-consuming task, i.e., the examination of all possible combinations is intractable and new algorithms are needed. It is obvious that SPM is applicable in a wide range since many types of data are in a time-related format. It can be used to develop marketing strategies from

Table 5.1: Data sequences of four customers over four days

Cust	June 04, 2004	June 05, 2004	June 06, 2004	June 07, 2004
C1	Camcorder, MiniDV	Digital Camera	MemCard	USB Key
C2	Camcorder, MiniDV	DVD Rec, DVD-R		Video Soft
C3	DVD Rec, DVD-R	MemCard	Video Soft	USB Key5
C4		Camcorder, MiniDV	Laptop	DVD Rec, DVD-R

a customer shopping database. By doing a web log analysis, patterns found can be useful to better structure a company's website for providing easier access to the most popular links [23]. SPM can also be applied to intrusion detection [21] and DNA sequences [65]. A good SPM algorithm should possess the following features. First, it should be able to find the complete set of patterns when possible, satisfying the minimum support threshold. Second, it should be highly efficient, scalable, involving only a small number of database scans. Lastly, it should be able to incorporate various kinds of user-specific constraints.

Some definitions need to be cleared first to help us understand the various problems and methods presented hereafter. We illustrate these definitions by explaining an example from [33] shown in Table 5.1.

An item can be considered as the object bought by a customer, or the page requested by the user of a website, etc. An itemset is the set of items that are grouped by timestamp. For example, all the pages requested by a user on June 04, 2004. A data sequence is a sequence of itemsets associated with a customer. In Table 5.1, the data sequence of C2 is the following: "(Camcorder, MiniDV) (DVD Rec, DVD-R) (Video Soft) which means that the customer bought a camcorder and miniDV the same day, followed by a DVD recorder and DVD-R the day after, and finally a video software a few days later. A sequential pattern is included in a data sequence. For instance "(MiniDV) (Video Soft)" is included in the data sequence of C2, whereas "(DVD Rec) (Camcorder)" is not included according to the order of the timestamps). The minimum support is specified by the user and stands for the minimum number of occurrences of a sequential pattern to be considered as frequent. A maximal frequent sequential pattern is included in at least "minimum support" data sequences and is not included in any other frequent sequential pattern. Table 5.1 gives

an example of four customers and their activities over four days in a shop. With a minimum support of "50% " a sequential pattern can be considered as frequent if it occurs at least in the data sequences of two customers. In this case a maximal SPM process will find three patterns:

- S1: (Camcorder, MiniDV) (DVD Rec, DVD-R)
- S2: (DVD Rec, DVD-R) (Video Soft)
- S3: (Memory Card) (USB Key)

One can observe that S1 is included in the data sequences of C2 and C4, S2 is included in those of C2 and C3, and S3 in those of C1 and C2. Moreover, the sequences do not have the same length, i.e., S1 has length 4, S2 has length 3 and S3 has a length of 2.

These basic definitions are essential for understanding SPM algorithms. Generally speaking, there are two types of sequential mining approaches. One is Apriori-based and the other one is pattern projection based.

The intuition of an Apriori-based approach is a simple fact that if a sequence S is not frequent, then none of the super-sequences of S is frequent. If $< ab >$ is infrequent, so is $< (ac)b >$. This approach is first stated in [4] with a SPM concept introduction and an initial Apriori-like algorithm. Then the problem and approach are improved in [47]. In [47], the GSP algorithm is based on a breadth-first principle since it is an extension of the Apriori model to the sequential aspect of the data. GSP uses the "Generating-Pruning" method defined in [3] and performs in the following way. Initially, every item in database is a candidate of length 1. Then for each level, i.e., sequences of length k), scan database to collect support count for each candidate sequence. After that, generate candidate length (k+1) sequences from length k frequent sequences using Apriori. Finally, repeat the above process until neither frequent sequence nor candidate can be found. One thing worth mentioning is candidate generation. A candidate sequence of length (k+1) is generated from two frequent sequences, $s1$ and $s2$, both having length k, if the subsequence

obtained by pruning the first item of s1 is the same as the subsequence obtained by pruning the last item of s2. With the example in Table 5.1, and k=2, let s1 be "(DVD Rec, DVD-R)" and s2 be "(DVDR) (Video Soft)", then the candidate sequence will be "(DVD Rec, DVD-R) (Video Soft)" since the subsequence described above (common to $s1$ and $s2$) is "(DVDR)".

Another method based on Apriori is SPADE which stands for Sequential PAttern Discovery using Equivalence classes [64]. The main idea in this method is a clustering of the frequent sequences based on their common prefixes and the enumeration of the candidate sequences. Thanks to a rewriting of the database (loaded in main memory), SPADE needs only three database scans in order to extract the sequential patterns.

Although Apriori based methods are the first solution to SPM problems, several drawbacks hampered further use of this naive approach. First, it generates a huge set of candidates, especially 2-item candidate sequence. Second, it requires multiple scans of database in mining since the length of each candidate grows by only one at each database scan. Third, it is inefficient for mining long sequential patterns. It is almost unfeasible to discover a long pattern from an exponential number of short candidates from which it grows.

A more efficient and common approach to solve SPM problem in practice is pattern projection based. An original approach called FreeSpan is proposed in [20] for mining sequential patterns by recursively projecting the data sequences into smaller databases. It is the first algorithm considering the pattern projection method for mining sequential patterns. This work has been continued with PrefixSpan which stands for Prefix-projected SPM, [38], based on a study about the number of candidates proposed by a Generating-Pruning method. Starting from the frequent items of the database, PrefixSpan generates projected databases with the remaining data-sequences. The projected databases thus contain suffixes of the data sequences from the original database, grouped by prefixes. The process is recursively repeated until no frequent item is found in the projected database. At this level the frequent sequential pattern is the path of frequent items driving to this projected database. The merit of this method compared with Apriori approach is mainly efficiency. For PrefixSpan,

no candidate sequence needs to be generated and the projected databases keep shrinking. The major cost is constructing projected databases which can be enhanced in several ways. More details of this method are presented in the next section and this is the one we adopt in our CRS.

5.2 Implementation of Course Recommender System based on Sequential Pattern Mining

In this section we discuss the concrete method of making course recommendations based on SPM. As mentioned in the last section, the most widely used SPM algorithm in practice is PrefixSpan [38] because of its efficiency. The general idea of its predecessor algorithm FreeSpan [20] is to use frequent items to recursively project sequence databases into a set of smaller projected databases and grow subsequence fragments in each projected database. The idea of the improved version PrefixSpan is to examine only the prefix subsequences and project only their corresponding postfix subsequences into Projected databases.

For the following part, we will try to formally define some concepts of SPM and then we will unfold the algorithm of PrefixSpan [38].

Let $I = \{ i_1, i_2, ..., i_n \}$ be a set of all items. An itemset is a subset of items. A sequence is an ordered list of itemsets. A sequence s is denoted by $\langle s_1, s_2, ..., s_l \rangle$, where s_j is an itemset, i.e., $s_j \subseteq I$ for $1 \leq j \leq l$. s_j is also called an element of the sequence, and denoted as $(x_1 x_2 ... x_m)$, where x_k is an item, i.e., $x_k \in I$ for $1 \leq k \leq m$. For brevity, the brackets are omitted if an element has only one item. That is, element (x) is written as x. An item can occur at most once in an element of a sequence, but can occur multiple times in different elements of a sequence. The number of instances of items in a sequence is called the length of the sequence. A sequence with length l is called an l-sequence. A sequence $\alpha = \langle a_1, a_2, ..., a_n \rangle$ is called a subsequence of another sequence $\beta = \langle b_1, b_2, ..., b_m \rangle$ and β a super sequence of α denoted as $\alpha \sqsubseteq \beta$ if there exist integers $1 \leq j_1 < j_2 < ... < j_n \leq m$ such that

Table 5.2: An example sequence database

Sequence_id	Sequence
10	$\langle a(abc)(ac)d(cf)\rangle$
20	$\langle (ad)c(bc)(ae)\rangle$
30	$\langle (ef)(ab)(df)cb\rangle$
40	$\langle eg(af)cbc\rangle$

$a_1 \subseteq b_{j1}, a_2 \subseteq b_{j2}, ..., a_n \subseteq b_{jn}$.

A sequence database S is a set of tuples $\langle sid, s\rangle$, where sid is a sequence_id and s is a sequence. A tuple $\langle sid, s\rangle$ is said to contain a sequence α, if α is a subsequence of s, i.e., $\alpha \sqsubseteq s$. The support of a sequence α in a database S is the number of tuples in the database containing α, i.e. $support_S(\alpha) = |\{\langle sid, s\rangle | (\langle sid, s\rangle \in S) \wedge (\alpha \sqsubseteq s)\}|$. It can be denoted as $support(\alpha)$ if the sequence database is clear from the context. Given a positive integer ξ as the support threshold, a sequence α is called a (frequent) sequential pattern in sequence database S if the sequence is contained by at least ξ tuples in the database, i.e., $support_S(\alpha) \geq \xi$. A sequential pattern with length l is called an l-pattern.

Let us see an example of database S shown in Table 5.2 and set min_support = 2. The set of items in the database is $\{a, b, c, d, e, f, g\}$.

A sequence $\langle a(abc)(ac)d(cf)\rangle$ has five elements: $(a), (abc), (ac), (d) and (cf)$, where items a and c appear more than once respectively in different elements. It is also a 9-sequence since there are 9 instances appearing in that sequence. Item a happens three times in this sequence, so it contributes 3 to the length of the sequence. However, the whole sequence $\langle a(abc)(ac)d(cf)\rangle$ contributes only one to the support of $\langle a\rangle$. Also, sequence $\langle a(bc)df\rangle$ is a subsequence of $\langle a(abc)(ac)d(cf)\rangle$. Since both sequences 10 and 30 contain subsequence s = $\langle (ab)c\rangle$, s is a sequential pattern of length 3 (i.e., 3-pattern). Given S and min_support = 2, the support of each item is listed below in support descending order (in the form of item:support). They are all frequent items since all supports are greater than two.

$$\text{f\_list} = a:4, b:4, c:4, d:3, e:3, f:3$$

Above is the formal definitions of basic concepts of a sequence, we now

dive deeper into PrefixSpan. Since items within an element of a sequence can be listed in any order, we assume they are listed in alphabetical order. For example, the sequence in S with sequence_id 10 in Table 5.2 is listed as $\langle a(abc)(ac)d(cf)\rangle$ instead of $\langle a(bac)(ca)d(fc)\rangle$. With such a convention, the expression of a sequence is unique.

Given a sequence $\alpha = \langle e_1 e_2 ... e_n\rangle$, a sequence $\beta = \langle e_1^{'} e_2^{'} ... e_m^{'}\rangle (m \leq n)$ is called a prefix of α if and only if $(1) e_i^{'} = e_i$ for $i \leq m-1$; $(2) e_m^{'} \subseteq e_m$; and (3) all the items in $e_m - e_m^{'}$ are alphabetically after those in $e_m^{'}$.

Given sequences α and β such that β is a subsequence of α, i.e., $\beta \sqsubseteq \alpha$. A subsequence $\alpha^{'}$ of sequence α (i.e., $\alpha^{'} \sqsubseteq \alpha$) is called a projection of α with regard to prefix β if and only if (1) $\alpha^{'}$ has prefix β and (2) there exists no proper super-sequence $\alpha^{''}$ of $\alpha^{'}$ (i.e., $\alpha^{'} \sqsubseteq \alpha^{''}$ but $\alpha^{'} \neq \alpha^{''}$) such that $\alpha^{''}$ is a subsequence of α and also has prefix β.

Let $\alpha^{'} = \langle e_1 e_2 ... e_n\rangle$ be the projection of α with regard to prefix $\beta = \langle e_1 e_2 ... e_{m-1} e_m^{'}\rangle$ $(m \leq n)$. Sequence $\gamma = \langle e_m^{''} e_{m+1} ... e_n\rangle$ is called the postfix of α with regard to prefix β, denoted as $\gamma = \alpha/\beta$, where $e_m^{''} = (e_m - e_m^{'})$. We also denote $\alpha = \beta \cdot \gamma$. If β is not a subsequence of α, both projection and postfix of α with regard to β are empty.

For example, $\langle a\rangle, \langle aa\rangle, \langle a(ab)\rangle$ and $\langle a(abc)\rangle$ are some prefixes of sequence $\langle a(abc)(ac)d(cf)\rangle$, but neither $\langle ab\rangle$ nor $\langle a(bc)\rangle$ is considered as a prefix. In addition, $\langle (abc)(ac)d(cf)\rangle$ is the postfix of the same sequence with respect to prefix $\langle a\rangle$, $\langle (\_bc)(ac)d(cf)\rangle$ is the postfix with respect to prefix $\langle aa\rangle$, and $(\_c)(ac)d(cf)$ is the postfix with respect to prefix $\langle a(ab)\rangle$.

For the same sequence database S in Table 5.2 with $min_{sup} = 2$, sequential patterns in S can be mined by a prefix-projections method in the following steps.

Step 1: Find length-1 sequential patterns. Scan S once to find all frequent items in sequences. Each of these frequent items is a length-1 sequential pattern. They are $\langle a\rangle : 4$, $\langle b\rangle : 4$, $\langle c\rangle : 4$, $\langle d\rangle : 3$, $\langle e\rangle : 3$, and $\langle f\rangle : 3$, where $\langle pattern\rangle : count$ represents the pattern and its associated support count.

Step 2: Divide search space. The complete set of sequential patterns

can be partitioned into the following six subsets according to the six prefixes: (1) the ones having prefix $\langle a \rangle$; ...; and (6) the ones having prefix $\langle f \rangle$.

Step 3: Find subsets of sequential patterns. The subsets of sequential patterns can be mined by constructing corresponding projected databases and mine each recursively.

For the clarity of the algorithm the mining process is explained as the following part.

First, let us find sequential patterns having prefix $\langle a \rangle$. Only the sequences containing $\langle a \rangle$ should be collected. Moreover, in a sequence containing $\langle a \rangle$, only the subsequence prefixed with the first occurrence of $\langle a \rangle$ should be considered. For example, in sequence $\langle (ef)(ab)(df)cb \rangle$, only the subsequence $\langle (\_b)(df)cb \rangle$ should be considered for mining sequential patterns having prefix $\langle a \rangle$. Notice that $\langle (\_b) \rangle$ means that the last element in the prefix, which is a, together with b, form one element. As another example, only the subsequence $\langle (abc)(ac)d(cf) \rangle$ of sequence $\langle a(abc)(ac)d(cf) \rangle$ should be considered. Sequences in S containing $\langle a \rangle$ are projected with respect to $\langle a \rangle$ to form the $\langle a \rangle$-projected database, which consists of four postfix sequences: $\langle (abc)(ac)d(cf) \rangle$, $\langle (\_d)c(bc)(ac) \rangle$, $\langle (\_b)(df)cb \rangle$ and $\langle (\_f)cbc \rangle$. By scanning $\langle a \rangle$-projected database once, all the length-2 sequential patterns having prefix $\langle a \rangle$ can be found. They are: $\langle aa \rangle : 2, \langle ab \rangle : 4, \langle (ab) \rangle : 2, \langle ac \rangle : 4, \langle ad \rangle : 2$, and $\langle af \rangle : 2$. Recursively, we continue to construct projected databases and to find sequential patterns with $\langle aa \rangle$, $\langle ab \rangle$, $\langle (ab) \rangle$, $\langle ac \rangle$, $\langle ad \rangle$ and $\langle af \rangle$. Similarly, we can find sequential patterns having prefix $\langle b \rangle$, $\langle c \rangle$, $\langle d \rangle$, $\langle e \rangle$ and $\langle f \rangle$, respectively.

The formal description of PrefixSpan algorithm is described in Algorithm 5.

With the explanation of PrefixSpan algorithm, we now connect it to our CRS. Students take a few courses each term. There is no order of courses in a specific term, yet the courses of different terms do follow a chronological order. It is very similar to the shopping example. A student is like a customer while courses are like shopping items. Students take several courses in one term like customers buying several products in one order. Students take courses in different terms like customers buying products in different days. Just like we

Algorithm 5 Algorithm of PrefixSpan

Input :

A sequence database S, and the minimum support threshold min_sup
Students set S who need recommendations

Parameters :

α: A sequential pattern;
l: The length of α;
$S\ |_\alpha$: The α-projected database, if $\alpha \neq \langle\rangle$; otherwise, the sequence database S which is the case when first call this algorithm.

1: Scan $S\ |_\alpha$ once, find the set of frequent items b such that (1) b can be assembled to the last element of α to form a sequential pattern; or (2) $\langle b\rangle$ can be appended to α to form a sequential pattern.
2: For each frequent item b, append it to b to form a sequential pattern $\alpha^{'}$, and output $\alpha^{'}$;
3: For each $\alpha^{'}$, construct $\alpha^{'}$-projected database $S\ |_\alpha$, and call PrefixSpan($\alpha^{'}$, $l+1$, $S\ |_{\alpha'}$).

can find a frequent sequence patterns of items bought by customers, frequent sequence patterns of courses taken by students can also be found.

Our CRS based on SPM works in the following way. We first filter all the course records and only the course record whose mark is A or A+ can be left. Note that a course deleted in one sequence may be selected in another sequence. For instance, a student who took CMPUT 101 and got an A on it then this course will be left in this student's sequence. If another student who also took CMPUT 101 but got a B on it then this course will be filtered out. After this step, the course records left in students history will all be A or A+. We take this step because we only want to find the positive sequential patterns of courses, i.e., sequences of courses taken by students with good outcome. A+ and A are taken as reference examples.

The second step is to treat these courses like the shopping items and process them with our SPM technique (PrefixSpan) to find all the sequential patterns of courses. Ideally, we want to recommend courses from the most significant patterns. The course sequential patterns we find, some are long while some are short. Suppose, one short frequent pattern s_1 we find is $\langle 174, 206\rangle$ while another long frequent pattern s_2 we find is $\langle 174, 175, 204, 304\rangle$. Apparently, the longer pattern s_2 is harder to find over the shorter one s_1. We can asso-

Algorithm 6 Algorithm of CRS based on SPM

Input :

Logs L of finished students course history

Student stu who needs course recommendations

Execute :

1: Filter all the course records of L with a predefined course mark standard as FL
2: Find all the course sequential patterns SP from FL with Algorithm 5.
3: **for all** Sequential pattern p from SP **do**
4: Compute the number of elements num of this sequential pattern that is also contained in stu's course history
5: Add the next course of this p to the Hashtable HT where the key is num
6: **end for**
7: Rank courses from HT's highest key as candidate courses CC based on selected metrics
8: Recommend the top courses from CC to stu

ciate it with a real scenario. Suppose, we have a student who needs course recommendations and he has already taken 174, 175, and 204. 206 can be an option for recommendation since it satisfies s_1. However, a more intuitive recommendation should be 304 because he has already finished 3 courses in s_2. The rareness of a long sequence pattern indicates a strong connection among those courses if it ever passes the criteria we set for a sequential pattern. Based on this intuition, the courses we recommend are the next unfinished elements from the sequential patterns that have the longest common elements with our student's current course history. By this algorithm, the course we recommend for our example student earlier will be course 304 since the length of common elements of s_2 and this student is three, longer than one which is of s_1.

A formal presentation of our CRS algorithm based on SPM is shown in Algorithm 6. The overall workflow of our CRS based on SPM algorithm is shown in Figure 5.1. We first filter out all low mark courses. Then we run Algorithm 5 to find all the sequential patterns. We add the next courses from the longest matched sequential patterns as candidate courses that could be recommended. We rank these candidate courses. Finally, we formally recommend the top courses to the student.

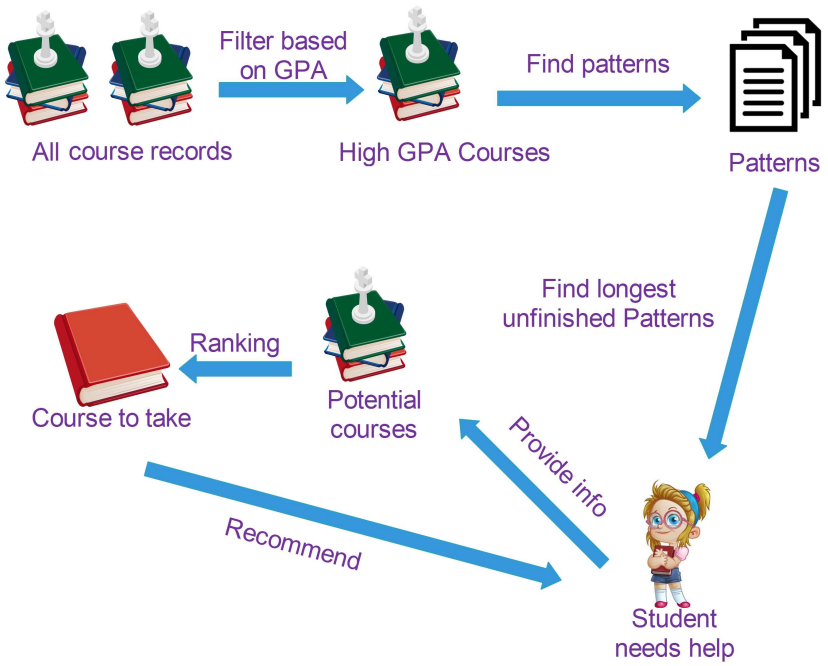


Figure 5.1: The overall workflow of our CRS based on SPM algorithm

It is rather obvious that the second DG algorithm is different from the other two, yet the difference between the first PM approach and the third SPM approach may not be quite clear since they both aim at "successful" students. One can consider that we extract many full paths of successful students in the PM approach, while in the SPM approach we only extract the successful sequence segments. Also, the the word "successful" in PM approach focuses on students. Such a successful student can have bad courses in his or her path. In contrast, the word "successful" in SPM focuses on courses. There is no low mark course in a frequent sequence we find.

Chapter 6

Course Recommender System Miscellany

The main topic of this research is to help students choose courses to graduate. This "help" can be in different ways. All methods previously mentioned focus more on student's course performance, while this part we concentrate on students who would like to graduate as soon as possible. An intuitive approach to achieve that is developed.

It typically falls into two categories to speed up a student's graduation. One is the compulsory requirements from school or department. This type of information can be obtained from a department's guide documents or websites and they can be easily input to our CRS. The other one is the optional part, which are courses not explicitly required by the department but have a crucial role in a student's graduation. These optional courses can be further considered in two aspects. First, these courses may be very important that many students decide to take them even though they are not in the mandatory list. For instance, suppose course CMPUT 206 is not indicated by the department guideline as a compulsory course, yet above 90% students choose to take it. Clearly, CMPUT 206 is in a significant status as much as compulsory courses that are required in the guideline. We can compute the percentage of students who take a specific course and rank courses based on this percentage from high to low. It could be a must for students who want to graduate as soon as possible if the percentage of students who take this course is above a certain threshold like 90%. The second aspect to distinguish courses that can speed

up graduation is doing a statistics calculation. For example, for one course, we can compute the average time needed to graduate by students who take this specific course. We do this for all the courses and rank them based on the average graduation time from low to high, the lower the number the faster a student graduates. Apparently, if a course is on top of the list, it is likely to contribute to the speed of a student's graduation.

In short, there are three attributes we consider when we decide how useful a course is to speed up a student's graduation. First is whether it is mandatory from the department's guideline. Second is the percentage of students who take this course. Third is the average time before graduation by students who take this course. The second category can actually be merged into the first category since they both indicate how crucial a course is, either by the department or the choice of students. We combine the courses that are chosen by more than 90% (this threshold can be changed) of students with the compulsory courses specified by educators as one group and we call this group key courses.

This "speedup" system is not used alone in our CRS, but is used to rank the potential recommended courses selected by our three sequence-based algorithms. This ranking process is always the last step of these three sequential based algorithms, which can be found in the algorithms in previous chapters. To be more exact, after selecting a few courses in the potential course list by one of the three sequence-based approaches, there are three methods to rank them with this "speedup" algorithm.

1. No "speedup": Rank courses merely on the GPA contribution of courses

2. Semi "speedup": Always rank key courses that are in the potential course list first. The key course list and the non-key course list will be ranked based on each course's GPA contribution respectively.

3. Full "speedup": Always rank key courses that are in the potential course list first. The key course list and the non-key course list will be ranked based on each course's average graduation time by students who take this course.

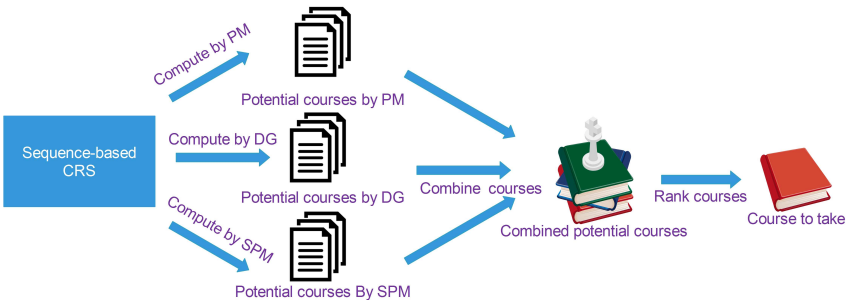


Figure 6.1: The overall workflow of our CRS that combines all three sequence-based algorithms

Even though the performance of students (GPA) is not considered in the third ranking approach, it will not be a big problem. Because the potential recommending course list is already a result returned by our sequence-based CRS.

We try to combine all of our three sequence-based methods. Since each of them produces a potential list of recommended courses, it is natural to combine the result of potential courses of all three methods and rank the result based on the ranking system we just mentioned. We provide this option in our CRS as "Comprehensive". The overall structure of this approach is shown in Figure 6.1.

Some special situations are into considerations such as new courses. Generally speaking, new established courses would not be in any recommendations since no student has taken them. Nevertheless, we do take it as an option and add it to the potential recommended courses randomly in order to let more students take it.

Chapter 7

Experiments

After introducing all three methods, here we evaluate these methods and compare them to assess which one outperforms the others. Due to the ethical approval problem, namely obtaining consent from past students, we are unable to access the real students data. We first discuss the data simulator we built to mimic the real undergraduate students behaviour records. Then we evaluate the performance of our CRS. Finally, we make a simple Graphic User Interface (GUI) to help students use our CRS.

7.1 Data Simulator

The Computing Science Department of the University of Alberta collects for each semester and for each student the courses they register in and the final mark they obtain. While there are prerequisites for courses and other strict constraints, the rules are not enforced and are thus often violated. This history for many years, constituting the needed event log, is readily available. However, such data cannot be used for research purposes or for publication even though anonymized due to lack of ethical approval. It is unpractical to gather the consent of all past students, which leaves us two options. We can either obtain written consent from current students and start collecting data for some years, or simulate historic curriculum data for proof of concept and publication, and use real data for local implementation. Due to the time limit, we opted for the simulation of the event log.

We write a curriculum simulator to mimic the behaviours of undergraduate

students in higher education with different characters from the department and be as close as possible to real data. To mimic the reality, two positions need to be considered. One is from the educator's, like what are the requirements from the department, from the educators. These requirements are mostly rules and they are mandatory and fixed. Another perspective is from the student's, like how students will behave when they study. These behaviours tend to be volatile and various from students to students. Our simulator takes both of them into consideration in order to simulate as real data as possible. The following is what we implement from the department's or the educator's angle. Firstly, courses are the cornerstone of the whole system. The courses we choose are real courses from the course directory of the Computing Science Department the University of Alberta [1]. The course information we extract includes, which is not limited to, course number, course name, the instructor, when the course is open etc. Not only the course information is based on what we have in reality in our Computing Science Department, but also most of the following constrains and variables settings in our simulator are based on the reality of our department. With these basic course information, we consider course relations. We predefine a set of rules or requirements that are according to those in the offered programs in the department. One constraint is prerequisites, i.e., some specified courses must be taken before others. For example, in our department CMPUT 204 must be taken if a student wants to take CMPUT 304. Other requirements include the first and the last course a student must take, mandatory courses, i.e., courses that students must take before they graduate, and non-coexisting courses, i.e., if students take one course in the group then they cannot take any other course belonging to the same group.

On the other side of the simulator is the student behaviour. As mentioned, this behaviour varies in great range from students to students. Although students may have different personalities, we only consider characters that are related to our task and they can be divided into three attributes. The first attribute is student's performance. This attribute influences how a student performs in a specific course, which is the mark obtained in the course. Per-

forming students tend to have a higher mark in their course in average. The second attribute is diligence, which has an impact on how many courses a student tends to take in each term. We differentiate students by the number of credits they take by semester, thus the time for completion. Some have a balanced and uniform distribution throughout, while others are irregular with regard to the number of courses per semester. The third attribute is responsibility. This affects the sequence of courses a student takes. There are three types of students: the responsible students who always satisfy the course constraints; the typical students who seldom violate course constraint rules; and the careless students who often do not follow the set rules like prerequisites. These attributes are not fixed just like students characters are not constant. We give students some probability to alter their characters. For example, students who were diligent that take a lot of courses at first may slow down when they go to their third year of college. We also add some probability for students to withdraw from a course given the course load and previous withdrawing behaviour.

Algorithm 7 Algorithm of data generation of taking courses by students

Input :
The total number of terms $numTerm$ that need to be generated
The number of students enrolled in each term $numNew$

Execute :
1: $ongoingList = \emptyset$
2: $finishedList = \emptyset$
3: $curTerm = 0$
4: **for** $curTerm < numTerm$ **do**
5: Add $numNew$ students to $ongolingList$
6: **for all** Student in $ongolingList$ **do**
7: Add courses for this term with Algorithm 8
8: **if** *student* satisfies graduation criteria **then**
9: Move *student* from $ongoingList$ to $finishedList$
10: **end if**
11: **end for**
12: **end for**

After we modelled fixed requirements from department and various characters of students, generating data of students taking courses is the final task

Algorithm 8 Algorithm of adding courses for one student at a specific term

Input :

The current term $curTerm$ such as Fall, Winter or Summer

The student stu who is going to take courses this term

Execute :

1: Set candidate courses $candCourses = \emptyset$
2: Add preferred courses to $candCourses$ based on $curTerm$ and the responsibility attribute of stu
3: Add several courses that do not have restrictions to $candCourses$
4: Add new established courses to $candCourses$ if applicable
5: Choose the courses that are going to be taken $coursesTaking$ randomly from $candCourses$ based on the diligence and the responsibility attributes of stu
6: **for all** $courseTaking$ in $coursesTaking$ **do**
7: Generate a mark for $courseTaking$ based on the performance attribute of stu
8: Add $courseTaking$ to the course history of stu
9: **end for**

for our simulator. Initially, we consider to generate data based by per student, i.e., we create the whole course record history of one student and then move on to the next student. However, this generation approach is not what happens in reality. Reality is time based and that is what we opt for in the end. We make one term or semester as a basic time unit. Each term, with time goes by, there are new students arriving, there are current students continuing their study and there are some students getting graduated as well. Therefore, there are two lists of students, one is for finished students who are already graduated and the other one is for ongoing students who are still taking courses. For each student in the ongoing list, we choose some courses for them to take in this term. After the term, we check whether the student has satisfied the criteria to graduate. If the criteria is met, we move this student from ongoing list to the finished list. The formal description of such algorithm is shown in Algorithm 7.

Several things need to be clarified in the algorithms above. First of all, the number of students enrolled in each term is not fixed. Apparently, most students enroll in the fall and winter term and new students rarely come in the summer. Second, the process of adding courses for students each term is very

important so we make Algorithm 8 to illustrate that. The course selection process is to some extent random yet they are based on the requirements from the department and students' attributes we mentioned before. We also set some hidden preferred courses for each term. These are the courses that students would take in a normal situation at a specific time. Students with higher responsibility attribute are more likely to choose these preferred courses. We set these preferred courses so as to decrease the randomness during course selection, otherwise the graph will be too messy and further from reality. As to the mark generation part, two aspects are considered. One is the performance attribute of students, i.e., the higher this attribute, the more likely students will get a high mark. Another aspect is course prerequisite relations. Student will have a higher probability to have a low mark on a course if they did not take its prerequisite courses or performed badly on prerequisite courses. Lastly, the criteria of graduation are from the real planner of the department. Students need to take enough course credits and meet some other constraints for special courses in different areas in order to graduate.

7.2 Course Recommender System Tuning

Before we analyze the effect of our algorithms, there are some important parameters that need to be tuned for all three methods. For tuning, all three methods are evaluated in a similar environment. We set the size of finished students dataset, which our CRS learns from, at 1500. The size of students who need recommendations is set to 200. For these 200 students their choices of courses in each term are decided by our CRS. We tune key parameters for each of the three sequence-base methods separately. For most of the experiments, we study how changes in parameters affect the average GPA of students who adopt our recommendations. For the baseline of analyzing our results, the average GPA is 3.446 if 200 students do not take any recommendations from us. As to the grading system and grade points conversion, A correspond to 4.0, A- to 3.7, B+ to 3.3 and B to 3.0. The numbers presented in each table and figure for this section are the average scores of their corresponding

Table 7.1: 200 students' average GPA varied by different thresholds of GPA and similarity in CRS based on PM

Similarity	GPA 3.4	GPA 3.6	GPA 3.8
0.3	3.431	3.443	3.536
0.5	3.453	3.510	3.574
0.7	3.484	3.549	**3.592**
0.8	3.482	3.551	3.589

experiment three times. In addition, the best threshold number may not be the exact number that maximizes our CRS. For example, if our CRS performs better when a certain threshold is 0.437 compared to 0.4, we may simply use 0.4 or 0.45 instead of 0.437. Because this threshold is really subject to different datasets, and we do not intend to make a threshold exact for a particular dataset.

Starting with our CRS based on PM, two important parameters are how we define successful students, which is the GPA threshold, and how we define two students are similar, which is the similarity threshold. For example, if we set the GPA threshold to 3.6 and the similarity threshold to 0.7, it means that only students whose GPA is above 3.6 are considered successful and only students who have a similarity above 0.7 are considered similar. Table 7.1 shows 200 students average GPA varied by different thresholds of GPA and similarity in CRS based on PM. From Table 7.1, we can see that, first, when we define successful students as those whose GPA is above 3.4, the recommendations do not work very well: average GPAs are even below the baseline. This is reasonable since GPA 3.4 is our baseline and it is not a high standard. Some improvements can be seen when we increase the GPA threshold a bit and these improvements are obvious when we set the GPA threshold to 3.8. Meanwhile, the similarity threshold has a positive relation with students performance which is logical, except for the case of the GPA threshold 3.4. Nonetheless, this positive relation no longer exists when the similarity threshold increases beyond 0.7.

Important parameters for our CRS based on DG is the support and confidence thresholds for selecting P_{i+} and P_{i-}. Support is an indication of how frequently the itemset appears in the database, while confidence is an indica-

tion of how often the rule has been found to be true. Their formal definitions are as following.

- The support of X with respect to T is defined as the proportion of transactions t in the database which contains itemset X.

  $supp(X) = \frac{\{|t \in T; X \subseteq T|\}}{|T|}$

- The confidence value of a rule, $X \Rightarrow Y$, with respect to a set of transactions T, is the proportion of the transactions that contains X which also contains Y.

  $conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)}$

One of the improvements we made to the original DG algorithm is a set of thresholds, which are support and confidence to deal with the noise. We focus on tuning the confidence values mostly and set the support threshold to 0.05 due to two reasons. Firstly, considering there are situations that some dependencies do not appear very frequently because the courses are rarely taken, it is unjustifiable to set a high support value. Secondly, the confidence value is a stronger indicator of the dependency compared to the support value.

The parameters we tune are P_{i+} confidence threshold and P_{i-} confidence threshold. On one hand, the higher the Pi+ confidence threshold is, the less likely we will accept a dependency for a course. On the other hand, the higher the Pi- confidence threshold is, the more likely we will accept a dependency for a course. Table 7.2 shows 200 students average GPA varied by different P_{i+} and P_{i-} confidence thresholds in CRS based on DG. There is not a clear rule that can be concluded from Table 7.2, but it seems that students do not perform well when we P_{i+} confidence is relatively lower than P_{i-} confidence. One possible explanation is that, if we are open to consider many candidate dependencies but do not filter them, many false dependencies will be found. In that case it will reduce the reliability of our CRS.

For our CRS based on SPM, the only parameter we need to tune is the support threshold. The higher the value, the stricter our requirement is for a sequential pattern. From Table 7.3 We find that the average GPA is higher

Table 7.2: 200 students' average GPA varied by different P_{i+} and P_{i-} confidence thresholds in CRS based on DG

	P_{i+} Conf 0.25	P_{i+} Conf 0.5	P_{i+} Conf 0.75
P_{i-} Conf 0.25	3.567	**3.643**	3.563
P_{i-} Conf 0.5	3.484	3.571	3.576
P_{i-} Conf 0.75	3.438	3.533	3.632

Table 7.3: 200 students' average GPA varied by different support thresholds in CRS based on SPM

Support	Average GPA
0.1	3.457
0.3	**3.602**
0.5	3.548

when the support threshold is 0.3. However, the average GPA decreases when the support threshold is set to 0.5. The reason may be that, when the threshold is set higher, many true sequential patterns that do not appear very often are filtered out, which may lead to a bad result. After all, requiring a sequential pattern to appear at least in half of the total sequences is too demanding.

7.3 Result Analysis

In this section we compare the performance of our CRS based on different sequence-based algorithms. We want to see which sequence-based algorithm performs better, whether the "speedup" algorithm works, what additional insights our CRS can provide. Moreover, we add one more approach to all experiments, which is called "comprehensive" that combines all results from three methods and it is introduced in Chapter 6. If not otherwise specified, the parameters of each algorithm are the ones that performed best in last section. The numbers presented in each table and figure for this section are the average scores of their corresponding experiment three times.

The first experiment is to compare the performance of different sequence-based approaches at different stages of students. "Different stages" means when do students use our CRS. For example, "Year 4" means students only begin to take courses recommended by our CRS in the fourth year, while "Year 1" means students start using our CRS from the first year. Table 7.4 with its

Table 7.4: 200 students' average GPAs varied by the year of starting CRS in different approaches

Approach	Year 4	Year 3	Year 2	Year 1
PM	3.453	3.516	3.569	3.588
DG	3.433	3.529	3.617	3.652
SPM	3.447	3.498	3.545	3.602
Comprehensive	3.441	3.512	3.564	3.593

corresponding Figure 7.1 shows the result of this experiment: 200 students' average GPAs varied by the year of starting CRS in different approaches. The blue line in the middle is our baseline 3.446 which is the average GPA if students do not take any recommendations. From Table 7.4 and Figure 7.1 we can observe the following. Firstly, we can see a substantial effect for students who use our CRS in the first two years. This steady increase indicates students can benefit more if they start using our CRS earlier in their study. Secondly, the performance of CRS for all methods is about the same with the baseline if students only start to use our CRS in the fourth year, which means it may be too late to improve a student's GPA even with the help of CRS. Other than Year 4, our CRS does have a positive impact. Thirdly, CRS based on DG outperforms all in nearly all scenarios while other approaches are equally matched. Note that the comprehensive approach does not outperform others. Our guess is that by combining the candidate courses from all three methods, it obtains too many candidates and cannot perform well if the candidates are not ranked properly. As to why CRS based on DG performs best, it may due to the intrinsic attribute of our data simulator. The mark generation part of our simulator considers course prerequisites, which may favour the DG algorithm. Thus, other approaches may outweigh DG if we are dealing with real data.

The next experiment is to check whether increasing the training data of students will lead to a better performance of our CRS. Table 7.5 and Figure 7.2 demonstrate 200 students' average GPAs varied by the number of training students of CRS in different approaches. We can see that, as the training data size increases from 500 to 1000, the performance of our CRS improves. However, when this size further increases from 1000 to 1500, the performance of our CRS does not have a significant change. This is the reason we use 1500

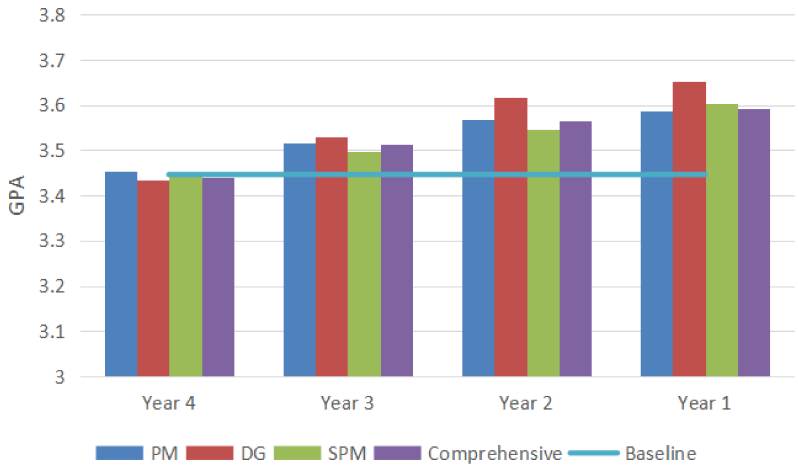


Figure 7.1: 200 students' average GPAs varied by the year of starting CRS in different approaches

Table 7.5: 200 students' average GPAs varied by the number of training students of CRS in different approaches

Approach	500	1000	1500
PM	3.513	3.57	3.586
DG	3.535	3.607	3.639
SPM	3.528	3.581	3.598
Comprehensive	3.522	3.582	3.597

as the training data size in most of the experiments of previous and current sections because it is enough for our CRS to learn the needed dependencies or patterns.

Besides improving students' performance in grades, our CRS can also speed up students' graduation process by ranking the candidate courses selected by sequence algorithms properly. Table 7.6 and Figure 7.3 show the effect of using the full "speedup" ranking setting to recommend courses based on DG to 200 students. Same as the first experiment in this section, Year X means students start to use our CRS from year X. We can see a remarkable decrease of terms that needed to graduate if students start using our CRS from the third year. However, after that, such change is not very notable. Since the pivot to graduate fast is to take all key courses as soon as possible, our explanation is that taking key courses from the third year is timely. There is no particular need to focus on key courses in the first two years. Note that although the

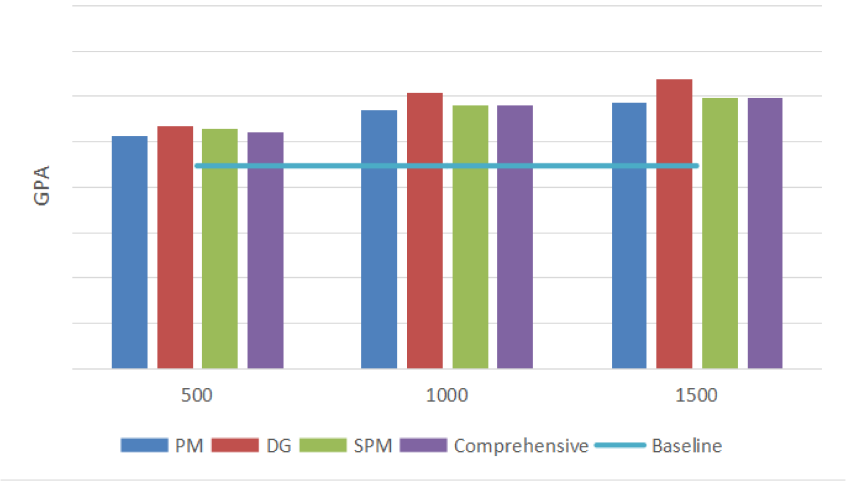


Figure 7.2: 200 students' average GPAs varied by the number of training students of CRS in different approaches

Table 7.6: 200 students' average graduation terms varied by the year of starting CRS based on DG with the full "speedup" setting

Starting Year	Average Graduation Terms
Year 4	11.917
Year 3	11.615
Year 2	11.567
Year 1	11.532

graduation time improvement of our CRS is only in a decimal level, it is already quite a boost considering students only need to study 12 terms in normal scenarios.

Other than recommending courses, our CRS may provide some insights to educators and course counsellors. We mentioned computing courses' GPA contribution and graduation time contribution in Chapter 6. A course's GPA contribution is the average GPA of students who take this course, while a course's graduation time contribution is the average time before graduation of students who take this course. These indicators are used to rank the candidate courses obtained by sequence-based algorithms. Yet, these indicators themselves may have values. Table 7.7 demonstrates the top 5 GPA contribution courses and graduation time contribution courses. One interesting finding is course CMPUT 201. This course is not one of the preferred courses in our simulator but is a prerequisite course for many courses. A preferred course is

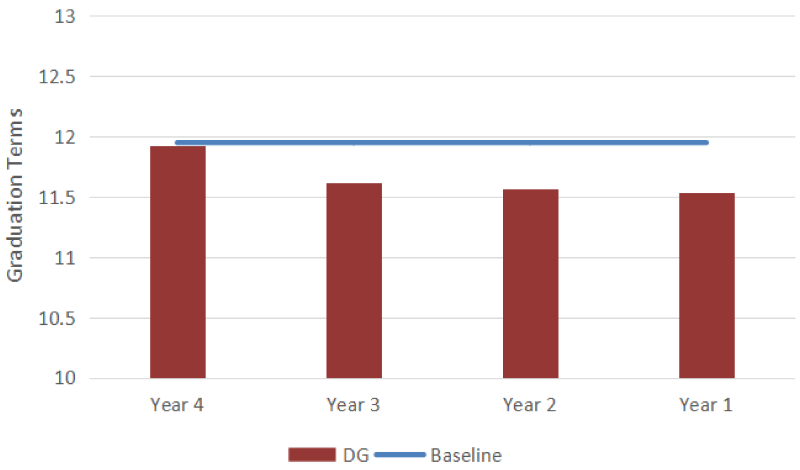


Figure 7.3: 200 students' average graduation terms varied by the year of starting CRS based on DG with the full "speedup" setting

a course that will have a very high probability to be taken in a particular term because it is the "right" course for that term. Being a prerequisite course but not a preferred course means that, CMPUT 201 has to be taken in order to perform well in other courses but many students do not take it. Thus, finding this course actually means that our CRS found an important course that is not in the curriculum but is necessary for students to succeed. As to the graduation contribution list, our first hypothesis is that the list will be filled with key courses, courses that students must take to graduate. However, when we find there is no key course on the list, we realize this hypothesis is wrong. Because if a course is a key course, then nearly all students have to take it, and that will make the graduation time contribution of this course equal to the average graduation time of all students. We could not draw other conclusions from the contribution list, and sometimes it is dangerous to force to do so. For example, CMPUT 275 is in the top position in the GPA contribution list, but we cannot know whether this course causes students to succeed or succeed students like to take this course. Nevertheless, this contribution list will still provide some insights to educators and course counsellors if it is trained by real students' data and is interpreted with caution.

Finally, our CRS can assist educators and administrators to gain deep in-

Table 7.7: The top 5 GPA contribution courses and graduation time contribution courses

Ranking	Top GPA Courses	Top Graduation Time Courses
1	CMPUT 275	CMPUT 301
2	CMPUT 429	CMPUT 274
3	CMPUT 350	CMPUT 300
4	CMPUT 333	CMPUT 410
5	CMPUT 201	CMPUT 366

sights of course relations and thus to improve the curriculum. Figure 7.4 shows the DG of courses with edge colours representing discovery sources. It combines the prerequisite relations used by our simulator and the dependencies discovered by our DGA. On one hand, we can consider the prerequisite course relations used by our simulator as the "current curriculum" or behaviours we expect to see from students. On the other hand, the courses' prerequisite relations discovered by our CRS based on the DG algorithm can be deemed as the prerequisite relations in reality or the actual behaviours by students. Many dependencies used by our simulator are found by our DG algorithm (green edges) like 204⇒304, which means that these rules are carried out successfully by students. Some dependencies used by our simulator are not found in the data (blue edges) like 175⇒229 because the students did not actually follow them, which indicates there are some discrepancies between what we expect from students and what students really perform. Administrators may want to check why this happens. There are also some dependencies found by our DG algorithm but are not in the rules for our simulator (red edges), such as 304⇒366 and 272⇒415. These dependencies indicate some relations among courses unknown and unexpected to administrators but are performed by students. Educators and administrators may want to consider to add these new found prerequisites to the curriculum in the future.

Figure 7.5 shows the paths of successful students (GPA above 3.8) filtered from the 1500 training students with the weight of edges representing the number of students. The thick edges mean many successful students have gone through these paths and they should be paid more attention when trying to improve the curriculum. All in all, the benefits of these findings can be

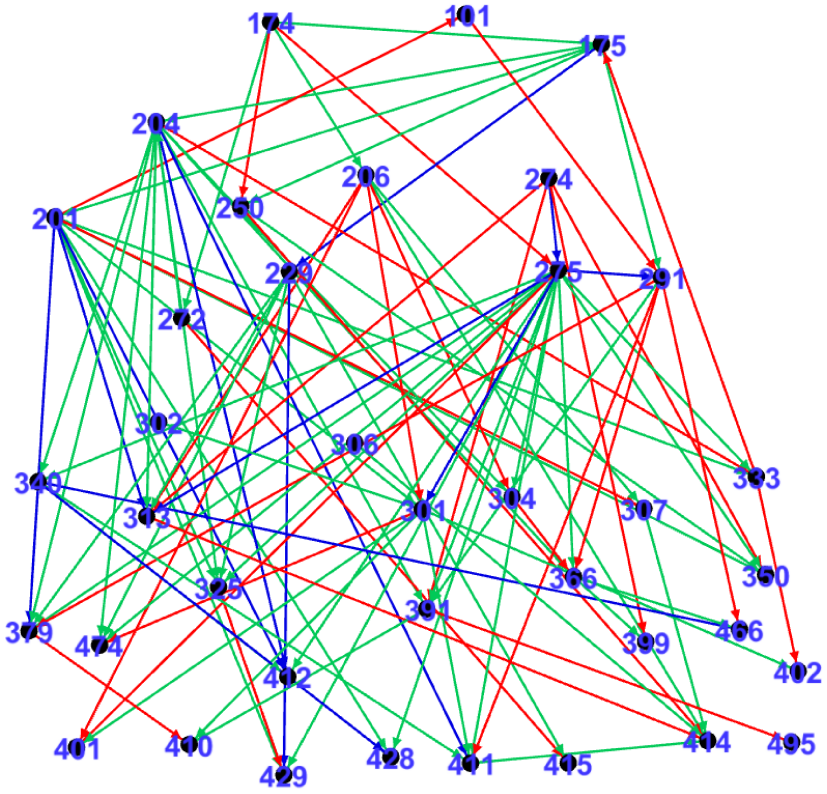


Figure 7.4: The DG of courses with edge colours representing discovery sources

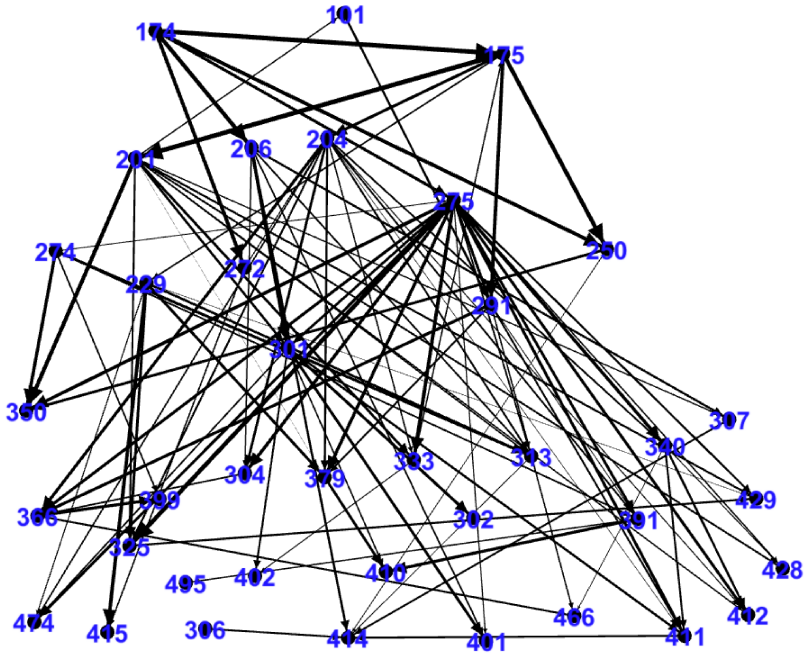


Figure 7.5: The paths of successful students filtered from the 1500 training students with the weight of edges representing the number of students

enormous if our CRS is trained on real students' data.

7.4 Course Recommender System GUI

To help students employ our course recommender system, we built a simple graphic user interface (GUI).

The first interface shown in Figure 7.6 is used by students who need recommendations. Normally, students' courses histories are stored in a database. Since we do not have such a database, a file system is used to keep course history output generated by our simulator and CRS. Students who need recommendation can input their course history file to our CRS. Then students can choose one of the sequential based course recommender algorithms as they want. The comprehensive option is what we introduced in Chapter 6 which gathers results of all three approaches. Students can also select how to rank the potential recommended courses. The ranking method is presented in Chapter 6 as well. The "Hybrid" method is referred as "semi-speedup" while the "To Graduate ASAP" is "full-speedup" in Chapter 6. After that, students can choose how many alternative recommended courses they want to see. With those parameters set, students can simply click "recommend" to get the courses our CRS recommends. Students can also check the detail of the courses we recommended by clicking "course detail" button.

Figure 7.7 shows the GUI of checking a specific course we recommended which includes the course ID, course name, available terms and the instructor. Some details of recommendation are also presented. These include the algorithm we used, the average GPA of students who took this course, whether it is a key course, a new course or a special course that students need, etc.

A GUI shown in Figure 7.8 is made for researchers to generate and analyze simulated data. Researchers can set the experiment output directory and all the parameters of simulator and the course recommender system in a parameter file. If we click "generate data", the data simulator will begin to work and generate simulated students course record history. There are also four options to help researchers directly mining the simulated student data we just

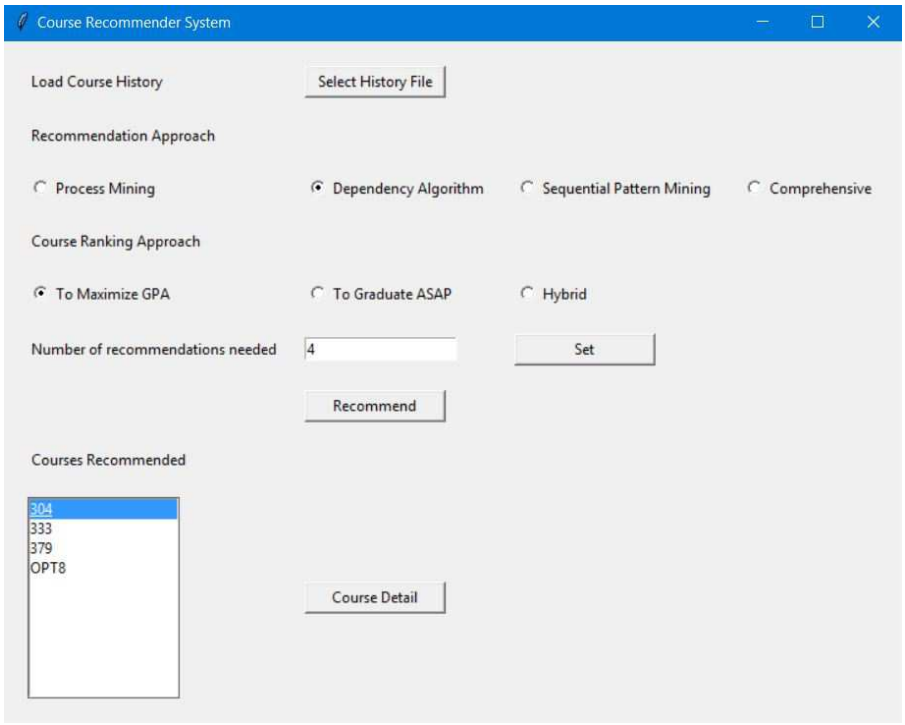


Figure 7.6: GUI of CRS

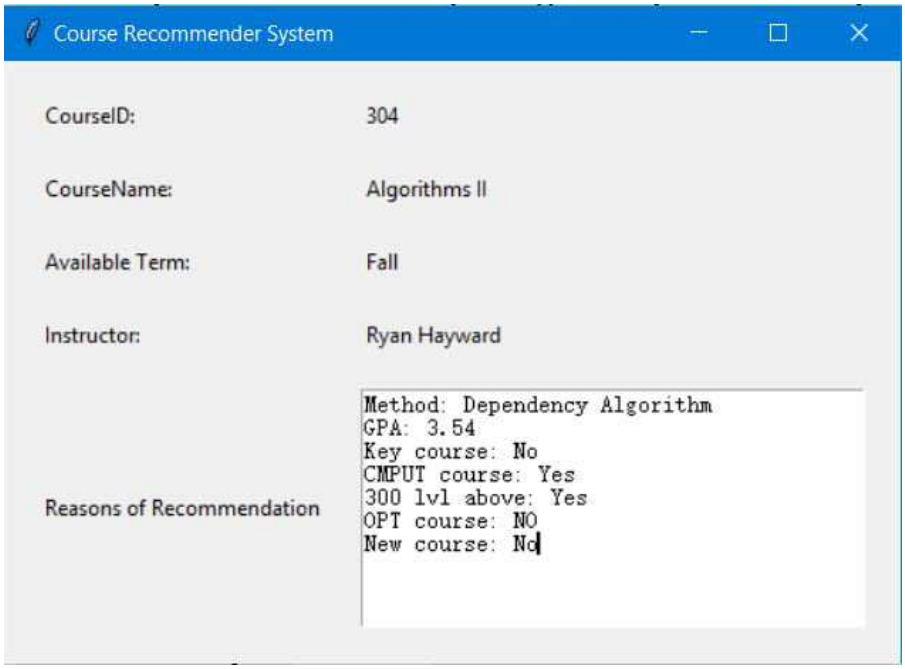


Figure 7.7: GUI of one specific course recommendation

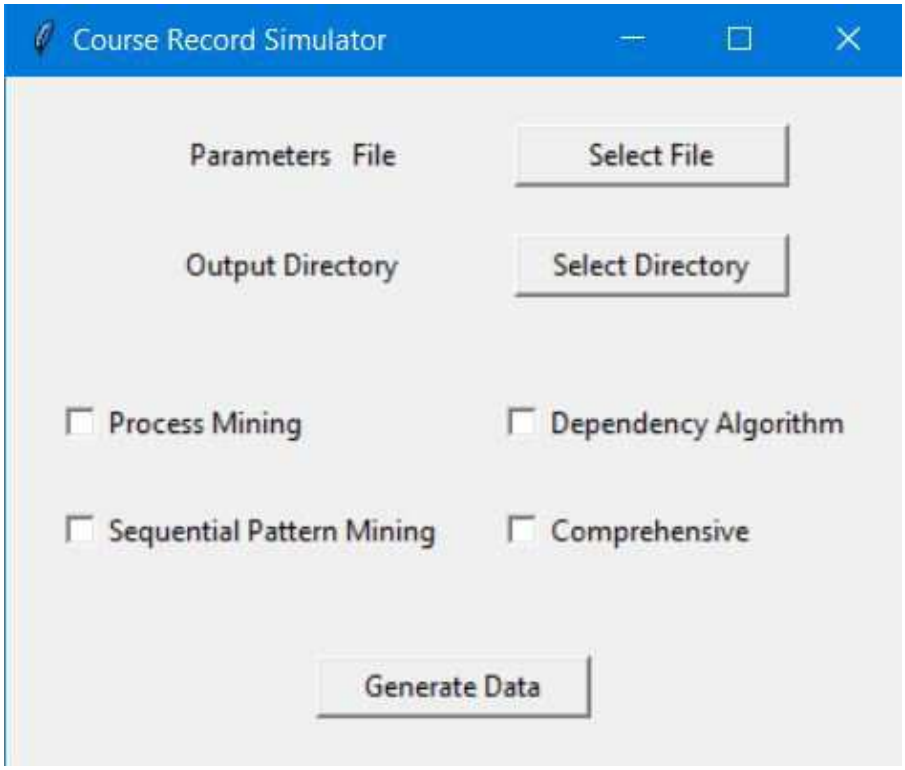


Figure 7.8: GUI of course record simulator

generated by ticking the mining algorithm to run. If no mining approach is chosen in the interface, then it will only generate the simulated data.

Chapter 8

Conclusions and Future Work

In this thesis we have built a course recommender system to help students choose suitable courses in order to maximize their performance. This recommender is based on three different methods yet all three are related to the sequence of course history, which has not been utilized much in the scope of a education recommender. Process mining method is our first approach and its power of conformance checking is what we rely on. We recommend courses to a student that successful students who have a similar course path have taken. The second one is building a course dependency graph which aims to find the deep prerequisite relationships among courses. We recommend courses whose prerequisites are finished. The third method, sequential pattern mining, can help us find frequent sequential patterns, in our case, patterns of successful students. To students who need recommendation, we check what phase they are at in our found patterns, and we help them complete those patterns. Other than students' performance in courses, we also implement an effective method to speed up graduation.

We conduct several experiments to evaluate our course recommender systems and to find the best recommendation approach. All three approaches can improve students performance in different scales. The best recommendation method is based on the dependency graph, and the number of recommended courses accepted by students have a positive correlation with the performance. Moreover, the course recommender system we build can speed up students' graduation if set properly, and provide some useful insights for educators and

course counsellors.

Due to time constraints, there are still many aspects of our work that can be improved upon in the future. Our work is confined by simulated data as a result of ethical problems, even though we try to simulate as real as possible. If we have real student's data, we may gain a deeper and more realistic insight. Also, our approaches are merely based on sequences. Our intention is to bring the novelty of utilizing sequences to build a course recommender system. Yet, with some more information, i.e., personal traits like gender, course information like instructors and diverse measurements of performance, the recommender may work better.

Bibliography

[1] Undergraduate course directory of computing science department of university of alberta. `https://www.ualberta.ca/computing-science/undergraduate-studies/course-directory`.

[2] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. *Mining process models from workflow logs*. Springer, 1998.

[3] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, 22(2):207–216, 1993.

[4] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, pages 3–14. IEEE, 1995.

[5] Judith Arter and Jay McTighe. *Scoring rubrics in the classroom: Using performance criteria for assessing and improving student performance*. Corwin Press, 2000.

[6] Borja Vázquez Barreiros, Manuel Lama, Manuel Mucientes, and Juan C Vidal. Softlearn: A process mining platform for the discovery of learning paths. In *IEEE 14th International Conference on Advanced Learning Technologies (ICALT)*, pages 373–375. IEEE, 2014.

[7] David Ben-Shimon, Alexander Tsikinovsky, Lior Rokach, Amnon Meisles, Guy Shani, and Lihi Naamani. Recommender system from personal social networks. In *Advances in Intelligent Web Mastering*, pages 47–55. Springer, 2007.

[8] Robin Burke. Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer, 2007.

[9] Soumen Chakrabarti, Martin Ester, Usama Fayyad, Johannes Gehrke, Jiawei Han, Shinichi Morishita, Gregory Piatetsky-Shapiro, and Wei Wang. Data mining curriculum: A proposal (version 1.0). *Intensive Working Group of ACM SIGKDD Curriculum Committee*, 2006.

[10] Dean Cummins, Kalina Yacef, and Irena Koprinska. A sequence based recommender system for learning resources. *Australian Journal of Intelligent Information Processing Systems*, 9(2):49–57, 2006.

[11] James Dalziel. Using marks to assess student performance, some problems and alternatives. *Assessment & evaluation in higher education*, 23(4):351–366, 1998.

[12] AK Alves De Medeiros and AJMM Weijters. Genetic process mining. In *Applications and Theory of Petri Nets 2005, volume 3536 of Lecture Notes in Computer Science*. Citeseer, 2005.

[13] Remco Dijkman, Jorg Hofstetter, and Jana Koehler. *Business Process Model and Notation*. Springer, 2011.

[14] Enrique García, Cristóbal Romero, Sebastián Ventura, and Carlos De Castro. An architecture for making recommendations to courseware authors using association rule mining and collaborative filtering. *User Modeling and User-Adapted Interaction*, 19(1-2):99–132, 2009.

[15] Khairil Imran Ghauth and Nor Aniza Abdullah. Learning materials recommendation using good learners ratings and content-based filtering. *Educational technology research and development*, 58(6):711–727, 2010.

[16] Christian W Günther and Anne Rozinat. Disco: Discover your processes. *BPM (Demos)*, 940:40–44, 2012.

[17] Christian W Günther, Anne Rozinat, Wil MP van der Aalst, and Kenny van Uden. Monitoring deployed application usage with process mining. *BPM Center Report BPM-08-11*, pages 1–8, 2008.

[18] Christian W Günther and Wil MP van der Aalst. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *Business Process Management*, pages 328–343. Springer, 2007.

[19] Christian W Günther and HMW Verbeek. Xes-standard definition. 2014.

[20] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 355–359. ACM, 2000.

[21] Yi Hu and Brajendra Panda. A data mining approach for database intrusion detection. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 711–716. ACM, 2004.

[22] Junzo Kamahara, Tomofumi Asakawa, Shinji Shimojo, and Hideo Miyahara. A community-based recommendation system to reveal unexpected interests. In *Multimedia Modelling Conference, 2005. MMM 2005. Proceedings of the 11th International*, pages 433–438. IEEE, 2005.

[23] Raymond Kosala and Hendrik Blockeel. Web mining research: A survey. *ACM Sigkdd Explorations Newsletter*, 2(1):1–15, 2000.

[24] Junzhou Luo, Fang Dong, Jiuxin Cao, and Aibo Song. A context-aware personalized resource recommendation for pervasive learning. *Cluster Computing*, 13(2):213–239, 2010.

[25] Tariq Mahmood and Francesco Ricci. Towards learning user-adaptive state models in a conversational recommender system. In *LWA*, pages 373–378. Citeseer, 2007.

[26] Nikos Manouselis, Hendrik Drachsler, Riina Vuorikari, Hans Hummel, and Rob Koper. Recommender systems in technology enhanced learning. In *Recommender systems handbook*, pages 387–415. Springer, 2011.

[27] Ronny S Mans, Helen Schonenberg, Giorgio Leonardi, Silvia Panzarasa, Anna Cavallini, Silvana Quaglini, and Wil MP van der Aalst. Process mining techniques: an application to stroke care. *Studies in health technology and informatics*, 136:573, 2008.

[28] Ronny S Mans, Helen Schonenberg, Minseok Song, Wil MP van der Aalst, and Piet JM Bakker. Application of process mining in healthcare–a case study in a dutch hospital. In *Biomedical Engineering Systems and Technologies*, pages 425–438. Springer, 2009.

[29] Ronny S Mans, Wil MP van der Aalst, Rob JB Vanwersch, and Arnold J Moleman. Process mining in healthcare: Data challenges when answering frequently posed questions. In *Process Support and Knowledge Representation in Health Care*, pages 140–153. Springer, 2013.

[30] Laura Maruster, Wil MP van der Aalst, AJMM Weijters, Antal van den Bosch, and Walter Daelemans. Automated discovery of workflow models from hospital data. *B. Kroose, M. de Rijke*, 18, 2001.

[31] Laura Maruster, JC Hans Wortmann, AJMM Weijters, and Wil MP van der Aalst. Discovering distributed processes in supply chains. In *Collaborative Systems for Production Management*, pages 219–230. Springer, 2003.

[32] Robert J Marzano, Debra Pickering, and Jay McTighe. *Assessing Student Outcomes: Performance Assessment Using the Dimensions of Learning Model*. Jossey-Bass San Francisco, CA, 1993.

[33] Florent Masseglia, Maguelonne Teisseire, and Pascal Poncelet. Sequential pattern mining. *Encyclopedia of Data Warehousing and Mining*, pages 1028–1032, 2005.

[34] Jan Mendling. *Event-driven process chains (epc)*. Springer, 2008.

[35] Michael P O'Mahony and Barry Smyth. A recommender system for online course enrolment: an initial study. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 133–136. ACM, 2007.

[36] Mykola Pechenizkiy, Nikola Trcka, Paul De Bra, and Pedro Toledo. Currim: Curriculum mining. In *International Conference on Educational data Mining (EDM)*, pages 216–217, 2012.

[37] Mykola Pechenizkiy, Nikola Trcka, Ekaterina Vasilyeva, Wil MP van der Aalst, and Paul De Bra. Process mining online assessment data. *International Working Group on Educational Data Mining*, 2009.

[38] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *icccn*, page 0215. IEEE, 2001.

[39] James Lyle Peterson. *Petri net theory and the modeling of systems*, volume 132. Prentice-hall Englewood Cliffs (NJ), 1981.

[40] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

[41] Cristobal Romero, Sebastian Ventura, Mykola Pechenizkiy, and Ryan SJd Baker. *Handbook of educational data mining*. CRC Press, 2010.

[42] Anne Rozinat, Ivo SM de Jong, Christian W Günther, and Wil MP van der Aalst. Process mining applied to the test process of wafer scanners in asml. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 39(4):474–479, 2009.

[43] D Royce Sadler. Interpretations of criteria-based assessment and grading in higher education. *Assessment & Evaluation in Higher Education*, 30(2):175–194, 2005.

[44] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[45] J Ben Schafer, Joseph A Konstan, and John Riedl. E-commerce recommendation applications. In *Applications of Data Mining to Electronic Commerce*, pages 115–153. Springer, 2001.

[46] Cornelia Schoor and Maria Bannert. Exploring regulatory processes during a computer-supported collaborative learning task using process mining. *Computers in Human Behavior*, 28(4):1321–1331, 2012.

[47] Ramakrishnan Srikant and Rakesh Agrawal. *Mining sequential patterns: Generalizations and performance improvements*. Springer, 1996.

[48] Tiffany Ya Tang and Gordon McCalla. Smart recommendation for an evolving e-learning system. In *Workshop on Technologies for Electronic Documents for Supporting Learning, AIED*, 2003.

[49] Arthur HM ter Hofstede, Wil MP van der Aalst, Michael Adams, and Nick Russell. *Modern Business Process Automation: YAWL and its support environment*. Springer Science & Business Media, 2009.

[50] Nguyen Thai-Nghe, Lucas Drumond, Artus Krohn-Grimberghe, and Lars Schmidt-Thieme. Recommender system for predicting student performance. *Procedia Computer Science*, 1(2):2811–2819, 2010.

[51] Nikola Trcka and Mykola Pechenizkiy. From local patterns to global models: Towards domain driven educational process mining. In *Ninth IEEE International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 1114–1119. IEEE, 2009.

[52] Wil MP van der Aalst. Business process management demystified: A tutorial on models, systems and standards for workflow management. In *Lectures on concurrency and Petri nets*, pages 1–65. Springer, 2004.

[53] Wil MP van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, volume 136. Springer Berlin Heidelberg: Berlin, Heidelberg, 2011.

[54] Wil MP van der Aalst. Challenges in service mining: record, check, discover. In *Web Engineering*, pages 1–4. Springer, 2013.

[55] Wil MP van der Aalst, Hajo A Reijers, AJMM Weijters, Boudewijn F van Dongen, AK Alves De Medeiros, Minseok Song, and HMW Verbeek. Business process mining: An industrial application. *Information Systems*, 32(5):713–732, 2007.

[56] Wil MP van der Aalst, Vladimir Rubin, Boudewijn F van Dongen, Ekkart Kindler, and Christian W Günther. Process mining: A two-step approach using transition systems and regions. *BPM Center Report BPM-06-30, BPMcenter. org*, 6, 2006.

[57] Wil MP van der Aalst and HMW Verbeek. Process mining in web services: The websphere case. *IEEE Data Eng. Bull.*, 31(3):45–48, 2008.

[58] Wil MP van der Aalst, AJMM Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[59] Boudewijn F van Dongen, Ana Karla A de Medeiros, HMW Verbeek, AJMM Weijters, and Wil MP van der Aalst. The prom framework: A new era in process mining tool support. In *Applications and Theory of Petri Nets 2005*, pages 444–454. Springer, 2005.

[60] HMW Verbeek, Joos CAM Buijs, Boudewijn F Van Dongen, and Wil MP van der Aalst. Xes, xesame, and prom 6. In *Information Systems Evolution*, pages 60–75. Springer, 2011.

[61] AJMM Weijters, Wil MP van der Aalst, and AK Alves De Medeiros. Process mining with the heuristics miner-algorithm. *Technische Universiteit Eindhoven, Tech. Rep. WP*, 166:1–34, 2006.

[62] Grant P Wiggins. *Assessing student performance: Exploring the purpose and limits of testing*. Jossey-Bass San Francisco, CA, 1993.

[63] Osmar R Zaïane. Building a recommender agent for e-learning systems. In *Computers in Education, 2002. Proceedings. International Conference on*, pages 55–59. IEEE, 2002.

[64] Mohammed J Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine learning*, 42(1-2):31–60, 2001.

[65] Mohammed J Zaki. Mining data in bioinformatics. *Handbook of Data Mining*, pages 573–596, 2003.