A Hybrid Recommender:  
Study and implementation of course selection recommender engine  

Yong Huang

Institutionen för informationsteknologi  
Department of Information Technology
Abstract

**A Hybrid Recommender: Study and implementation of course selection recommender engine**

Yong Huang

This thesis project is a theoretical and practical study on recommender systems (RSs). It aims to help the planning of course selection for students from the Master Programme in Computer Science in Uppsala University. To achieve the goal, the project implements a recommender service, which generates course selection recommendations based on these three factors:

- student users’ preferences
- course requirements from the university
- best practices from senior students

The implementation of the recommender service takes these three approaches:

- applying frequent-pattern mining techniques on senior students’ course selection data
- performing semantic queries on a simple knowledge organization system (SKOS) taxonomy file that classifies computing disciplines
- applying constraint programming (CP) techniques for problem modelling and resolving when generating final course selection recommendations

The recommender service is implemented as a representational state transfer (REST) compliant web service, i.e., a RESTful web service. The result shows that aforementioned factors have positive impact on the output of the service. Preliminary user feedback gives encouraging rating on the quality of the recommendations.

This report will talk about recommender systems, the semantic web, constraint programming and the implementation details of the recommender service. It focuses on in-depth discussion of recommender systems and the recommender service's implementation.

Keywords: course selection, recommender systems, frequent pattern mining, semantic web, constraint programming
Acknowledgements

First, I would like to thank my thesis supervisor Justin Pearson and my thesis report reviewer Pierre Flener. I have been working a full-time job since starting this project. The time span of the project is hence several years. During these years, Justin has always been patiently giving me support and guide whenever I contact him. I still remember that the university’s library ordered a book about recommender systems very soon after I asked for Justin’s support. Things like this really give me impression that the university and the teachers are trying best to support students’ study. He always listened carefully when I explained my progress and ideas to him. I felt pleasant to have discussion with him. My report is a long one. But that did not stop him reading it several rounds with constructive and detailed comments. I am very thankful to it. I took a course about constraint programming taught by Pierre in my first semester study in Uppsala University. It was a challenging course and through it I knew Pierre has a high standard requirement of students’ work. When reviewing my thesis report, Pierre gives me a lot of very detailed comments. Some sparkling comments from him has made improving the report a very pleasant experience. Thank you, Pierre, for your reviewing, which has given me more confidence in the quality of my report.

I would also like to thank Uppsala University and Sweden for offering me the opportunity to study and live here. Because of the generosity of the Swedish society, as a foreigner to study master degree, I did not need to pay the tuition fee when I started my study in 2009. Otherwise I could not be able to afford it. This opportunity has given me a meaningful experience. I have always felt grateful to this opportunity.

Friendship is always one important element to make life colourful. I enjoy my life in Uppsala very much. My friends are an essential part of my life in Uppsala. From flatmates to great friends, Xu Cheng and I have grown our friendship over the years, out of our passion to basketball, Chinese cooking and travel. Age difference is never a problem when it comes to friendship. Anders Berglund was my teacher and later we became good friends. We have developed our discussion topics from my study to international news, Chinese food, cheese and wine. To my other friends: Wu Weiling, Xue Linyan, Wan Ru, Zheng Liying, Li Hao, Zhao Chenyue, it is so great that I have met you in Uppsala.

Growing up in a village in Southwest China and now living in Stockholm, I know I could not make it without the love and support from my family. I feel extremely lucky that my parents have always supported me to pursue my dreams and each member of my great family cares each other so much. My fondest love and thanks to my parents, my uncles, my aunts and my cousins.

Last but not least, my special thanks to my dear and beautiful girlfriend Liu Jinyan. Thank you for your love and company during the past years. You are my dear lover and meanwhile best friend. It has been so wonderful to explore the beauty of Sweden with you. It is always pleasant to have discussions with
you. Your endless energy and passion about life have always kept me optimistic about tomorrow while enjoying the best of today. Thank you and love you.

Yong Huang (黄勇)
Stockholm
February 2017
Contents

1 Introduction
   1.1 Background ............................................. 1
   1.2 Solution Proposal ...................................... 1
   1.3 Relevant Work .......................................... 2
   1.4 Overview of this Report ................................. 3

2 Survey to Collect Best Practices
   2.1 Design of the Survey ................................. 4
   2.2 Results and Findings of the Survey ..................... 4
   2.3 Application of the Findings in this Project .......... 5

3 Design Overview
   3.1 Introduction ........................................... 7
   3.2 CSP Modelling .......................................... 7
   3.3 Workflow of Recommendation Generation .............. 8

4 Recommender Systems
   4.1 Introduction ........................................... 12
   4.2 RS Implementation Approaches ......................... 13
      4.2.1 Content-Based RS ................................. 13
      4.2.2 Collaborative Filtering (CF) Based RS ............ 13
      4.2.3 Demographic Profile Based RS .................... 14
      4.2.4 Knowledge-Based RS .............................. 14
      4.2.5 Community-Based RS .............................. 15
      4.2.6 Hybrid RS ......................................... 15
   4.3 Study of Example Algorithms ........................... 16
      4.3.1 Content-Based Algorithms ........................ 16
      4.3.2 Collaborative Filtering (CF) Algorithms ......... 19
   4.4 Interesting Challenges and Solutions .................. 22
      4.4.1 The Cold Start Problem ............................ 23
      4.4.2 The Large Scale Data Set Problem ................. 23
      4.4.3 The Sparsity Problem in Collaborative Filtering (CF) . 24
   4.5 RSs Practice in this Project .......................... 24
      4.5.1 FP-Growth Algorithm for Frequent Pattern Mining . 25
<table>
<thead>
<tr>
<th>Appendices</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A  The Complete Survey and Its Result</td>
<td>77</td>
</tr>
<tr>
<td>B  FP-Growth Frequent Pattern Mining Sample Code</td>
<td>89</td>
</tr>
<tr>
<td>C  CSP Model Imposing Constraints Sample Code</td>
<td>91</td>
</tr>
<tr>
<td>D  User Evaluation Feedback</td>
<td>93</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter will start with discussing the background problem area of this project. After that, it will continue to present the proposed solution. In the end, it discusses some relevant work from other literature.

1.1 Background

At the Master Programme in Computer Science from Uppsala University (henceforth to be referred as the programme), students freely select courses to attend (except one mandatory course). To be eligible to apply for the degree diploma after the study, these attended courses must meet the requirements from the programme. For example, the programme requires students to earn at least 120 credit points from their study, which usually consists of attending courses and finishing a thesis project.

While my fellow students and I enjoy the freedom of selecting courses, we also face challenges. For instance, in the survey conducted in this project, 85.72% (24 out of 28 responses, 4 skipped) respondent students said that study workload imbalance had affected study outcome. Another finding from the survey shows that there is a high demand on the relevance information regarding courses to computing disciplines (85.72%, 24 out of 28 responses, 4 skipped).

Even though not mentioned in the survey, the diversity of the courses might help introduce challenges. For example, there were around 100 courses to select from during 2015-16 academic year. These courses vary in computing disciplines, credit points (e.g., 5, 10, 15, etc.) and levels (i.e., advanced and basic).

1.2 Solution Proposal

This project tries to address the aforementioned challenges. It aims to help students create a study plan that meets requirements from the programme, satisfies personal preferences and incorporates best practices from senior students.
This project is proposed out of my interest in the relevant technologies and the match between the technologies and the problem area. I found myself interested in the courses about data mining and constraint programming after attending them during my study in the programme. And I had learned some semantic web knowledge from my work experience before studying in Uppsala.

It was my hope that I could apply these interesting technologies to build software that can create practical value. The course selection problem is essentially about solving a combinatorial problem. That makes it a good candidate for constraint programming. Finding best practices from senior students' course selection data would naturally guide a programmer to search solution from the data mining field.

Hence the project was proposed with a clear understanding of the problem and the resolving technologies.

1.3 Relevant Work

Since the problem and the resolving technologies were clear from the beginning, this project is more engineering oriented than research oriented.

The study of relevant work was done after the recommender was implemented. This section will do some comparison between the recommender from this project and the ones from related work.

As pointed out in [7] and [54], there are just a few implementations of course selection recommenders. Nevertheless, neither [7] nor [54] generates the recommendation as a complete study plan, but a list of preferred courses. In contrast, this project not only tries to match user interest, but also satisfies some academic requirements. More importantly, the final recommendation from this project is a complete study plan.

The work of [7] solely used the association rule mining technique from the data mining field for implementation. The implemented algorithm was called Apriori and had the drawback of candidate generation as discussed in Section 4.5.1. This project implements a hybrid recommender by applying both data mining and semantic web technologies. The implemented frequent pattern mining algorithm in this project does not generate any candidate pattern, which helps gain performance and improve scalability.

The work of [54] implemented a hybrid recommender. It generated a list of collaborative filtering based recommendations (Section 4.2.2) and a list of content based recommendations (Section 4.2.1).

The authors of [36] stated there was no formal study on academic requirements when they started the project. Their work emphasized modelling academic requirements as constraints. The recommender from [36] supported more complex academic requirements than this project’s recommender, e.g., course dependencies. Two implementation approaches were discussed in [36]. They were based on flow algorithms and integer linear programming. As the authors said, their work focused on satisfying constraints and the work was done in the context of an existing system. Hence there were no details on how the
candidate courses were generated before solving the constraints. In contrast, this project covers not only details on constraint solving but also on candidate generation. And the implemented recommender is published as an open source project, which enhances the possibility for reuse and extension.

1.4 Overview of this Report

The remaining part of this report will start with discussing the survey from this project in Chapter 2. Then it will describe the design overview in Chapter 3. The overview covers constraint satisfaction problem (CSP) modelling and the workflow of recommendation generation. The details of the design and the supporting technologies will be discussed from Chapter 4 to 6. These chapters will discuss in detail three computing fields (including recommender systems, the semantic web and constraint programming) and their practice in this project. The report ends with the discussion of the recommender service’s implementation details and test results in Chapter 7, followed by a description of future work in Chapter 8 and a conclusion in Chapter 9.
Chapter 2
Survey to Collect Best Practices

In order to find out the challenges and collect quantitative information about best practices on course selection, a survey was conducted in the beginning of this project. This chapter will discuss the design of the survey, its findings and the use of these findings in this project.

2.1 Design of the Survey

From both personal experience and talks with fellow students, I have noticed that study workload, requirements from the programme and personal preferences are the key factors influencing course selection. The tangible information of these factors is essential for the implementation of the course selection recommender. The survey is hence designed to capture such information.

The survey consists of ten questions, which cover the topics of study workload, personal preferences in computing disciplines and best practices. Seven of these questions are semi-close-ended and the remaining three are open-ended. The "Yes" answers to the semi-close-ended questions represent the opinions based on my personal experience and communication with fellow students. They are expected to receive positive responses.

The survey was published on the web [14] to collect responses. It is also available at Appendix A. The target population were senior students from the programme.

2.2 Results and Findings of the Survey

In total, there were 32 respondents. Table 2.1 gives three example questions and their results. The complete results are available at Appendix A.
<table>
<thead>
<tr>
<th>Question</th>
<th>Response</th>
</tr>
</thead>
</table>
| Q2: Do you think that around 15 credits per period is the most reasonable plan? | Answered: 32
Yes: 87.50%
Other opinions: 12.50% |
| Q4: Do you think doing your master thesis in the last semester is the best choice? | Answered: 32
Yes: 81.25%
Other opinions: 18.75% |
| Q6: What additional information do you want to get while making your course plan? | Answered: 28, Skipped: 4
85.72% Voted for: "More detailed information about courses, such as which CS subfield do a certain course belong to."
Other opinions: 21.43%
Note: some respondents have checked both options. |

Table 2.1: Part of the survey on course selection planning

Overall, the results align with the expectation, i.e., the "Yes" answers get significant support. For example, the support ratios of the "Yes" answers in Table 2.1 are all above 80%. And even the lowest support of the semi-close-ended questions' "Yes" answers is 65.52% (the 5th question in Appendix A). In general, the results reveal these findings:

- around 15 credit points per study period is a balanced study workload;
- doing master thesis at the last semester is a best practice;
- the mapping information from courses to computing disciplines is important;
- the study plan should reflect personal preferences.

2.3 Application of the Findings in this Project

The findings from Section 2.2 are the guidelines to the implementation of the recommender in this project. This section will elaborate the application of these findings in this project.

- around 15 credit points per study period is a balanced study workload

This finding helps build some of the constraints in the CSP model of this project. For example, a Sum constraint is applied over the total credit points from a
study period. The value 15 is used to restrict the domain of the sum variable in this constraint.

- doing master thesis at the last semester is a best practice

This project assumes the student user will do the thesis project at the last semester. Therefore, it only recommends a course selection for the first three semesters (i.e., 6 study periods) of the two-year programme.

- the mapping information from courses to computing disciplines is important

The mapping information from courses to computing disciplines is a source to generate candidate courses in this project. For example, the recommender allows student users to specify preferred computing disciplines. With this input and the mapping information from courses to computing disciplines, the recommender can deduce that the relevant courses might be of interest to the student users. These courses are then used as candidates to generate the final recommendations.

- the study plan should reflect personal preferences

When the recommender generates recommendations, firstly it uses the explicitly specified courses as a candidate list to build domains for the variables in the CSP model. If no solution is found for the model, then it continues to enlarge the candidate list by deducing courses from the explicitly specified computing disciplines. If there is still no recommendation generated, then it will at last continue to enlarge the candidate list through frequent pattern mining, which will be covered in detail in Section 4.5.1. By doing so, the recommender tries to reflect a student user’s preferences as much as possible.
Chapter 3

Design Overview

Overall, this project implements a hybrid recommender, which applies more than one recommendation generation approach. This chapter starts with a brief description of the recommender and then continues to give an overview of its design.

3.1 Introduction

The major user inputs of the recommender include a set of courses and a set of computing disciplines, indicating personal preferences. The outputs of the recommender are a set of course selection recommendations. Each of these recommendations consists of a set of courses with relevant information, such as name, credit points, schedule information, etc. The problem of generating these recommendations is modelled as a CSP. The remaining part of this chapter will discuss how the user inputs and other data sources are used to build and solve the CSP.

3.2 CSP Modelling

A CSP consists of a set of variables, their corresponding domains and a set of constraints upon them.

The course selections for the six study periods are modelled as variables in this project’s CSP model. The domains of these variables are the intersections between the scheduled courses from the programme and the ones that the recommender considers to be of interest to the student user. And the constraints are based on the requirements from the programme (e.g., total credit points) and findings from the survey (e.g., around 15 credit points per period).
3.3 Workflow of Recommendation Generation

The variables and constraints in this project’s CSP model are fixed. But the domains of the variables are dynamic because of the recommendation generation strategy. The strategy can be described as follows:

1. At the beginning, the recommender uses the intersections between user specified courses and the scheduled ones from the programme to decide the CSP variables’ domains.

2. If no solution is found for the CSP model and the student user has enabled the computing discipline deduction feature, then the recommender enlarges the CSP variables’ domains.

   The domains are enlarged with the course list generated through computing discipline deduction. This process involves two steps. The first step is to generate a list of disciplines. It is done through a semantic web query. The second step is to generate the course list. It is done by using the generated discipline list and the mapping information from courses to computing disciplines. The second step is basically the process of finding the mapping keys with values in a collection of key-value pairs.

3. If there is still no solution to the CSP model and the student user has enabled the frequent pattern mining feature, then the recommender enlarges the variables’ domains again.

   This time, the domains are enlarged with the course list generated through frequent pattern mining. The mining process uses user specified courses and history course selection data from senior students.

To put it all together, the following flowcharts present the aforementioned procedure:

- Figure 3.1 explains the process of building the CSP model.
- Figure 3.2 explains the process of performing deduction on computing disciplines and courses to generate an additional candidate course list.
- Figure 3.3 explains the process of recommendation generation.
Figure 3.1: The flowchart of CSP modelling

1. Declare 6 CSP variables to represent the course selections for the 6 study periods.
2. Define the domains for the 6 CSP variables with these data sources:
   a. user inputs, e.g., preferred courses;
   b. the history course selection data;
   c. the knowledge of computing disciplines;
   d. the course schedule from the programme.
3. Define CSP constraints with these data sources:
   a. the findings from the survey;
   b. the requirements from the programme.
The flowchart of deducing preferred courses from the mappings of courses to computing disciplines

Figure 3.2: The flowchart of deducing preferred courses from the mappings of courses to computing disciplines
Figure 3.3: The flowchart of recommendation generation
Chapter 4

Recommender Systems

From this chapter onward, the following three chapters will introduce the major supporting technologies in this project. They include recommender systems, the semantic web and constraint programming.

This chapter will discuss recommender systems (RSs). It starts with a brief introduction to RSs and then continues with a discussion on the implementation approaches of RSs. After that, some example algorithms will be discussed. At last, this chapter ends with a detailed discussion on the algorithm FP-Growth [21] that is used for frequent pattern mining in this project.

4.1 Introduction

RSs are software tools and techniques providing suggestions for items to be of use to a user [43]. They have wide commercial use, e.g., the shopping item recommendation service from Amazon.com [18] and the Who-to-Follow recommendation service from Twitter [19].

As a computing field, RSs also draw interest from research and academic communities. For example, the annual conference ACM Recommender Systems Conference has been held since 2007. The online education platform Coursera teaches RSs in one of its specialization offerings [29].

Before continuing with the discussion, let me introduce some RS terms used in this chapter. An object that an RS examines and recommends is called an item. An item is represented by its properties. These properties are called attributes or features. For example, a restaurant is an item in a restaurant RS and it can be represented by features like name, cuisine, cost, etc. The process that a user shows her preference of an item is called rating. A user’s rating history consists of all the ratings she has made. Rating portfolio is another way to express rating history, but from an analytical perspective. For example, a user’s rating history consists of 10 ratings. Her rating portfolio covers 4 ratings on restaurants, 4 ratings on gyms and 2 ratings on online learning platforms.
4.2 RS Implementation Approaches

The implementation of RSs can be categorized into six approaches [43], which will be discussed from Section 4.2.1 to Section 4.2.6. In general, the course selection recommender in this project takes the hybrid approach introduced in Section 4.2.6. In detail, the recommender takes approaches from both Section 4.2.2 and Section 4.2.4.

4.2.1 Content-Based RS

In this approach, recommendations are generated by comparing the candidate items with the ones that the user liked before [43]. The similarity of the items is calculated using their features. For example, if a user likes camping and picking mushrooms, then fishing in the lakes can be recommended as an appealing activity to this user, because these activities are all relaxed outdoor activities.

One obvious point with this approach is that it does not use rating history data from other users. Instead, it uses the rating history from the current active user and the features’ data of the related items.

But one drawback is that the final diversity of the user’s rating portfolio can be limited. Items that are not similar to any rated items from the current user’s rating history will not be recommended.

4.2.2 Collaborative Filtering (CF) Based RS

In this approach, recommendations are generated based on the rating history from other users who have similar taste as the current active user [43]. CF can be further divided into two categories [16]: user-user CF and item-item CF.

- User-User CF: The idea is first to find other users who have similar rating history to the current active user and then use these users’ ratings on other items as references to generate the recommendation. These users are considered as neighbours to the current active user.

  These neighbour users’ ratings might cover items that the current user has not considered before. Hence it helps to enrich the diversity of the current user’s rating portfolio.

  This approach involves the process to compute the similarity between users. An example algorithm for this approach is discussed in Section 4.3.2.

- Item-Item CF: The idea is to find rating patterns that indicate item similarity. In such a pattern, a set of items are frequently liked together by users. And hence these items are considered similar to each other. If the current active user likes any item from such an item set, then the remaining items from the set can be recommended.

  The difference between content-based filtering and item-item CF is about the approach to find item similarity. Content-based filtering uses a single
user’s rating data and item metadata. But item-item CF uses different users’ rating data.

In summary, user-user CF computes user similarity and item-item CF computes item similarity. Both use rating data from other users.

Elaboration on Item-Item CF

Item-item CF inspires the implementation of this project’s recommender. The recommender takes a data mining approach to find similar items patterns. In the implementation, history course selection records are collected anonymously. Every single student’s attended courses are considered as a transaction data set. Similar item patterns are considered as the frequent patterns from these transaction sets.

To understand better the process, let me define frequent pattern first. As its name tells, a frequent pattern represents a set of items that appear together frequently in all the transaction sets [5]. When saying frequent, it means the pattern’s appearance frequency matches the threshold requirement. For example, given 100 transaction sets, when the threshold requirement for a frequent pattern is 25, it means all the items in the frequent pattern must appear together in at least 25 transaction sets out of the total transaction sets.

When the specified courses from a student user are contained by a frequent pattern, the remaining courses from the pattern will be used as candidate courses to generate a recommendation.

4.2.3 Demographic Profile Based RS

In this approach, recommendations are generated based on the demographic profile of the user [43]. For example, when a user visits a global online store’s website, the user is usually redirected to the local website of the store after she selects the preferred language of the site. Correspondingly, the list of recommended merchandises is generated based on the locale information.

4.2.4 Knowledge-Based RS

In this approach, recommendations are generated based on domain knowledge [43]. Case-based RSs and constraint-based RSs are two categories from this approach. Both share some common processes, e.g., collecting user requirements, proposing repairs while these requirements are not satisfied, etc. But they also have a difference: case-based RSs mainly use similarity metrics between user requirements and item features but constraint-based RSs use explicit problem domain knowledge.

Elaboration on Knowledge-Based RS

This project uses both the case-based RS approach and the constraint-based RS approach. Here is the explanation:
• The recommender uses the preferred computing disciplines specified by the student user to enlarge the list of candidate courses. This process does not have strict requirements on the candidate courses, regarding their covered computing disciplines. For example, if the student specifies software testing as her interested computing discipline, then the course “Large Scale Programming” can be a candidate course, which also covers other topics in addition to software testing. This course is considered as a candidate course, because its content is considered to be similar to the user’s requirement. This approach can be considered as case-based RS.

• The recommender’s CSP model uses requirements from the programme to build constraints, e.g., the constraint upon total credit points. The recommender is strict on such requirement. That’s because if this requirement is not satisfied, then the student will not be eligible to apply for a degree after following the recommended study plan. In this case, the recommender is taking the constraint-based RS approach.

4.2.5 Community-Based RS

In this approach, recommendations are generated based on the preferences of the user’s friends [43]. The rationale behind this approach is based on people’s social behaviour. People tend to perform the same activities as their friends do.

The community-based approach is very similar to the collaborative filtering approach. They both use rating data from other users, who are considered to have similar taste as the current active user does. But they differ in the details. For collaborative filtering (CF), people sharing a similar taste do not necessarily have to be friends. CF has to calculate either user similarity or item similarity. But for community-based RSs, the process of calculating user similarity can be shortened to a great extent, because it already takes the user’s friends as people with similar taste.

4.2.6 Hybrid RS

In this approach, recommendations are generated based on the combined application of the aforementioned approaches [43]. The recommender in this project is in fact a hybrid one. It is explained as follows:

• it takes the CF approach when generating candidate courses with senior students’ course selection records through frequent pattern mining;

• it takes the case-based approach from knowledge-based RSs when generating candidate courses with the preferred computing disciplines;

• and it takes the constraint-based approach from knowledge-based RSs when building the constraints for its CSP model.
4.3 Study of Example Algorithms

This section will introduce some typical RS implementation algorithms. For example, it will introduce the ID3 algorithm [41], which is used to build a decision tree [72] to examine the features of items in content-based RSs.

4.3.1 Content-Based Algorithms

The key elements for content-based RSs are a user profile model built with her rating history and an approach describing an item with its features. The algorithms from [38] can be used to build content-based RSs. Below are some of them.

ID3 Decision Tree

When a decision tree is applied in content-based RSs, a user’s rating history is used as training data to build the tree. The tree is then used as a model to classify the target item and decide whether the item should be recommended. ID3 is an example algorithm to build a decision tree and it was invented by Ross Quinlan [41].

Figure 4.1 is an example of a data set and its corresponding decision tree. [15] shows how a decision tree is built step by step. ID3 has two important elements:

- **Entropy** is a term from information theory. It was expressed as an equation when Ross Quinlan introduced ID3 in [41]. The following is the description:

  \[
  \text{Entropy} = - \sum p \log_2 p - \sum n \log_2 n
  \]

  where \(p\) and \(n\) are the number of objects in class \(P\) and \(N\), respectively.

  Given an arbitrary collection \(C\) of objects, which contains \(p\) objects of class \(P\) and \(n\) of class \(N\), then \(C\)’s corresponding decision tree can be regarded
as a source of a message ‘P’ or ‘N’, with the expected information needed to generate this message given by the equation:

\[ I(p, n) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n} \]

This equation gives the entropy for the complete decision tree. Since a decision tree is built over the attributes of the objects in the collection, the branching nodes in the tree represent these attributes. The entropy for such nodes is expressed slightly differently. For example, the entropy for the root node is expressed as the following:

Given the root node is corresponding to attribute A, which has values of \{A_1, A_2, ..., A_v\}, then the entropy for the tree with A as root is given as:

\[ E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I(p_i, n_i) \]

- **Information Gain** is a term from information theory. \[41\] defines information gain of branching over attribute A as:

\[ \text{gain}(A) = I(p, n) - E(A) \]

When building the tree, ID3 uses the attribute with maximum information gain as root node. Each branching is a sub-tree, which can recursively apply the same strategy until the complete tree is built.

**k-Nearest Neighbour (k-NN)**

k-NN \[25\] classifies an item by looking at the classification information of the item’s closest neighbours. At first, it tries to find out the target item’s closest neighbours and then examines the classification information of these neighbours. The item is classified to the class that the majority of its neighbours belong to. The leading character \(k\) represents the number of neighbours to check.

The rationale behind the algorithm is that if most of one’s closest friends belong to one group, then one most probably also belongs to the same group. Accordingly, in a recommender, if an item is considered to belong to the same group of most of the items a user liked before, then the recommender will recommend this item to the user.

Some distance functions can be used to measure an item’s neighbourhood, e.g., Euclidean distance \[25\].

**Naïve Bayesian**

The naïve Bayesian classifier \[20\] is a probabilistic classifier and is based on Bayes’ theorem \[20\]. The remaining part of this section will start with the introduction to Bayes’ theorem and then discuss how Bayes’ theorem is used in the naïve Bayesian classifier.
Bayes’ theorem can be expressed as the following formula:

\[
P(H|X) = \frac{P(X|H) \cdot P(H)}{P(X)}
\]

The symbols and terms in the formula have the following explanations [6] [20]:

- \( X \) is considered as “evidence” in Bayesian terms. It is described by measurements made on a set of \( n \) attributes. In the context of a classification problem, \( X \) represents an item to be classified.

- \( H \) is a hypothesis. For example, in the context of a classification problem, \( H \) can be the hypothesis that \( X \) belongs to a specified class \( C \).

- \( P(H|X) \) is the probability that \( H \) holds, given that we know the attribute description of \( X \). It is also called posterior probability because it is derived from or depends on the specified value of \( X \).

- \( P(H) \) is the prior probability that \( H \) holds. It is “prior” in the sense that it does not take into account any information about \( X \).

- \( P(X|H) \) is the posterior probability of \( X \) given that \( H \) holds. For example, in the context of a classification problem, \( P(X|H) \) is the probability that an item is \( X \), given that we know the item belongs to a class \( C \).

- \( P(X) \) is the prior probability of \( X \).

The following is an example of applying Bayes’ theorem to calculate the probability of playing golf using the weather outlook [32]:

- \( X \) is the weather outlook.

- \( H \) is the hypothesis that a positive decision to play golf will be made.

- The posterior \( P(H|X) \) is the probability of a positive play golf decision, given the outlook is sunny.

- The prior \( P(H) \) is the probability of a positive play golf decision (among all decisions).

- The likelihood \( P(X|H) \) is the likelihood that the outlook is sunny, given it is a positive play golf decision.

- The prior \( P(X) \) is the proportion of decisions made on a sunny day to all the decisions.

When classifying an item, the naïve Bayesian classifier takes the following steps (these steps are briefly mentioned here, refer to [20] for elaborations):

1. Suppose there are \( m \) classes, \((C_1, C_2, ..., C_m)\), into which the items can be classified. The item \( X \) is represented by an \( n \)-dimensional attribute vector, \((x_1, x_2, ..., x_n)\).
2. The probability that $X$ belongs to a class $C_i$ is computed with Bayes’ theorem as:

$$P(C_i|X) = \frac{P(X|C_i) \cdot P(C_i)}{P(X)} \quad (1 \leq i \leq m).$$

3. Since $P(X)$ is constant for all classes, the problem to decide the class label of $X$ is in fact to find a class $C_i$ so that the class meets the following condition:

$$P(X|C_i) \cdot P(C_i) > P(X|C_j) \cdot P(C_j) \quad (1 \leq j \leq m, j \neq i).$$

4. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, i.e., $P(C_1) = P(C_2) = \cdots = P(C_m)$. So the class labelling of $X$ turns into the task of calculating $P(X|C_i)$.

5. Since $X$ is represented with a vector, the calculation of $P(X|C_i)$ is defined as the following formula in [20]:

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \cdot P(x_2|C_i) \cdots \cdot P(x_n|C_i).$$

The calculation of $P(x_k|C_i)$ is covered in [20].

In the context of a content-based recommender, an item can be represented with its attribute vector. A naïve Bayesian classifier is then used to label the item’s class (e.g., to be included in the recommendation list or not). Based on the class labelling result, the recommendation decision is then made.

### 4.3.2 Collaborative Filtering (CF) Algorithms

The following sections will present example algorithms from [16]. They cover both user-user CF and item-item CF.

**User-User CF**

User similarity is the key to user-user CF RSs and it is computed with the users’ previous rating data. Pearson correlation [39] is one approach to compute the similarity.

Pearson correlation is a statistical method. As stated in [40], ‘Pearson correlation is a measure of the linear correlation between two variables $X$ and $Y$, giving a value $r$ between +1 and −1 inclusive, where 1 is total positive correlation, 0 is no correlation, and −1 is total negative correlation. It is widely used in the sciences as a measure of the degree of linear dependence between two variables’.
An explanation of the method is available at [39].

In user-user CF RSs, each commonly rated item by two users has a rating pair, which represents its ratings from these two users. These rating pairs are considered as coordinates of points. Hence all the rating pairs can be plotted on a coordinate graph. The Pearson correlation coefficient of these points will represent the similarity of these two users. This can be visualized with Figure 4.2. The figure shows some cases with different Pearson correlation coefficients $r$. In each case, the x-axis and the y-axis represent the users $x$ and $y$ respectively; each point represents an item rated by the two users; the correlation coefficient $r$ represents the similarity of the two users. When $r$ is 1, it means the two users have high similarity.

In addition to statistical methods, linear algebra can also be used to measure similarity, e.g., cosine similarity [16]. In a vector space, as stated in [49],

‘the angle between two vectors is used as a measure of divergence between the vectors. The cosine of the angle is used as the numeric similarity (since cosine has the nice property that it is 1.0 for identical vectors and 0.0 for orthogonal vectors). As an alternative, the inner-product (or dot-product) between two vectors is often used as a similarity measure’.

With cosine similarity, a user’s ratings on $n$ items are represented by an $n$-dimensional vector. Each component in the vector means a rating towards an item. And the cosine similarity between two users is computed with their rating vectors’ dot product and Euclidean lengths. Figure 4.3 gives an example
of cosine similarity on 2-dimensional vectors. A more detailed description can be found at [16].

**Item-item CF**

Algorithms used to implement item-item CF RSs can be further divided into the following two categories [16]: binary-valued rating domain (e.g., like/dislike) and a broader scale of real-valued rating domain (e.g., the number range from 0.0 to 5.0 with steps of 0.5). The remaining part of this section will explain these two categories.

- Item-item CF – binary-valued rating domain

A binary-valued rating domain contains value pairs such as like and dislike. An example algorithm for this category is from Amazon. Amazon introduced an item-item CF algorithm [18] to generate purchase recommendations on its online store. In a purchase transaction database, a product has either been purchased by a user or not yet. Hence, a customer’s purchase history toward a product can be represented as binary feedback.

Amazon’s algorithm uses a vector to represent a product’s purchase history from customers. Each component in the vector represents a customer’s purchase history on this product. The algorithm computes the similarity of items by computing the cosine similarity of these vectors. The algorithm is described in pseudo code as in Algorithm 1.

The algorithm builds a product-to-product matrix, with each node representing the cosine similarity of a product pair. The matrix can be precomputed.
The algorithm’s complexity at worst case is $O(N^2 \cdot M)$ with $N$ being the number of products and $M$ the number of customers. According to [18], in practice, the complexity is around $O(N \cdot M)$. That is because of the small ratio when comparing the products that most customers have purchased with the total number of products in the catalogue.

For Amazon, the numbers of its customers and products are both at the millions level. One of the approaches it uses to generate recommendations is using the customers’ purchase history and a precomputed matrix. This combination of offline and online processing helps solving the scalability problem and the recommendation quality. The algorithm is discussed in detail in [18].

- **Item-item CF – real-valued rating domain**

The real-valued rating domain has broader domain values compared with binary-valued rating domain. For example, a real-valued rating domain can consist of a number range from 0.0 to 5.0 with steps of 0.5. The slope one algorithms [28] are a family of simple algorithms to generate recommendations for RSs in this category. The algorithms take the simple linear regression (also called predictor) with the form of $f(x) = x + b$ to calculate rating differences between items. In [28], three slope one algorithms are discussed: slope one (namely basic slope one), weighted slope one, and bi-polar slope one. A detailed description of basic slope one and weighted slope one is available in [34]. The description is also available online at [12].

Here is an example to explain the basic idea of the algorithms. Assume there are two users, UserA and UserB; two items, ItemA and ItemB; UserA has given ratings to ItemA and ItemB with values of 3 and 5 respectively; we also know UserB has given a rating to ItemA with value of 1; with basic slope one, the predicted rating that UserB will give to ItemB can be calculated as $v = 1 + (5 - 3) = 3$.

When relating the equation above to the $f(x) = x + b$ form, the value 1 that UserB gives to ItemA represents $x$. And the value of 2 (from 5 – 3) represents the constant $b$.

One key element in implementing slope one algorithms is to have an item-item rating matrix. The value of a matrix node represents the rating difference. It is calculated with ratings from users who have rated both items. When a recommender is built with a rich data set, the rating difference between two items is an average of the differences from all users who have rated both items. When generating a candidate item list, basic slope one computes the average differences between the not-rated items and the rated ones. Then it returns the list upon the average differences rank.

### 4.4 Interesting Challenges and Solutions

Professor John Riedl, a world-renowned expert in the field of RSs [30], mentioned some research challenges in this field [44]. Some of the example challenges are listed below:
• evaluating recommenders (e.g., measuring customer loyalty for an online store)

• interpreting user action (e.g., analysing the implied information from user behaviour)

• dealing with the cold start problem (e.g., recommending new items to a new user)

From Section 4.4.1 to Section 4.4.3, three challenges will be studied, including the cold start problem, the large data set problem and the data sparsity problem.

4.4.1 The Cold Start Problem

The cold start problem is usually explained with examples, e.g., explanations in [46] and [37]. This section will take the same approach to explain it.

The cold start problem in RSs can mean the following cases as also mentioned in [37]:

• generating recommendations on new items to users, about whom the recommender already has knowledge, e.g., preference rating history;

• generating recommendations to new users with existing items, which other users have already rated;

• generating recommendations to new users with new items.

Cold start problems are solved with different approaches. For example, the community-based approach is applied when recommending existing items to new users, as studied in [48]. The idea is to use the rich information from social media platforms to model the users’ community and hence find user similarity. This user similarity information is then used to generate recommendations.

In [37], all aforementioned three cold start problems are addressed. The proposed methodology takes a regression approach, using a user profile vector and an item profile vector. In addition to these two vectors, the approach also uses a weight variable to characterize the affinity of these two vectors. The approach uses knowledge from linear algebra, probability and regression in statistics.

4.4.2 The Large Scale Data Set Problem

While recommender systems emerged as a research area in the mid-1990s [43], that was not in the big data era yet. But nowadays, for companies with huge numbers of customers and products, how do they generate recommendations? For example, Amazon.com had 270 million active customers as in 2014 [33] and Twitter had 200 million users, generating 400 million tweets every day in as early as 2013 [19]. This section will use one of Twitter’s services as an example to show how RSs can scale to handle big data.
One important service Twitter provides to its users is the Who-to-Follow service [19]. This is a recommendation service Twitter uses to connect users sharing common interests, connections and other factors.

The key element in the recommender service is the open source in-memory graph processing engine, Cassovary [53]. The recommender models the user-follow relationship as a directed graph. A snapshot of the graph, including all users, is loaded into memory first. And then Cassovary will process the graph to help generate user-follow recommendations. The final recommendations are a combined result from around 20 algorithms. SALSA (stochastic approach for link-structure analysis) [19] is one of these algorithms. SALSA is performed on the basis of “circle of trust” as mentioned in [19]. The generated recommendations are stored in a database called WTF (WhoToFollow). The front-end clients, e.g., web page, will retrieve recommendations from the WTF database and eventually display them to end users.

One highlight point of the service is that it handles 200 million users’ recommendations in memory on a single server with 144 GB RAM. More detail of the service is available at [19].

4.4.3 The Sparsity Problem in Collaborative Filtering (CF)

The sparsity problem happens in user-user CF RSs when rating info is not sufficient to generate recommendations. One example condition is that there are too many users and items, but even the very active users can only rate on a small number of items when comparing with the total number of items; and even the very popular items can only receive ratings from a small number of users when comparing with the total number of users.

In user-user CF RSs, the users’ rating on items can be represented as a user-item matrix. Each row represents the ratings that a user gives to all the items. One approach to address the problem is dimensionality reduction [45]. Another approach is to use trust inferences as described in [35]. The approach builds a trust relationship between users in the context of a social network. To become a registered user of the social network, a user is required to submit at least one rating to any item that has been rated by a second user. The approach considers user-user similarity as trust among users. It computes the similarity using Pearson correlation with the ratings on commonly rated items. It also introduces trust inferences and a trust path among users. Confidence properties and uncertainty properties are also discussed. Evaluation shows the approach has outstanding recommendation quality.

4.5 RSs Practice in this Project

This section will discuss the frequent pattern mining algorithm FP-Growth [21]. It is applied to generate candidate courses in this project. This algorithm can be categorized as an item-item CF algorithm, because the items from a frequent pattern are considered to be similar to each other.
4.5.1 FP-Growth Algorithm for Frequent Pattern Mining

Introduction

A frequent pattern in a transaction database is defined as a set of items, whose occurrence frequency meets the minimum threshold requirement. The absolute occurrence frequency of an itemset is sometimes called support, e.g., in [21].

Frequent pattern mining is related to association rule mining, which takes the form of: \( X \Rightarrow Y \). The expression means that \( Y \) is likely to happen when \( X \) happens. One typical example of association rule is that every time certain customers buy some items, they also tend to buy the same set of other items.

The Apriori algorithm [70] is one of the popular association rule mining algorithms. It starts with a frequent pattern mining task and then uses the result patterns to generate association rules. It takes the generate-and-test approach for frequent pattern mining. The process of frequent pattern mining starts with fewer items. Then it continues to generate candidate sets with more items and to test these new sets. It has an obvious drawback that a lot of candidate item sets are generated and tested, especially when the item base in the problem domain is huge.

Because of the scalability concern for the Apriori algorithm, this project implements another algorithm called FP-Growth [21]. It performs frequent pattern mining without generating candidate itemsets.

FP-Growth Algorithm Elaboration

The FP-Growth algorithm includes two tasks: build an FP-tree and mine frequent patterns from the FP-tree.

An FP-tree consists of a header table and a prefix tree. Each element in the header table has two fields: item ID and the head of node link. The node link points to the first inserted node in the tree that has the same item id as the element from the header table. The elements in the header table are sorted in descending order using their support count. The table only includes elements whose support meets the threshold requirement.

Each node in the prefix tree has four fields: Item ID, Support count, Node-link to the same id node (inserted later than the current one) and a link to its parent node.

Figure 4.4 is an example FP-tree. The support counts of the head table’s elements are not mandatory.

FP-tree can be built with Algorithm 2. As the algorithm shows, the tree building process requires only two complete scans of the database. The result tree can be a very condensed data structure if many transactions share the same prefix path.

The frequent patterns of an FP-Tree can be mined with Algorithm 3 from [21]. The terms conditional pattern base and conditional FP-tree are defined in [21].

When implementing this algorithm, this project used a different version from [69]. The basic idea is still the same, but the version from [69] first mines on the single path prefix (if any) and branching sub-trees (if any) respectively;
then it gets the complete frequent pattern sets by returning the union and the cross products of these results.

Sample FP-Growth Java Code

Appendix B provides sample Java code from this project. The sample code performs the frequent pattern mining part of FP-Growth algorithm.
Scan the transaction database for the first time to build the header table $HT$;
Create the root node marked as null for the result tree $Tree_r$;
Let a root reference $Ref_{root}$ point to the root node;
Scan the transaction database for the second time, do:

forall transaction $T_i$ do
    Sort the items in $T_i$ according to the order in header table $HT$;
    forall item $I_j$ in $T_i$ do
        if the node pointed by the root reference has a direct child node with the same id as $I_j$ then
            Increase the count of the child node by 1;
            Let $Ref_{root}$ point to this child node;
        else
            Create a new tree node;
            Set the ID of the new node to be same as the one of $I_j$;
            Set the count of the new node to be 1;
            Follow the node link from the same id element in header table $HT$ to the end;
            Point the end of the node link to this new node;
            Point the parent link of this new node to the node pointed previously by the root reference;
            Let $Ref_{root}$ point to the new node;
    end
end

Algorithm 2: Build an FP-Tree

Call procedure $FP$-Growth($FP$-tree, null)

Procedure $FP$-Growth($tree_i$, α):
if $tree_i$ contains only a single path $P$ then
    forall combination (denoted as $\beta$) of the nodes from $P$ do
        Generate a new pattern from $\beta \cup \alpha$ with its support as the minimum support of the nodes in $\beta$;
    end
else
    forall $\alpha_i$ in the header table of $tree_i$ do
        Generate pattern $\beta = \alpha_i \cup \alpha$ with $\beta$'s support as $\alpha_i$.support;
        Construct $\beta$'s conditional pattern base and then $\beta$'s conditional
        $FP$-tree $Tree_\beta$;
        if $Tree_\beta \neq \emptyset$ then
            Call $FP$-Growth($Tree_\beta$, $\beta$)
        end
    end
end

Algorithm 3: Frequent pattern mining upon an FP-Tree
Chapter 5

Semantic Web

This project uses semantic web technology for deducing additional computing disciplines a student user might like, based on her specified ones. This chapter will describe how the deducing process works. It starts with an introduction to the semantic web.

5.1 Introduction

The basic ideas of the semantic web were described in [52]. One of the ideas, quoted from the paper, is:

‘The semantic web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation’.

Another paper [11] from 2009 said:

‘The vision of a semantic web has been interpreted in many different ways ... However, despite this diversity in interpretation, the original goal of building a global Web of machine-readable data remains constant...’.

Now in 2015, on the World Wide Web Consortium’s (W3C) page about the semantic web [63], it also says: ‘The ultimate goal of the Web of data is to enable computers to do more useful work...’.

The semantic web as a field has evolved over time. For example, [63] lists some existing semantic web technologies, e.g., RDF (resource description framework), SPARQL (SPARQL protocol and RDF query language), OWL (web ontology language) and SKOS (simple knowledge organization system). These technologies’ development timeline is presented in [31]. They will be discussed in Section 5.3.
5.2 The Technology Layer Stack

In [52], three elements are mentioned when discussing the functionality of the Semantic Web: a) the structured collections of information, b) inference rules and c) automated reasoning.

The first two elements are considered to be the foundation for the third one. But what technologies are supporting these elements? The semantic web layer cake explains it. As stated in [47], ‘the semantic web layer cake is an illustration of the hierarchy of languages, where each layer exploits and uses capabilities of the layers below’. In [8], Tim Berners-Lee presented the semantic web layer cake as in Figure 5.1. The supporting semantic web technologies are categorized into different layers in this figure.

Over the past years, different technologies have been developed and they have enriched the technology stack in Figure 5.1. In 2009, this stack was revisited by James A. Hendler [22]. He then presented an enhanced diagram as Figure 5.2.

If Figure 5.1 shows a design, then Figure 5.2 shows a realization. Technologies like RDF, SPARQL and OWL were available in 2009 and are now W3C standards.

W3C’s categorization of semantic web technologies is the following [63]:

- **Linked Data**: publishing and connecting structured data on the web, with technologies like RDF, RDF in attributes (RDFa), etc.

- **Vocabularies**: defining concepts and relationships used to describe and represent an area of concern, with technologies like RDF, RDF schema (RDFS), SKOS, OWL, etc.
5.3 Technology Elaboration

This section elaborates on some key semantic web technologies, such as RDF, OWL, SKOS, etc. It starts with an example and then continues with the explanations on the supporting technologies.

5.3.1 An RDF/XML Example

Let us begin with an RDF/XML example from [59]. The example consists of Figure 5.3 and its corresponding RDF/XML document in Listing 5.1. In this
There are two objects, namely a person named BOB and the painting ‘The Mona Lisa’.

BOB is interested in ‘The Mona Lisa’. He was born on 1990-07-04 and he knows Alice.

The painting ‘The Mona Lisa’ has the title ‘Mona Lisa’. Its creator was ‘Leonardo Da Vinci’. It is the topic of a webcast from the website http://www.europeana.eu.

The figure presents an RDF graph (Section 5.3.2) and the document presents an RDF/XML document (Section 5.3.2).

5.3.2 Resource Description Framework (RDF)

W3C has a suite of documents to cover different aspects of RDF. For example, [58] describes the concepts and the abstract syntax of RDF and it also discusses the specifications of some RDF-based languages such as Turtle, RDF/XML, etc.

In RDF’s data model (i.e., abstract syntax), the core structure is the concept of triple. A triple, as shown in Figure 5.4, consists of a subject, a predicate (considered as a verb, representing a relationship) and an object.

All these three elements in a triple can be represented with a uniform resource identifier (URI). An RDF triple is also known as an RDF statement. A set of related triples form an RDF graph, which can be expressed with an RDF-based language such as the RDF/XML [60].
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF

xmlns:dcterms="http://purl.org/dc/terms/"
xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:schema="http://schema.org/"

<rdf:Description rdf:about="http://example.org/bob#me">
  <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
  <schema:birthDate rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
    1990-07-04
  </schema:birthDate>
  <foaf:knows rdf:resource="http://example.org/alice#me"/>
  <foaf:topic_interest rdf:resource="http://www.wikidata.org/entity/Q12418"/>
</rdf:Description>

<rdf:Description rdf:about="http://www.wikidata.org/entity/Q12418">
  <dcterms:title>Mona Lisa</dcterms:title>
</rdf:Description>

  <dcterms:subject rdf:resource="http://www.wikidata.org/entity/Q12418"/>
</rdf:Description>
</rdf:RDF>

Listing 5.1: Example RDF/XML document
5.3.3 RDF Schema (RDFS)

Similar to XML Schema [68], RDFS provides a rich data-modelling vocabulary for RDF languages, e.g., rdfs:Resource, rdfs:Class, etc. Refer to [61] for more details about RDFS.

5.3.4 Web Ontology Language (OWL)

In the semantic web, an ontology is specified in [56] as: ‘a set of precise descriptive statements about some part of the world (usually referred to as the domain of interest ... )’.

As mentioned in [52], a typical ontology in the semantic web has a taxonomy and a set of deduction rules. The taxonomy defines classes of objects and relations among these classes.

OWL is a semantic web language used to describe an ontology. OWL 2 was introduced to distinguish from the old version OWL. OWL 2 ontology differs from other ordinary RDF-based documents in the way that an OWL 2 ontology has its focus and usually addresses the domain of common interest to the general public. Even though an OWL 2 document can express an ontology, it does not necessarily cover how to do deduction based on the declarative statement, i.e., missing deduction rules. Deduction rules will be discussed in Section 5.3.5.

An OWL 2 ontology can be modelled as an RDF graph and be expressed with an RDF-based language, e.g., RDF/XML. Take Figure 5.5 for example, the centre eclipse represents the abstract notion of an ontology, which can be considered as an RDF graph. As shown at the top of the figure, an ontology can be used to produce an RDF/XML document; meanwhile, an RDF/XML document can be parsed into an ontology. These are in fact the processes of serializing and deserializing an ontology. These processes enable the exchange of ontologies. The other parts of the figure are explained in detail in [55].

A complete OWL 2 ontology example is available in [57].

5.3.5 Rule Interchange Format (RIF)

As pointed out in [62], a rule can be a production rule, which is related to the idea of instruction (e.g., if A, then do something) or a declarative rule, which is related to declaring a fact (e.g., if A, then B is true). The semantic web RIF Working Group addresses both types of rules.

RIF is for the purpose of exchanging rules in the semantic web but it is just a format. There are concrete RIF languages, e.g., RIF-BLD (basic logic
dialect) [62], RIF-PRD (production rule dialect) [62].

An example RIF-Core rule is given as Listing 5.2 from [62]. The example uses IMDB and DBpedia as fact resources regarding actors in the cast of a film. The rule says that if a fact from IMDB shows that an actor plays a role and this role is in a certain film, then there exists the fact in DBpedia that this actor is in the cast of that film.

At [67], the W3C RIF Working Group provides a suite of documents on other topics related to RIF, e.g., RIF XML mapping to RDF, etc.

```xml
Document(
  Prefix(rdfs <http://www.w3.org/2000/01/rdf-schema#>)
  Prefix(imdbrel <http://example.com/imdbrrelations#>)
  Prefix(dbpedia <http://dbpedia.org/ontology/>)

  Group(
    Forall ?Actor ?Film ?Role (
      If And(imdbrel:playsRole(?Actor ?Role)
                           imdbrel:roleInFilm(?Role ?Film))
        Then dbpedia:starring(?Film ?Actor)
    )
  )
)
```

Listing 5.2: Example RIF-Core rule
5.3.6 Simple Knowledge Organization System (SKOS)
SKOS is an RDF-based vocabulary for representing semi-formal knowledge organization systems, e.g., taxonomies, classification schemes, etc [66]. The word semi-formal tells the difference between SKOS and OWL in the way that OWL is a more formal approach to express the meaning of information.

An example of using SKOS is ACM’s Computer Classification System [1], which is a taxonomy of the computing field.

5.3.7 SPARQL Query Language for RDF (SPARQL)
Similar to SQL (structured query language), SPARQL uses keywords like SELECT, WHERE, etc in its query statement. For example, Listing 5.3 is a SPARQL query statement to find the name and the number of friends of each person in a target group.

Listing 5.3: Example SPARQL statement

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name (COUNT(?friend) AS ?count)
WHERE {
  ?person foaf:name ?name .
} GROUP BY ?person ?name
```

5.4 Industry Application and Tool Support
Semantic web technologies have been applied in the health care and life sciences disciplines according to W3C [65]. More activities can be found on W3C’s semantic web interest group’s page [64].

As a collaborative community activity, www.schema.org is created to promote schemas for structured data.

Some tools are available to facilitate the use of semantic web technologies, such as the open source framework Apache Jena.

5.5 Semantic Web Practice in this Project
The course selection recommender directly uses SPARQL, SKOS, and the open source framework Apache Jena [2].

The recommender uses a SKOS file [1] from ACM. The SKOS file is in fact a taxonomy on computing disciplines. It defines the relationship between different computing disciplines, e.g., which discipline is related to which. In the SKOS file, computing disciplines are organized as a tree structure. For example, the computing discipline covers sub-disciplines like hardware, networks, and
software and its engineering, etc. Listing 5.4 is a sample part of the SKOS file. This sample part defines the discipline Data Mining as a SKOS concept.

The course selection recommender uses the SKOS file as a source for deducing computing disciplines that are considered to be of interest to a student user. The deduction is implemented as a SPARQL query by using this SKOS file and the specified computing disciplines from a student user. The query is performed by Apache Jena.

Listing 5.5 shows the SPARQL statement for querying the narrower computing disciplines of data mining. The token #10003351 in the listing corresponds to the RDF URI for data mining in the SKOS file. The token skos:narrower is an element from the SKOS vocabulary. It is considered as the sub-discipline relationship in this project.

The complete list of computing disciplines considered to be of interest to the student user then consists of the following two parts: the specified computing disciplines from the user and the deduced computing disciplines from the SPARQL query. With this list and the mapping information between courses and computing disciplines, the recommender can find a list of courses that are considered to be of interest to the student user. These courses are used to enlarge the CSP variables’ domains when generating the final recommendations.
Chapter 6

Constraint Programming

The course selection recommender uses user inputs, computing discipline deduction and frequent pattern mining to generate candidate courses. Once the candidates are ready, it is the time to generate the final recommendations. This is achieved by searching for solutions to the CSP model in the recommender. This chapter will discuss constraint programming (CP) and the CSP model in the recommender.

6.1 Introduction

Like many others, my acquaintance with constraint programming started with the sudoku puzzle. So, it is no surprise that I start the introduction to constraint programming with the sudoku puzzle. Figure 6.1 is an example of a sudoku puzzle.

A sudoku puzzle gives some initial numbers to certain cells and leaves the others blank. To solve the puzzle, it is required to fill the blank cells with numbers from one to nine and meet the requirements listed below:

- each number can only appear once in each row;
- each number can only appear once in each column;
- each number can only appear once in each block.

Figure 6.2 is the solution to the puzzle in Figure 6.1.

The sudoku puzzle is an example of combinatorial task and can be solved with CP. The task of solving a sudoku puzzle can be modelled as a constraint satisfaction problem (CSP). A CSP consists of three elements: a) a set of variables abstracted from the problem; b) the corresponding value domains of these variables; c) a set of constraints which specify the required relations among the variables.

A solution to a CSP is an \( n \)-tuple of values, where \( n \) is the number of the variables in the CSP model. Each value in the tuple is assigned to its
corresponding variable in the CSP and all the requirements from the constraints in the CSP are met with this assignment. A formal definition of CSPs can be found in [10].

Below is an example CSP model for the sudoku puzzle:

- each cell is modelled as a variable;
- the blank cells have the domain of $[1, 9]$ (1 to 9, inclusive) and each already initialized cell has the given value as domain;
- constraints requiring variables to have different values to each other are applied to the cells on each row, each column and each block respectively.

Once the modelling is done, the CSP model can be implemented and solved with a constraint programming solver such as JaCoP [27].

As pointed out in [4], ‘the idea of constraint programming is to solve problems by stating constraints (requirements) about the problem area and, consequently, finding a solution satisfying all the constraints’.
6.2 Some Key Concepts and Techniques

This section will cite the content from [4] to explain some key CP concepts and techniques.

To solve a constraint satisfaction problem, systematic search algorithms can be applied. Below are two examples:

- **generate-and-test (GT):** generate a complete labelling for all variables and then test against the constraints. A failed testing will trigger the generation of another labelling;

- **backtracking (BT):** generate incrementally a labelling for the variables in a step-by-step manner until a complete satisfying labelling is found. If a partial solution violates any constraint, then backtracking is performed to the most recently instantiated variable that still has alternatives available. BT differentiates itself from GT in the way that GT labels all the variables for each labelling while BT gradually increases the number of variables in the labelling.

In addition to systematic search, the use of consistency techniques [4] is another approach to find solutions. A consistency technique removes inconsistent
values from the variables’ domains until a solution is found. Different consistency algorithms are discussed in [4], such as node-consistency, which works with unary constraints (constraint on a single variable); arc consistency, which works with binary constraints (constraints over a pair of variables); path consistency, which also works on binary constraints but involves more than two variables.

Since CP handles combinational problems, optimisation is important. Constraint optimisation techniques, such as branch and bound (B&B) and partial constraint satisfaction, are also covered in [4].

Another important CP concept is constraint propagation, which is expressed as: ‘domain reduction due to one constraint can lead to new domain reduction of other variables’ in [42]. In [9], constraint propagation is defined as: ‘a very general concept, which appears under different names such as constraint relaxation, filtering algorithms, narrowing algorithms, constraint inference, simplification algorithms, label inference, local consistency enforcing, rules iteration, chaotic iteration’.

One more important CP concept is a global constraint. A global constraint is a constraint that captures a relation between a non-fixed number of variables, such as the $\text{AllDiff}(x_1,...,x_n)$ constraint [23]. The AllDiff constraint requires all the variables to have a different value and 27 such constraints can be used to model any $9 \times 9$ grid sudoku puzzle: with 9 for the rows, 9 for the columns and 9 for the blocks.

6.3 Tools

Constraint solvers are software tools used to find CSP solutions. For example, Gecode [17] is a C++ based solver and JaCoP [27] is a Java based one. JaCoP is used in this project. Some example solvers are listed in [71] with a brief introduction. Another reference [13] gives a better organization of some CSP solvers by categorizing them according to programming languages.

6.4 CP Practice in this Project

This section will discuss this project’s CSP model in detail.

6.4.1 The CSP Model

A CSP model can be represented as triple $\langle X, D, C \rangle$, where $X$ is a set of variables; $D$ is the set of corresponding domains of the variables; $C$ is a set of constraints.

This section will start with introducing the variables of the CSP model, i.e., the set $X$; then it continues to explain how the domains of these variables are defined, i.e., the set $D$; in the end, it discusses the constraints in the CSP model, i.e., the set $C$. Some sample Java code of the CSP model’s implementation is discussed in Section 6.4.3.
• the variables

Since the course recommendation covers six study periods, there are six variables in the model. Each variable represents the set of courses recommended for its corresponding study period. During the six study periods, each course is identified by a unique integer value. Hence each study period variable has the type of a set variable with integer elements. The variable type is declared as SetVar in JaCoP. So the variables of the model are defined as follows:

\[ X = \{P_1, P_2, \ldots, P_6\}, P_i(i \in [1, 6], \text{SetVar}) \]

• domains of the variables

The student office publishes the scheduled courses before a semester starts. The set of scheduled courses for study period \( i \) is then called \( \text{Scheduled}_i \) in the model.

With the work from early chapters, a set of candidate courses is generated to represent user interest. The set consists of user specified courses, courses found from semantic web deduction, and courses found from frequent pattern mining. (The set is gradually enlarged at runtime because of the recommender’s strategy.) This set is called \( \text{Preferred}_{\text{all}} \).

Then the domain \( D_i \) for study period \( i \) is defined as follows:

\[ D_i = [\emptyset, \text{Scheduled}_i \cap \text{Preferred}_{\text{all}}], i \in [1, 6] \]

The set of the domains for the variables is defined as follows:

\[ D = \{D_1, D_2, \ldots, D_6\} \]

• constraints

The constraints reflect the requirements from the programme and the best practices from previous students, as discussed in Section 2.3:

1. Take no more than three courses per study period.

   The constraint is defined as follows:

   \[ \text{Cardinality}(P_i) \in [1, 3], i \in [1, 6] \]

2. Take the recommended number of courses for all six study periods.

   By default, the recommender uses 15 as the lower bound and 18 as the upper bound of the number of the total recommended courses. The recommender allows the user to specify the upper bound of the total number, called \( \text{maxTotalCourseNumber} \).

   So the constraint on total recommended courses is defined as follows:

   \[ \sum_{i=1}^{6} \text{Cardinality}(P_i) \in [15, \text{maxTotalCourseNumber}] \]
3. Take the mandatory courses in the corresponding study period. This requirement is modelled as an *element in set* constraint, called $EinA$.

When a course $C_m$ is required to be taken in study period $i$, the constraint is defined as follows:

$$EinA(C_m, P_i)$$

4. Avoid selecting the same course more than once.

The same course may be scheduled at more than one study period during the six periods. When this case happens, this course may have the same code or different codes in these periods. The recommender should include at most one occurrence of such courses in the final recommendation.

The following steps are taken to implement this constraint:

(a) Define a *set* to hold all the unique ids representing the scheduled occurrence of each course that has more than one scheduled occurrence. For example, the same course can be scheduled for both the 1st period and the 5th period. This course will have two unique ids from these two periods to form its id set.

Since the recommender assumes the 5th and 6th periods have the same scheduled course sets as the 1st and 2nd periods respectively, there exist such id sets representing the same courses. If the total number of such courses is called $totalMoreThanOnce$, then all the sets are as follows:

$$SameCourseIdSet_i, i \in [1, totalMoreThanOnce]$$

(b) Define a *SetVar* to represent the union of the recommended courses for the six periods. It’s defined in the equation below. The implementation code is listed in Listing 6.2.

$$U_{all} = P_1 \cup P_2 \cdots \cup P_6$$

The union $U_{all}$ is gradually acquired by applying the JaCoP constraint $AunionBeqC$ to $P_i$. (This union will be reused by the following constraints as well.)

(c) Get the intersection between $SameCourseIdSet_i$ and $U_{all}$

$$SameCourseOccurrence_i = SameCourseIdSet_i \cap U_{all},
\quad i \in [1, totalMoreThanOnce]$$

(d) Post the cardinality constraints on the intersection sets above as follows:

$$\text{Cardinality}(SameCourseOccurrence_i) \in [0, 1],
\quad i \in [1, totalMoreThanOnce]$$
5. Meet the advanced course credits requirement.
   Since the programme restricts a maximum of 30 basic credits in a student’s total study credits and this recommender assumes the thesis project has 30 credits, the total advanced courses in the recommendation have a minimum of 60 credits.

   The approach to constrain the total advanced credits consists of the following steps:
   (a) define the SetVar variables to represent the intersection sets between $U_{all}$ and the advanced courses from the different credit categories, e.g., 7.5 points, 10 points, etc.
   (b) post the cardinality constraints on the SetVar variables above;
   (c) post the arithmetic constraints $X \cdot Y = Z$, where each $X$ and each $Y$ represent the cardinality and the corresponding credit of each SetVar variable above respectively; each $Z$ represents the numeric product variable. The array advancedCreditsFromDifferentCreditCategories represents all these product variables;
   (d) post the constraint Sum on the product variables above. The variable creditsAdvanced is the sum variable in the Sum constraint.

   The following formula presents the final constraint above:

   $$\text{creditsAdvanced} = \text{Sum}(\text{advancedCreditsFromDifferentCreditCategories})$$

   $$\text{creditsAdvanced} \in [60, \text{maxAdvanced}]$$

   maxAdvanced uses the default value unless the user has specified one.

6. Take courses for a reasonable number of credits in each period.
   When posting this constraint, the model uses a similar strategy as constraining advanced credits to constrain credits for each period. The differences are:
   - it does not only look at advanced courses but also includes basic level ones in the constraints.
   - it looks at the intersection between $P_i$ and courses from different credit categories, instead of using $U_{all}$.

7. Earn at least 90 credits as required by the programme.
   The model applies the same credit constraint posting strategy as for the advanced courses when posting this one. The only difference is that it does not only look at advanced courses but also includes basic level ones.
The model has a default value for the maximum total credits (called \( \text{maxTotal} \)) and it also accepts a customized value from the user. The constraint is applied when defining the domain of the total credits variable (called \( \text{creditsTotal} \)) and posting a \( \text{Sum} \) constraint over the course credits from different credit categories (represented as \( \text{creditsFromDifferentCategories} \)):

\[
\text{creditsTotal} = \text{Sum}(\text{creditsFromDifferentCategories})
\]

\[
\text{creditsTotal} \in [90, \text{maxTotal}]
\]

### 6.4.2 Notes on the CSP Model Implementation

My CSP model uses the constraints only applicable to integer values. But some courses have credits with float values, e.g., 7.5 points. To solve this problem, the CSP model multiplies all credit values by 10 to scale them up.

### 6.4.3 Sample Implementation Java Code

The CSP model is implemented in Java and it uses the library JaCoP. This section will start with introducing how to use the JaCoP library and then continues to give some sample code to explain how the CSP model is implemented in this project.

**Using JaCoP**

Listing 6.1 is a modified version of an example from [26]. It shows how to use JaCoP. The CSP model in the example is about assigning different values to three variables. The assignment \{\( v[0] \mapsto 2, v[1] \mapsto 3, v[2] \mapsto 1 \)\} is a solution to the model in the example.

As the code comment shows, the steps to use JaCoP include: 1. define a store; 2. define and initialize variables; 3. define and impose constraints; 4. search for a solution.

**Sample Code from this Project**

A Java project (Maven module) is created to model and solve the CSP problem in this project. The code structure of the project is presented in Figure 6.3. As the figure shows, the classes under package \( \text{se.uu.it.cs.recsys.constraint.solver} \) are in charge of building the CSP model and searching for a solution; the classes under package \( \text{se.uu.it.cs.recsys.constraint.constraints} \) are in charge of defining and imposing constraints.

As mentioned earlier when describing the CSP model constraints, one important constraint from the model is to find the course id union for all study periods. Listing 6.2 gives a code sample on how this constraint is implemented.
//1. define finite domain store
Store store = new Store();

//2. define and initialize finite domain variables
int size = 3;
IntVar[] v = new IntVar[size];
for (int i=0; i<size; i++)
    v[i] = new IntVar(store, "v"+i, 1, size);

//3. define and impose constraints
store.impose( new XneqY(v[0], v[1]) );
store.impose( new XneqY(v[0], v[2]) );
store.impose( new XneqY(v[1], v[2]) );

//4. search for a solution
Search<IntVar> search = new DepthFirstSearch<IntVar>();
SelectChoicePoint<IntVar> select =
    new InputOrderSelect<IntVar>(store, v,
        new IndomainMin<IntVar>());
boolean result = search.labeling(store, select);

Listing 6.1: Example Java code on using JaCoP

In Listing 6.2, each study period is represented by an integer SetVar. All the courses are represented with an integer value. The sample code imposes the union constraint on these integer set variables.

In Appendix C, the sample code shows how this project imposes all the constraints in the CSP model.
public static SetVar imposeAndGetUnion(
    Store store,
    SetVar[] periodCourseIdVars,
    Set<Integer> interestedCourseIdSet) {

    LOGGER.debug("Posting constraints on the union of all course ids.");

    if (periodCourseIdVars == null || periodCourseIdVars.length == 0) {
        throw new IllegalArgumentException("Array must be non-empty!");
    }

    int minCourseId = Collections.min(interestedCourseIdSet);
    int maxCourseId = Collections.max(interestedCourseIdSet);

    SetVar union = periodCourseIdVars[0];

    for (int i = 1; i < periodCourseIdVars.length; i++) {
        SetVar partUnion = new SetVar(store,
           "part_union_" + i,
           new BoundSetDomain(minCourseId, maxCourseId));

        store.impose(new AunionBeqC(union, periodCourseIdVars[i],
            partUnion));
        union = partUnion;
    }

    return union;
}

Listing 6.2: Code sample on imposing union constraint
Figure 6.3: Code structure of the CSP solver
Chapter 7

Additional Implementation Details and Test Results

So far, this report has covered the topics of recommender systems, the semantic web and constraint programming. In detail, it has discussed: a) how candidate courses are generated through frequent pattern mining with FP-Growth and computing discipline deduction with semantic web technology; b) how these candidate courses are used to solve the CSP model to generate the final recommendations.

This chapter will discuss additional implementation details, such as the data models, Java projects structure, etc. It will also discuss the test results of the recommender.

7.1 Modular Design

Modular design is essential for maintainable software. I learned this practice from both Unix philosophy [73] and my work experience. This project tries to apply this practice in its implementation.

Figure 7.1 presents the architecture overview of the recommender. As the figure shows, the goal of modular design is reached by organizing different functionalities into separate modules. There modules are Apache Maven [3] Java projects. Figure 7.2 shows the dependency relationship between these modules.

When presenting different functionality modules, Figure 7.1 also shows the corresponding technologies used in these modules. For example, the Persistence module uses Spring Data JPA [50] to access the database.

7.1.1 Description of Modules

This section will describe the responsibilities of the modules.

- **CourseRecommenderParent**: parent Maven project. It specifies the
shared dependency libraries. For example, it specifies the versions of Spring framework, JUnit, Logback, etc.

- **CourseRecommenderService**: the RESTful web service implementation project. It exposes the recommender as a RESTful web service. As a result, the recommender can be integrated with other systems.

- **CourseRecommenderConstraintSolver**: the CSP model implementation project. It generates the final recommendations by finding solutions to the CSP model.

- **CourseRecommenderDomainReasoner**: the computing discipline deduction implementation project. With given input, it returns related computing disciplines through a semantic web SPARQL query.

- **CourseRecommenderRuleMiner**: the frequent pattern mining implementation project. It implements the FP-Growth algorithm to generate frequent patterns.

- **CourseRecommenderPersistence**: the database access implementation project. It works as a bridge between the service layer and the database.
• **CourseRecommenderDataLoader**: the original anonymous course selection data from senior students was kindly provided by Uppsala University’s IT department as printed documents. These documents were scanned and the data in these scanned documents were then recognized with an OCR (optical character recognition) tool. These data were parsed and imported to the database by this module.

• **CourseRecommenderAPI**: this module provides global data types used by different modules. It also provides Java service clients, which invoke the RESTful web service. These service clients can be used by other applications to integrate with the recommender.

### 7.1.2 Database Design

One important source for understanding a software product is to look at its data model. This section presents the database design ER-diagram (entity-relationship) of the recommender in Figure 7.3.

The following list describes the tables in the ER-diagram:

- **course**: table for all the courses that have been either taught or scheduled.

- **computing_domain**: table for all the computing disciplines, as specified in the SKOS file from ACM.

- **course_domain_relevance**: table for the mapping information between courses and computing disciplines.

- **supported_course_credit**: table for supported course credit, e.g., 5, 7.5, etc.
• **supported_course_level**: table for supported course level, e.g., basic, advanced.

• **course_selection_original**: table for the original course selection data.

• **course_normalization**: table to store the course normalization information. For example, if a course from a previous year is divided into two courses in a later year, then such information is stored in this table.

  The recommender uses an integer value to identify a course, instead of its code. That is because the same course can be scheduled in different periods. So the normalization also covers the course code to course id relationship.

• **course_selection_normalized**: table to store the normalized data of the ones from course_selection_original and course_normalization.

### 7.2 Tools and Their Uses

This section discusses the tools mentioned in Figure 7.1. It will explain the uses of these tools in this project:

• **Apache Maven**: used for Java project build and dependency management.

• **Git**: used for version control of Java source code and this report. Git and its related solutions, such as GitHub, are main stream version control.
system in IT industry. As a distributed version control system, it allows a user to commit changes to a local repository and synchronize with a remote repository when needed. The source code of this project is published as an open source project at GitHub: https://github.com/yong-at-git/CourseRecommenderParent

- **Jersey**: used as RESTful web service container.

- **Apache Tomcat**: used as web server to host Jersey and the RESTful web service.

- **Swagger framework**: The Swagger framework [51] gives a convenient way to document a RESTful web service. For example, Swagger annotations can be used as ordinary Java annotations to describe a Java RESTful resource class. When a Java RESTful resource is annotated with Swagger annotations and deployed, then Swagger UI can be used to display the documentation of the service in a web browser. The Swagger UI can also be used to invoke the service endpoints, as shown in Figure 7.4.

- **Spring Framework**: used for dependency injection. By using dependency injection, the resource management is taken care of by Spring Framework. For example, when using a Spring data JPA repository bean to access data, the developer does not need to care about database transactions (namely, the sequences of operations performed as a single logical unit of work) and connections (from the application to the database) because they can be managed by Spring.

- **Spring Data JPA (Java Persistence API)**: it is one solution from Spring for database access. After being properly configured, only JPA entity classes (i.e., mirror classes to database tables) and repository classes (i.e., interfaces to retrieve data) are needed to implement a functional JPA data access layer.

- **JUnit**: the framework for unit testing.

- **Google Guava**: a Java caching solution from Google. Instead of computing the result every time when there is a request, caching stores the computed data. When the same input is provided by a request, a cached result will be returned. This is an effective way to improve performance.

In this project, after the FP-tree is built with the complete course selection data from the database, this tree is cached. When another recommendation request comes, this tree is reused. By doing this, it avoids database access and the expensive process to build an FP-tree. The result in Section 7.4 shows that the use of caching increases performance significantly.

- **Java 8**: it is the Java version used in this project. The stream feature in processing data collections is heavily used.

- **MySQL**: it is the database server for this project.

52
• **SurveyMonkey:** it is the on-line survey service used to conduct the course selection best practices survey in this project.

### 7.3 Example Preference Input and Recommendation Output

To verify the recommender, some example courses and computing disciplines are provided as a user’s preference inputs. The example courses are listed in Table 7.1 and the example computing disciplines are listed in Table 7.2. In addition to user inputs, a mandatory course listed in Table 7.3 is also proved as an input. These inputs are sent to the recommender’s RESTful web service through the Java service client from **CourseRecommenderAPI**. The corresponding result is listed in Table 7.4. Section 7.4.2 will examine the example recommendation.
<table>
<thead>
<tr>
<th>Course code</th>
<th>Course name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1DL210</td>
<td>Algorithms and Data Structures I</td>
</tr>
<tr>
<td>1DL301</td>
<td>Database Design I, 5 hp</td>
</tr>
<tr>
<td>1DL340</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>1DL360</td>
<td>Data Mining I</td>
</tr>
<tr>
<td>1DL400</td>
<td>Database Design II</td>
</tr>
<tr>
<td>1DL441</td>
<td>Combinatorial Optimisation using Constraint Pro-</td>
</tr>
<tr>
<td></td>
<td>gramming</td>
</tr>
<tr>
<td>1DL450</td>
<td>Advanced Functional Programming</td>
</tr>
<tr>
<td>1DL600</td>
<td>Software Testing and Maintenance, 10 hp</td>
</tr>
<tr>
<td>1MD016</td>
<td>Human Computer Interaction</td>
</tr>
<tr>
<td>1TD186</td>
<td>Computational Finance - Pricing and Valuation, 5</td>
</tr>
<tr>
<td>1TD480</td>
<td>Programming of Parallel Computers</td>
</tr>
</tbody>
</table>

Table 7.1: Example list of preferred courses

<table>
<thead>
<tr>
<th>Discipline ID</th>
<th>Discipline name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10002953</td>
<td>Database design and models</td>
</tr>
<tr>
<td>10003121</td>
<td>Human computer interaction (HCI)</td>
</tr>
<tr>
<td>10010257</td>
<td>Machine learning</td>
</tr>
<tr>
<td>10011074</td>
<td>Software creation and management</td>
</tr>
<tr>
<td>10011076</td>
<td>Requirements analysis</td>
</tr>
</tbody>
</table>

Table 7.2: Example list of preferred computing disciplines

<table>
<thead>
<tr>
<th>Course code</th>
<th>Course details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1DT032</td>
<td>name=Advanced Computer Science Studies in Sweden, 5hp, taughtYear=2015, startPeriod=1, endPeriod=1, credit=5.0, level=ADVANCED</td>
</tr>
</tbody>
</table>

Table 7.3: Example list of mandatory courses
<table>
<thead>
<tr>
<th>Recommendation summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total recommended courses: 16</td>
</tr>
<tr>
<td>Total recommended courses credits: 95.0</td>
</tr>
<tr>
<td>Total recommended ADVANCED courses: 11</td>
</tr>
<tr>
<td>Total recommended ADVANCED courses credits: 70.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>2015 Period 1</strong></td>
</tr>
<tr>
<td>code=1DT052</td>
</tr>
<tr>
<td>code=1DT032</td>
</tr>
<tr>
<td>code=1TD046</td>
</tr>
</tbody>
</table>

| **2015 Period 2** |
| code=1DT074 | name=Computer networks II, 10 hp, taughtYear=2015, startPeriod=2, endPeriod=3, credit=10.0, level=ADVANCED |
| code=1DL301 | name=Database Design I, 5 hp, taughtYear=2015, startPeriod=2, endPeriod=2, credit=5.0, level=BASIC |
| code=1TD396 | name=Computer assisted image analysis I, 5 hp, taughtYear=2015, startPeriod=2, endPeriod=2, credit=5.0, level=ADVANCED |

| **2016 Period 3(Spring semester)** |
| code=1MD030 | name=Medical Informatics, 5 hp, taughtYear=2016, startPeriod=3, endPeriod=3, credit=5.0, level=ADVANCED |
| code=1TD480 | name=Programming of parallel computers, 10 hp, taughtYear=2016, startPeriod=3, endPeriod=3, credit=10.0, level=ADVANCED |

| **2016 Period 4(Spring semester)** |
| code=1DT082 | name=Computer networks III, 5 hp, taughtYear=2016, startPeriod=4, endPeriod=4, credit=5.0, level=ADVANCED |
| code=1TD204 | name=Software architecture with Java, 5 hp, taughtYear=2016, startPeriod=4, endPeriod=4, credit=5.0, level=ADVANCED |

| **2016 Period 1(Autumn semester)** |
7.4 Performance Metrics

This section discusses some performance metrics of the recommender. It starts with examining the feedback from the user evaluation.

7.4.1 User Evaluation

In total, there were three users participating in the user evaluation of the recommender. One user has already finished his study and the other two are finishing their thesis projects.

The user evaluation is performed in this way: I collected the preferred course list and computing discipline list from the users, generated the recommendations based on the collected inputs and then asked the ratings from the users about how much the recommendations satisfied their preferences. The rating has a range from 1 to 5, with 5 meaning most satisfied.

Fortunately, the result showed the users were quite satisfied with the recommendations. The average rating from them was above 4 out of 5.

Appendix D lists some feedback from the users.
7.4.2 Correctness of the Recommendations

When looking at the example generated recommendation in Table 7.4, it is easy to tell that the recommendation meets the constraints from the CSP model. For example, the summary in the table says the total recommended course credits is 95, which is not below the required minimum value of 90.

The recommendation is also relevant to the input preferences. For example, the input specifies HCI as preferred computing discipline and the recommendation covers two relevant courses, i.e., “Interface Programming with a User Perspective” and “Human Computer Interaction”. The recommendation also includes courses for the inputs parallel programming, computational finance, etc.

7.4.3 Runtime Performance

This section discusses different aspects of the recommender’s runtime performance. It starts with describing the data size of the recommender.

Data Size

In total, there are around 100 unique candidate courses for the six study periods. The recommender assumes the first and second study periods of 2016 have the same scheduled courses as the same periods in 2015 respectively.

In total, there are 3426 original anonymous course selection records. Each record represents a student has attended a course. Since some courses later on split into several courses or some courses used a different code, these selection records are normalized. The recommender uses an integer value to identify a course, instead of its code. That is because the same course can be scheduled in different periods. So the normalization also covers the course code to course id relationship. There are 6349 normalized records in total.

These anonymous records are from 267 students. If considering each student’s complete course selection history as a transaction, then the database contains 267 transactions.

Environment Configuration

The test is performed on a laptop. The following is the configuration of the laptop:

- Processor: Intel® Core™ i7-4600U CPU @ 2.10GHz 2.70GHz
- Installed memory (RAM): 8.00 GB (7.90 GB usable)
- System type: 64-bit Operating System

The Runtime

The runtime was measured with the example inputs as in Tables 7.1, 7.2 and 7.3. The recommender was configured to generate ten recommendations for each request.
The same request was sent to the service twice by the service client. The first request was sent when the service is freshly deployed and the second one was sent after the first request got the response. The purpose of this context setup is to measure the performance gain from using caching. Table 7.5 shows the runtime details for the first request, and Table 7.6 shows the runtime details for the second request.
## Runtime details for the first request

<table>
<thead>
<tr>
<th>CSP model searching summary</th>
<th>Total CSP solution searching times:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Nodes: 31</td>
<td>11</td>
</tr>
<tr>
<td>• Decisions: 27</td>
<td></td>
</tr>
<tr>
<td>• Wrong Decisions: 4</td>
<td></td>
</tr>
<tr>
<td>• Backtracks: 6</td>
<td></td>
</tr>
<tr>
<td>• Max Depth: 21</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Distributed CSP solution searching times:</th>
<th>Total runtime:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• with explicitly specified courses: 1</td>
<td>23857 ms</td>
</tr>
<tr>
<td>• after enlarging candidates list with discipline deduction: 4</td>
<td></td>
</tr>
<tr>
<td>• after enlarging candidates list with frequent pattern mining: 6</td>
<td></td>
</tr>
</tbody>
</table>

| Total runtime: | 23857 ms       |
| Runtime with explicitly specified courses: | 1071 ms         |
| Runtime after enlarging the candidates list with discipline deduction: | 1. first retry: 868 ms |
| | 2. second retry: 770 ms |
| | 3. third retry: 756 ms |
| | 4. fourth retry: 720 ms |
Runtime after enlarging the candidates list with frequent pattern mining:

<table>
<thead>
<tr>
<th></th>
<th>1. first retry: 435 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. second retry: 418 ms</td>
</tr>
<tr>
<td></td>
<td>3. third retry: 449 ms</td>
</tr>
<tr>
<td></td>
<td>4. fourth retry: 422 ms</td>
</tr>
<tr>
<td></td>
<td>5. fifth retry: 450 ms</td>
</tr>
<tr>
<td></td>
<td>6. sixth retry: 1360 ms</td>
</tr>
</tbody>
</table>

Table 7.5: Runtime details of the first request
## Runtime details for the second request

| CSP model searching summary | • Nodes : 31  
|                            | • Decisions : 27  
|                            | • Wrong Decisions : 4  
|                            | • Backtracks : 6  
|                            | • Max Depth : 21  
| Total CSP solution searching times: | 11  
| Distributed CSP solution searching times: | • with explicitly specified courses: 1  
|                                            | • after enlarging candidates list with discipline deduction: 4  
|                                            | • after enlarging candidates list with frequent pattern mining: 6  
| Total runtime: | 8376 ms  
| Runtime with explicitly specified courses: | 392 ms  
| Runtime after enlarging the candidates list with discipline deduction: | 1. first retry: 483 ms  
|                                                           | 2. second retry: 527 ms  
|                                                           | 3. third retry: 474 ms  
|                                                           | 4. fourth retry: 415 ms  

Runtime after enlarging the candidates list with frequent pattern mining:

<table>
<thead>
<tr>
<th></th>
<th>1. first retry: 401 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. second retry: 406 ms</td>
</tr>
<tr>
<td></td>
<td>3. third retry: 419 ms</td>
</tr>
<tr>
<td></td>
<td>4. fourth retry: 440 ms</td>
</tr>
<tr>
<td></td>
<td>5. fifth retry: 416 ms</td>
</tr>
<tr>
<td></td>
<td>6. sixth retry: 1146 ms</td>
</tr>
</tbody>
</table>

Table 7.6: Runtime details of the second request

Discussion

As Table 7.5 and Table 7.6 show, the second request has significant performance gain over the first one, i.e., total runtime of 8376 ms vs 23857 ms. That is because of caching. When the FP-tree for all the course selection records was built for the first request, the data was read from database. But once it is built, it is cached in memory. When the second request was sent, this cached tree was returned and hence it gave a performance gain.
Chapter 8

Future Work

Even though the test result shows the recommender has met the basic design goals, there is still room for improvement. According to self-evaluation on the recommender and user feedback, some points are identified as areas for improvement. They are listed as follows:

- **Perform offline frequent pattern mining**: with the current implementation, the FP-tree is cached, but not the frequent patterns. The pattern mining process is performed when a request reaches the service. Since the course selection history data is relatively stable (updated every semester), the frequent patterns are also stable. So the frequent patterns can be generated and stored in the database in advance.
  
  That means the service does not need to call the expensive FP-Growth algorithm to generate a recommendation. Instead, it can just read frequent patterns from database and easily enlarge the candidate list when needed.
  
  This improvement is expected to reduce the process time of the recommender.

- **Introduce a rating of recommendations**: with the current implementation, a user can specify the number of recommendations but the recommendations are ordered randomly.
  
  It would be better to have a rating metric to decide the order of the recommendations. The most relevant recommendation should be listed first and so forth.
  
  This improvement is expected to increase the quality of the recommender.

- **Improve the accuracy of computing discipline to course mapping**: when implementing the recommender, I mapped the computing disciplines (according to ACM’s SKOS file) to the courses based on my own understanding. Hence, it has a high chance of lacking accuracy because of the broad scope of the disciplines and also because of my limited understanding of the courses and disciplines.
If this mapping can be reviewed by people with a better understanding of these disciplines and courses, the recommendation quality could also be improved.

- **Consider course dependency relationship:** as pointed out by one user, the course dependency relationship is not explicitly supported with the current implementation. When saying *explicitly*, it means the recommender indirectly considers course dependency to a certain extent. That is because of applying frequent pattern mining, which represents a collective decision from many users. In other words, if two courses have a dependency relationship, then many students must have attended them together and hence these two courses become a frequent pattern.

  But the limitations of the current implementation are:

  - it does not assure that all the items in the same frequent pattern will be included in the same recommendation;
  - it does not consider the order of the dependency, e.g., course A should be attended before course B.

  This improvement should be considered as a critical one.

- **Consider multi-period courses:** as one user pointed out, there are some courses taking more than one period to finish. The current implementation considers all courses as a single period course. Even though the constraints on total credits, single period credit, and the number of courses per period could mimic the effect of multi-period courses, it is still good to thoroughly analyse the impact from such courses.

- **Enlarge the amount of mining data:** the course selection data was retrieved more than four years ago. Since there are graduated students every year, there are more data available in the IT department. When the data is richer, it could help to improve the quality of frequent patterns and hence improve the quality of recommendations.

- **Tune the solution searching strategy of the CSP model:** in the current implementation, I only used the simple branching strategy when searching for CSP solutions. But JaCoP provides several strategies to assign values to each variable and select variable for a solution assignment from the array of variables [24]. It will be helpful to try out these strategies and optimize the implementation.

  The improvement is expected to reduce the process time of the recommender.

- **Collect more user evaluation feedback:** even though the users gave a high rating to the quality of the recommender, there were just three of them.
If more test users are involved, then the rating of the quality of the recommender can be more objective and further constructive feedback could be expected.

- **Provide recommendations at any stage of the study path:** the current implementation is designed to generate the study plan recommendation for the complete two-year programme. It is expected to be used when a student starts her study from the first study period.

The recommender should support generate recommendation at any stage of the study path, given it is provided with sufficient information. For example, a student should be able to use the recommender after she has finished her first semester’s study, given she tells the recommender what courses she has finished during her first semester’s study.

- **Improve test automation:** the test code has a low coverage in this project. It is partly because some methods are not easy to test, e.g., when searching for a CSP solution. Test automation is essential for building maintainable software.

With this improvement, the maintainability of the recommender is expected to be assured.
Chapter 9

Conclusion

In this project, I tried to apply what I have learned from my study in the Computer Science Master Programme in Uppsala University and my work experience to address an interesting real life problem. The problem is about facilitating the planning of course selection for students from the programme. The solution is to implement a course selection recommender. The applied knowledge in the solution covers recommender systems, the semantic web and constraint programming.

This report tries to give a thorough introduction to the relevant knowledge and explain how they are applied to the implementation of the recommender. It also examines the performance of the recommender, including performing a user test.

The preliminary result shows that the work has met its original expectation.
Bibliography


[21] Jiawei Han, Jian Pei and Yiwen Yin. ‘Mining frequent patterns without candidate generation’. In: ACM Sigmod Record. Vol. 29. 2. ACM. 2000, pp. 1–12.


[64] W3C. Semantic Web Health Care and Life Sciences (HCLS) Interest Group. URL: http://www.w3.org/blog/hcls/ (visited on 04/10/2015).
[67] W3C. W3C RIF documents list. URL: http://www.w3.org/TR/#tr_RIF_Rule_Interchange_Format (visited on 04/10/2015).
[69] Wikibooks. Data Mining Algorithms In R/Frequent Pattern Mining/The FP-Growth Algorithm. URL: https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Frequent_Pattern_Mining/The_FP-Growth_Algorithm#FP-Growth_Algorithm (visited on 02/10/2015).
## List of Algorithms

1. Amazon’s item-item CF algorithm ........................................ 21
2. Build an FP-Tree .............................................................. 27
3. Frequent pattern mining upon an FP-Tree ......................... 27
List of Figures

3.1 The flowchart of CSP modelling ........................................... 9
3.2 The flowchart of deducing preferred courses from the mappings of courses to computing disciplines .................................. 10
3.3 The flowchart of recommendation generation ............................ 11

4.1 Decision tree example ......................................................... 16
4.2 Pearson correlation example .................................................. 20
4.3 Cosine similarity example ..................................................... 21
4.4 Example FP-Tree ............................................................... 26

5.1 Semantic web layer cake (architecture design) - 2000 - Tim Berners-Lee ................................................................. 29
5.2 Semantic web layer cake (with standardized technologies) - 2009 - James A. Hendler ......................................................... 30
5.3 Example RDF graph ............................................................. 31
5.4 RDF triple ........................................................................... 33
5.5 Structure of OWL 2 ............................................................... 34

6.1 Example sudoku puzzle ......................................................... 38
6.2 Solution to the example sudoku puzzle of Figure 6.1 ................. 39
6.3 Code structure of the CSP solver .......................................... 47

7.1 Architecture, supporting technologies and tools ......................... 49
7.2 Maven dependency among major modules ................................. 50
7.3 Database design ER-diagram .................................................. 51
7.4 Swagger UI documentation of the RESTful service .................... 53
List of Listings

5.1 Example RDF/XML document ......................................... 32
5.2 Example RIF-Core rule .................................................. 34
5.3 Example SPARQL statement ............................................. 35
5.4 ACM computing classification system SKOS sample .......... 36
5.5 Sample SPARQL query statement from this project .......... 36
6.1 Example Java code on using JaCoP ................................. 45
6.2 Code sample on imposing union constraint .................... 46
B.1 Code sample on FP-Growth frequent pattern mining ......... 89
C.1 Code sample on imposing constraints in the CSP model ...... 91
List of Tables

2.1 Part of the survey on course selection planning .......................... 5
7.1 Example list of preferred courses ............................... 54
7.2 Example list of preferred computing disciplines .................. 54
7.3 Example list of mandatory courses ................................. 54
7.4 Example recommendation ........................... 56
7.5 Runtime details of the first request ............................ 60
7.6 Runtime details of the second request .......................... 62
A.1 The complete course selection best practices survey ............ 88
D.1 User evaluation feedback ........................................ 94
Appendices
Appendix A

The Complete Survey and Its Result
<table>
<thead>
<tr>
<th>Question</th>
<th>Result</th>
</tr>
</thead>
</table>
| **Q1:** Do you think that it is better starting with at most one more advanced course (in addition to Advanced CS Studies in Sweden) at the first period? | Answered: 32  
Yes: 68.75%  
Other opinions: 31.25%                                                                                                           |
| **Q2:** Do you think that around 15 credits per period is the most reasonable plan? | Answered: 32  
Yes: 87.50%  
Other opinions: 12.50%                                                                                                           |
| **Q3:** Do you think that knowing your preference/specialization in certain CS sub-fields is important and relatively easy for you to select courses? | Answered: 32  
Yes: 84.38%  
Other opinions: 15.62%                                                                                                           |
| **Q4:** Do you think doing your master thesis in the last semester is the best choice? | Answered: 32  
Yes: 81.25%  
Other opinions: 18.75%                                                                                                           |
| **Q5:** What problems have you encountered while selecting courses? | Answered: 29, Skipped: 3  
65.52% Voted for: "Not clear about your preference/specialization among CS subfields."  
Other opinions: 34.48%                                                                                                           |
| **Q6:** What additional information do you want to get while making your course plan? | Answered: 28, Skipped: 4  
85.72% Voted for: "More detailed information about courses, such as which CS sub-field do a certain course belong to."  
Other opinions: 21.43%  
Note: some respondents have checked both options                                                                                     |
| **Q7:** From which way do you think your course plan has affected your study outcome? | Answered: 28, Skipped: 4  
85.71% Voted for: "Workload imbalance, e.g. too much workload in certain period and consequently lead to poor performance"  
Other opinions: 17.85%  
Note: some respondents have checked both options                                                                                     |
Q8(Part 1/5): If you have any other opinions on above questions, please write them down in below box. (Due to limitation of this online service as a free one, I have to design the survey this way. Sorry for any inconvenience.)

- It's not clear how to apply to courses in departments other than CS, as well, finding relevant courses in those departments. For instance selecting mathematics courses, or humanities courses (say - improving English writing skills).

- Need more formal methods courses. I find in some periods there are many courses to take and in others not many. Also there are plenty of math courses related to logic that should be promoted more. I am very upset I did not take more of Erik Palmgren courses as they were not advertised by the IT department. Theoretical computer science seems to be neglected quite a bit in Uppsala.

- Sometimes I do not find relative courses to my field in each period.

- Not really knowing what the course is going to be about from the title. The meeting before each semester helps a little, but it would be great to listen to old student’s opinions.
It is really important to choose a topic which is your speciality for thesis. Some students (including me) may think that doing a thesis in a fresh and strange topic may benefit you a lot, because you learn so many new things which may help you in getting your future career. But from my experience, it is not wise to do like that. I have been doing my master’s thesis on a very strange topic, it took me almost 4 months to get to know about the topic and get started with the project in a real sense, and during those 4 months, I could say I felt like staying in hell, it was very discouraging and disappointing, my passion for thesis and research was long gone. Even after I had enough knowledge and background to do the thesis project, the thesis was not fun for me any more, it became an extremely stressful burden and nightmare, because I already spent too much time to get started, I had very little time to do the real work which was very stressful, even prohibiting. And I had to prolong the period of my thesis project for another 2 months. Now, I am about to finish my thesis but the time-limit, stressful experience and uncomfortable situation have lowered the quality of my thesis and the software I wrote. The worse thing is that I had a very unpleasant collaboration with my supervisor because of my slow progress, he even refused to write me a recommendation letter not to mention a PhD. So my fellow friends should really learn something from my experience, you can put or use my tragic story anywhere :p
Q8(Part 3/5):

- 1) Sure, starting with more than the CS-swe course is good, but personally I do not think courses should be forced on to students. I might have misunderstood the question and in that case I totally agree, a master should have advanced courses.

- 3) It is far from easy to select courses even if you "know" what you like. In most cases what you think is an interesting subfield may turn out not to be what you expected. There are a lot of courses at the university and almost all of them sound interesting to all students. I have not met one CS student that is only interested in one "sub-field".

- 6) See 9.

- 7) Personal stuff.

- The problem is the scheduling that some courses are conflict, thus it is sometimes difficult to select among courses.

- regarding question no 1: I really hated the course, advanced studies in computer science, it was really a waste of time, so no more similar course.
Q8(Part 4/5):

- Sometimes, you know that what field is your interest, but having 15 credits in one period is really complex, because you do not have enough time to focus on learning new methods.

- 1. In my opinion, master thesis can be started at any time in second year, not the last semester. 2. The courses were not descriptive enough to select. In my opinion, the related courses should be grouped so that it will be easy to select according to our field of interest.

- 1) Lack of courses in software development. 2) No courses in testing, except the grund 1. 3) No courses related to mobile applications. 4) No course in game development during regular semester, except grund and summer 1. 5) Very limited number of courses which are interesting personally for me, especially for autumn semester. Courses cover mostly from networks area + math + HCI. Big gaps of courses related to another areas. 6. I want to be sure that teacher stated in schedule is really teacher, who teach the course. I do not want choose a courses for period and that face that one of them is taught by professor’s PhD student. 7. Dropped course due to not sufficient information about it at the beginning or presentation of the course.
| Q8(Part 5/5): | • I want to see which courses have overlapping lectures and how many.  
• I think the course Advance CS in Sweden itself is OK, but it should not be mandatory but highly recommended for beginners. 15 credits, yes that would be fine. But if a student wants to take like 20 credits there should not be any restrictions binding on him. It’s the individual’s concerns. Of course the teacher has to provide them their valuable suggestions to this. To do a Master thesis, the minimum requirement is 60 credits. So I do not think it is relevant to do their thesis the last semester. It depends on the individual, if he get a read nice one he can start earlier also.  
• 5. There is not a clear information about the aim of the course, objectives, few support during the laboratories. 6. The ones I wrote In question No. 5 7. When there is unclear laboratories, homeworks or projects and few support of the teachers then it gets very stressful and I get low motivated to continue with it. |
Q9(Part 1/3):
What else best practice will you recommend concerning course selection?

- the course pages must be available online before the course starts.
- Workload is, I think, very important, there should be no suggestion for selecting more than two, known to be, hard courses; not to exceed 30 credits.
- Have a specific stream for formal methods/computer languages. Uppsala is so strong in this area yet the masters does not reflect that.
- 2 courses at a period is the best, I strongly recommend my fellow CSers not to be too ambitious about credits in a semester. Quality is much more important than the quantity when taking courses. To my fellows, if you want to have a fulfilling and happy experience at UU, choose appropriate amount of courses at a period and get the best out of them :)
- Talk to other students! This might be hard but it is something that can give you a whole different opinion about a course/teacher. Most of the time it’s hard to know in beforehand if the course is something you really want to do. Again, talk to other students.
- My main strategy was to pick a teacher, not a subject. Because bad teacher can ruin the most interesting subject. So recommender should give more info about teachers, rather than "official course info", that might fail to materialise.
Q9(Part 2/3):

- it can be good when every general major like, artificial intelligence have a special course plane like exactly things which network has at beginning of the each semester.

- Be realistic

- Talk to other students and finding out if the teacher is good. This is almost the only factor that I feel affects the outcome of the course. Of course the material is important too, but most bad courses I have taken is all due to the teacher.

- 1) Some suggestions like when we select one course, then we must know better next course. Like when student chooses the course Human Computer Interaction, the next better course would be like website construction etc.

- For students - to consult with senior students. Some courses descriptions are so tricky that you can't make a true impression about them. Senior fellows, who studied that courses already can help a lot. Of course, their opinion is subjective, but the quality of teaching, the amount of workload and usefulness of course they can detect very precisely.
Q9(Part 3/3):

- The courses during the second year, can be changed because, it will be the same structure when the individual would be in his first year. This would make us to go for the same courses. And also, the courses should be within a period 1 & 2 and period 3 & 4 NOT between the other way. That will not be useful as a student to select his courses. That is what I feel.

  The courses should also be more evenly balanced between the 4 semesters.

- Small projects in groups of 3, but individual testing when evaluating design or the methodologies.

- should make the feedback of the course available to the future students. If that is not possible then at least the rating that was given in the feedback for the previous year.

- Create your own profile by course selection.
Q10(Part 1/2):
What would you expect from a course selection recommender system to help students make wise course plan?

• Such detail as course timetable should be taken into account. There is no point in picking courses which have a lot of clashes. Also, I do not think that selection recommender should be overly smart at suggesting things for not to exclude good courses even though they may be not in line with other selections.

• Tell them what subjects are suited to them

• anonymous notes about the course, teacher. summarized information of the courses..

• attention to the course which we get, can recommend courses with good description

• Recommend courses based on a short form that the student fills, like his preferences, previous studies, strength points...etc

• Give example of chunks of courses that are good too make, and then you can fine-tune by removing and adding one or two.

• yes, perfectly what I stated above. It would be great idea. And it would help to become specialised with a field of interest.
• Good idea, as on my mind, is to make a courses division by specializations. It will help students to orient, and by the way, will show the lack of courses in many areas.

• If a such a system was there, the main objective of that would be, to help students, to focus on their carrier paths, and choose the course accordingly, so that they are more focused towards their goal rather than just taking all courses and then deciding on the goal. In one sentence, Decide on a goal and then go for the courses which would lead to that goal rather the other way.

• I think the design of the websites with the course information is too much and too complicated sometimes to understand. Making more attractive and simple the information will be easy to make a best selection.

• More often students are not sure of what they want to do in their masters. So in addition to the plans that the student selects there should be a section with most popular plans and also what all course plans will lead to what kind of specialization.

• I have no idea
Appendix B

FP-Growth Frequent Pattern Mining Sample Code

```java
/* Frequent pattern mining with FP-Growth */
private Map<Set<Integer>, Integer> miningWithFPGrowth(FPTree fpTree, Set<Integer> suffixPattern) {
    Map<Set<Integer>, Integer> frequentPatternFromSinglePrefixPath = new HashMap<>();

    FPTree branchingTree = fpTree;

    /* Mining on single-prefix path */
    if (fpTree.hasSinglePrefixPath()) {
        List<Item> singlePrefixPath = fpTree.getSinglePrefixPathInTopDownOrder();
        Map<Set<Integer>, Integer> frequentPatternWithinSinglePrefixPath = getFrequentPatternFromSinglePrefixPath(singlePrefixPath);

        frequentPatternFromSinglePrefixPath = frequentPatternWithinSinglePrefixPath.entrySet()
            .stream()
            .collect(Collectors.toMap(
                entry -> {
                    Set<Integer> existingPattern = new HashSet<>(entry.getKey());
                    existingPattern.addAll(suffixPattern);
                    return existingPattern;
                },
                entry -> entry.getValue()
            ));
    }

    return frequentPatternFromSinglePrefixPath;
}
```

89
branchingTree = fpTree.getBranchingTree();

if (branchingTree == null) {
    return frequentPatternFromSinglePrefixPath;
}

/* Mining on branching tree */
Map<Set<Integer>, Integer> frequentPatternFromBranchingTree = new HashMap<>();
List<HeaderTableItem> headerList = branchingTree.getHeaderTable();
ListIterator<HeaderTableItem> itr = headerList.listIterator(headerList.size());

while (itr.hasNext()) {
    HeaderTableItem visitingItem = itr.next();

    Set<Integer> newPattern = new HashSet<>(suffixPattern);
    newPattern.add(visitingItem.getItem().getId());

    frequentPatternFromBranchingTree.put(newPattern,
        visitingItem.getItem().getCount());

    visitingItem.getItem().getCount();

    Map<List<Integer>, Integer> patternBase = FPTreeUtil.getPatternBase(visitingItem);

    FPTree conditionalTree = FPTreeBuilder.
        .buildConditionalFPTree(patternBase, this.minSupport);

    Boolean hasChild = !conditionalTree.getRoot().
        .getChildren().isEmpty();
    if (conditionalTree != null && hasChild) {
        frequentPatternFromBranchingTree.
            putAll(miningWithFPGrowth(conditionalTree, newPattern));
    }
}

/* Gets union and cross product. */
return consolidatePatterns(frequentPatternFromSinglePrefixPath,
    frequentPatternFromBranchingTree);

Listing B.1: Code sample on FP-Growth frequent pattern mining
Appendix C

CSP Model Imposing Constraints Sample Code

```java
@Component
public class Modeler {

    private static final Logger LOGGER = LoggerFactory.getLogger(Modeler.class);

    private ModelConfig config;

    public void postConstraints(Store store, SetVar[] pers, ModelConfig config) {
        LOGGER.debug("Posting constraints!");
        this.config = config;

        // 1. on cardinality
        SinglePeriodCourseCardinalityConstraint.impose(store, pers);
        TotalCourseCardinalityConstraint.impose(
            store,
            pers,
            config.getMaxCourseAmount());

        // 2. on the must-have ones
        FixedCourseSelectionConstraint.impose(
            store,
            pers,
            this.config.getPeriodIdxToMustHaveCourseId());

        SetVar allCourseIdUnion = AllPeriodsCourseIdUnionConstraint
```
imposeAndGetUnion(
    store,
    pers,
    this.config.getInterestedCourseIdCollection());

// 3. avoid selecting same course
AvoidSameCourseConstraint.impose(
    store,
    allCourseIdUnion,
    this.config.getAvoidCollectionForCourseWithDiffIdSet());

// 4. on credits
AdvancedCreditsConstraint.impose(
    store,
    allCourseIdUnion,
    this.config.getCreditToAdvancedId(),
    this.config.getMaxAdvancedCredits());

SingleStudyPeriodCreditsConstraint.impose(
    store,
    pers,
    this.config.getCreditToCourseId());

TotalCreditsConstraint.impose(
    store,
    allCourseIdUnion,
    this.config.getCreditToCourseId(),
    this.config.getMaxTotalCredits());
}
}

Listing C.1: Code sample on imposing constraints in the CSP model
Appendix D

User Evaluation Feedback

Note that user replies are given unedited and may contain spelling or grammar errors.
User A

This is a nice recommendation and some of the courses I actually took are in it. I have some comments though:

1. There are some constraints which I think you should take into consideration. For example “Networks II” extends into the thesis period when you might not want it to.

2. “Database Design II” is taken before “Database Design I”. This might not even be possible if somebody has no clue about DBMS.

The ratings:

- 2015 Autumn Semester ==> 4/5, I just think “Compiler Design” is not a good recommendation.
- 2016 Spring Semester ==> 5/5, Actually these are the courses I took.
- 2016 Autumn Semester ==> 3/5, “Real Time Systems”; I was told that it is more in the embedded systems realm so there is not much to learn (from my perspective). “Human Computer Interaction” I really see no point in this recommendation. I do not know if you have included the “Project DV” in your system but you should. This is THE course to take in this Masters programme.

User B

I would give 5 to the “correctness” and quality of the recommendation...

User C

I will give my rank 4 out of 5.

Table D.1: User evaluation feedback