Project Documentation

Studyplan Visualization

Christopher Chiche
Julien Perrochet

CRAFT
EPFL

January 21, 2013

Supervisor
Pierre Dillenbourg
CRAFT – EPFL
CH-1015, Lausanne

Table of Contents

Chapter 1

Introduction

1.1 About

This project's original motivation was to explore the study plans proposed by each section based on actual student choices – analysing what courses students took – instead of the standard course prerequisites and descriptions that are generally available, and to offer a tool to allow anybody to take a look at the data for himself.

To do so required two important steps:

- Represent the students study history with a useful model;
- Find good visual representations for the available data.

Both steps were achieved and solid foundations were lain to enable the implementation of new visualisations.

1.2 Project Directory Structure

We use sbt to build the code, and hence stick to its structure. Furthermore, the `view`, `core` and `import` segments of the application each live in their own sbt sub-project.

The relevant directories are listed hereafter:

```
core/
    src/main/
        scala/
            ch/epfl/recom
                graph/
                model/
                    administration/
                processing/
                    maps/
                        assist/
                storage/
                    db/
                    maps/
                util/
        plpgsql/
        sqlview/
import/
    src/main/scala/
        ch/epfl/craft/recom/dimport/isa/
view/
    src/main/scala/
        ch/epfl/craft/recom/
            view/
                model/
                snippet/
                    draw/
                util/
                view/
```

Chapter 2

Model

The model is built around the students and their course history, and this is reflected at the database level. Whenever going through the model is useless (most of the cases that were implemented), the data is accessed and processed directly through stored procedures.
For further in-depth data-mining where individual course histories must be analyzed, we believe the model will prove useful.

2.1 Classes

A few of the main classes are mentioned hereafter. They are meant to give you a glimpse of the model: for any of the classes referenced within, please refer to the project's Scaladoc[1]. and to the source code[2].

```
class Topic(
    val id: String,
    val name: String,
    val section: Section,
    val prerequisites_id:
        Set[Topic.TopicID],
    val description:
        Option[String],
    val credits: Option[Int]
)
```

```
class Course(
    id: String,
    name: String,
    section: Section,
    prerequisites_id: Set[Topic.TopicID],
    description: Option[String],
    credits: Option[Int],
    val semester: Semester,
    val head: Head
) extends Topic
```

```
class Student(
    val id: Student.StudentID,
    val arrival: AcademicSemester,
    val section: Option[Section],
    val currentSemester:
        Option[AcademicSemester],
    val semesterHistory:
        Set[AcademicSemester],
    val courses: Set[TakenCourse]
)
```

```
case class TakenCourse(course: Course,
    count: Int,
    grade: Option[Int],
    evaluation: Option[Int],
    semester: AcademicSemester
)
```

[1]http://shastick.github.com/orgarec/doc/core/#package
[2]https://github.com/Shastick/orgarec

2.2 Database Structure

The storage layer and structure are mainly defined through Lift's Mapper. Only for the processing and data extraction have stored procedures been written, and tables added manually. Figure 2.1 shows the database schema.
To avoid confusion with class names, the database entities representing objects are suffixed with `map`: a `Student` object is stored in the `studentmap` table.

2.2.1 Stored Procedures

The following stored procedures are already available:

computeCostudents Populates the `courserelationmap` table with the co-students count between any two courses.

sectionPerTopicDetail Gets details about section participation in each topic given by a section and during which academic level.
Example: `SELECT sectionPerTopicDetail` ('SHS'::varchar[],'MA1,MA3'::varchar[], '2012-07-01','2012-07-01')
Returns: the topic isa_id and name, the section participating in this topic (NOT the one teaching it), the average participating students from the section and the total number of students in this topic. these two last numbers are averaged over the number of courses that were considered in the query (if a span greater that 1 semester is provided, several occurences of the same topic will be counted.) students not in the specified academic leves (here: MA1, MA3) are not counted.

sectionTopics Returns the topics teached by the specified sections
usage: `sectionTopics('{IN,SC}'::varchar[])`

sectionTopicsWStudentCount Return the topics teached by the specified sections, and the quantity of students that attended during the specified period.
usage: `sectionTopics('{IN,SC}'::varchar[],'2011-07-01','2012-07-01')`

topicCostudentsSemFilter get the costudents quantity between any two topics taught by the specified sections during the specified time interval and at the specified academic level.
usage : select `topicCostudents`('SC,IN'::varchar[],'BA1'::varchar[], '2011-07-01','2012-07-01)

topicCostudents Get the costudents quantity between two topics taught by the specified section during the specified time interval
usage : select topicCostudents('SC,IN'::varchar[],'2011-07-01','2012-07-01)

topicSectionRatio Returns the sections present in this course and the corresponding ratio. The ratio is averaged over the number of courses that occurred in the given time interval.
usage: select `topicsectionratio(<isa_id>,'2011-07-01','2012-07-01)`

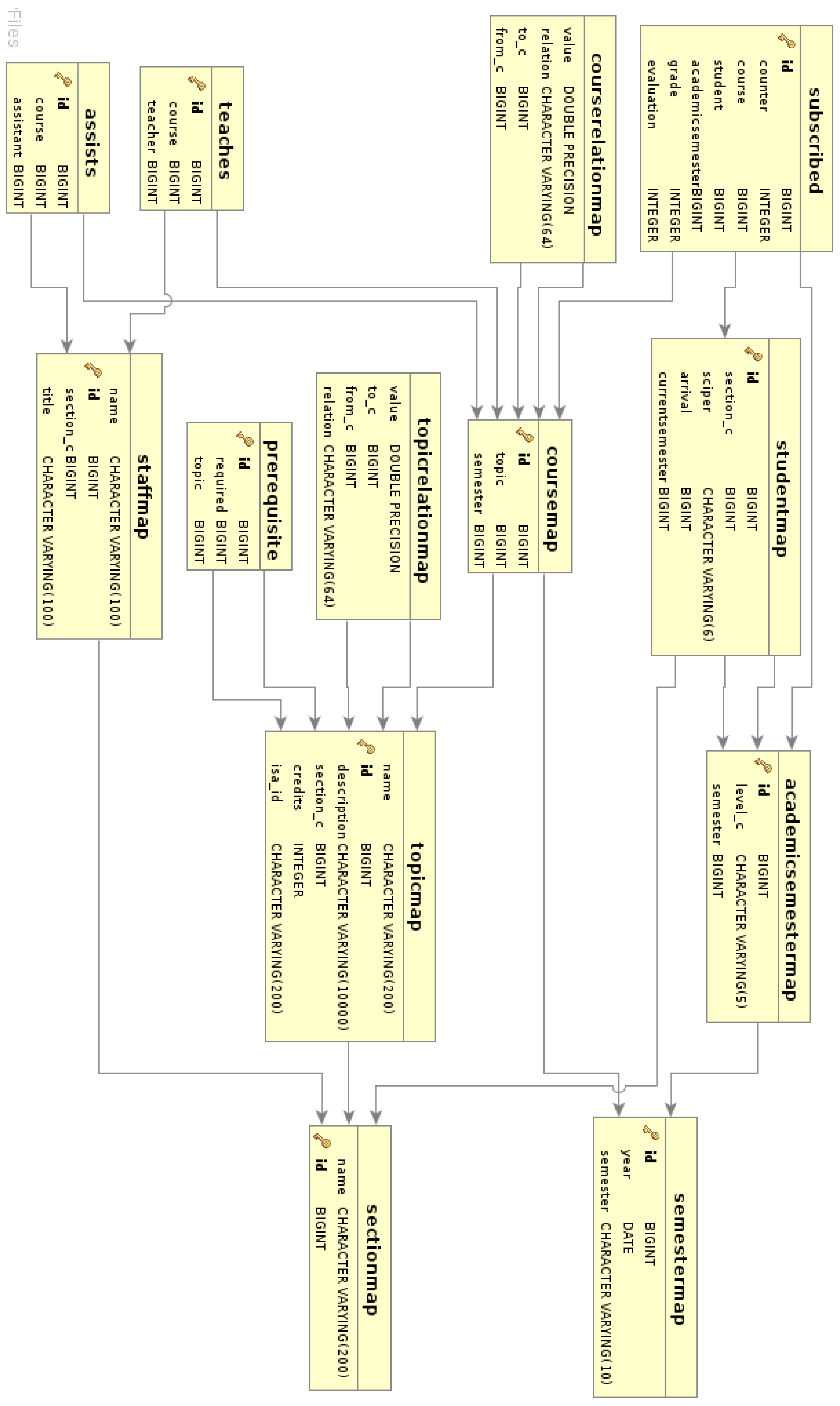


Figure 2.1: database schema

Chapter 3

Visualisation

3.1 Technology

The web application and data serving use Lift, which is also Scala based, enabling an easy integration with the core.
JavaScript is extensively used for the actual browser-side visualization. The D3.js[1] library provides advanced and complete visualization tools.

3.2 Implemented visualisations

The final version of our project includes two different visualization types. The first one, which was the primary objective of the project, represents a force directed graph[2] of topics which are linked by their relations (co-students ratio for now). The second type of representation, which was thought to be useful to explore SHS courses, helps representing the repartition of students in courses given their section using a Sankey diagram[3].

3.2.1 Organic graph of courses

This visualization is meant to easily visualize the relationship between various courses and the study plan. In this project we only use the percentage of students taking two courses to evaluate their distance, the more co-students they have, the closer they will be.

Key principles In this visualization, courses are represented as nodes in a graph, while their relations are represented as links between the nodes.
Here are the parameters we represent and how we represent them:

Credits: size of the node ;

Costudents (The percentage of students following two classes): length and width of the link.

The settings tab enables to choose easily which study plans we want to represent using the following parameters:

Sections: the sections from which we want to show the courses;

Semesters: the time range we want to consider;

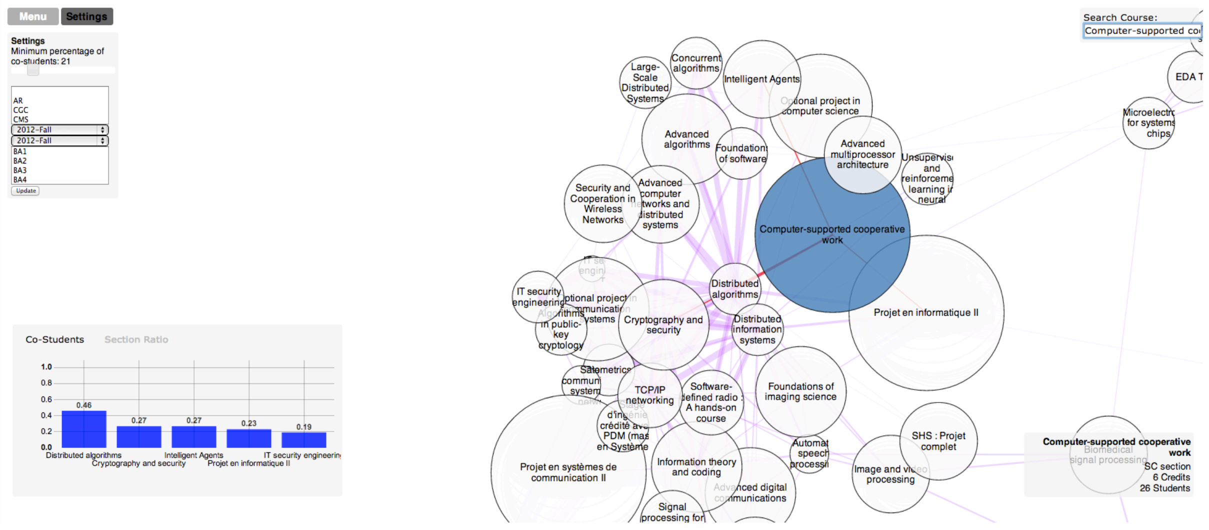
[1]www.d3js.org
[2]https://github.com/mbostock/d3/wiki/Force-Layout
[3]https://github.com/d3/d3-plugins/tree/master/sankey

Acedemic Semester: the academic level (BA1, MA3,...) of the students taking the courses.

It is also possible to modify the minimum number of co-students required to show an edge between two classes in order to generate a lighter or more complete graph.

Displaying more information on courses When a course is selected (either by putting the mouse on the node or by using the search tool), some additional information (if available) is displayed on the lower left corner of the screen. For now, in this additional information field, there is a graph showing the most related classes (by number of co-students) and the sections of students following this class. Course links' are also highlighted in red and the size of the course node is doubled to improve its visibility.
These bar plots provide more precise informations on a specific class than the original graph but would be less useful if we used alone.

Search tool On the upper right corner of the page, there is a search tool allowing for quick class lookups using auto-complete. This feature was implemented after the user study, conducted in the end of the semester, showed that it was the most requested feature.


Menu
Settings
Update
Co-Students
Section Ratio
1.0
0.8
0.6
0.4
0.2
0.0
0.46
0.27
0.27
0.23
0.19
Distributed algorithms
Cryptography and security
Intelligent Agents
Projet en informatique II
IT security engineerin
Large-Scale Distributed Systems
Concurrent algorithms
Intelligent Agents
Optional project in computer science
Advanced algorithms
Foundations of software
Advanced multiprocessor architecture
Unsupervised and reinforcement learning in neural
Security and Cooperation in Wireless Networks
Advanced computer networks and distributed systems
Computer-supported cooperative work
Projet en informatique II
IT security engineering
Optional project in communication systems
Algorithms in public-key cryptology
Cryptography and security
Distributed algorithms
Distributed information systems
Satelmetrics
TCP/IP networking
Software-defined radio : A hands-on course
Foundations of imaging science
Projet en systèmes de communication II
Information theory and coding
Advanced digital communications
Signal processing for
Automatic speech processing
Image and video processing
SHS : Projet complet
EDA T
Microelectronics for systems on chips
Biomedical signal processing

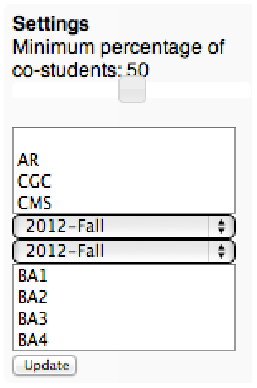
Settings
Minimum percentage of co-students: 21

AR
CGC
CMS
2012-Fall
2012-Fall
BA1
BA2
BA3
BA4

Search Course: Computer-supported co

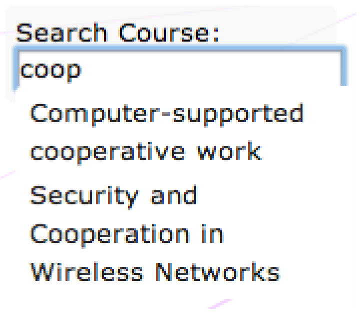
Computer-supported cooperative work
SC section
6 Credits
26 Students

Figure 3.1: Graph explorer page



Settings
Minimum percentage of co-students: 50

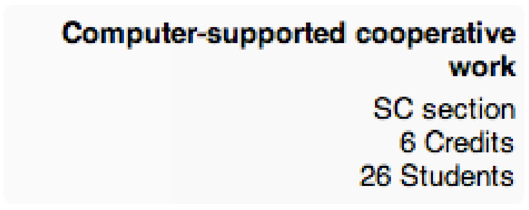
(a) Settings



Search Course:
coop

Computer-supported cooperative work
Security and Cooperation in Wireless Networks

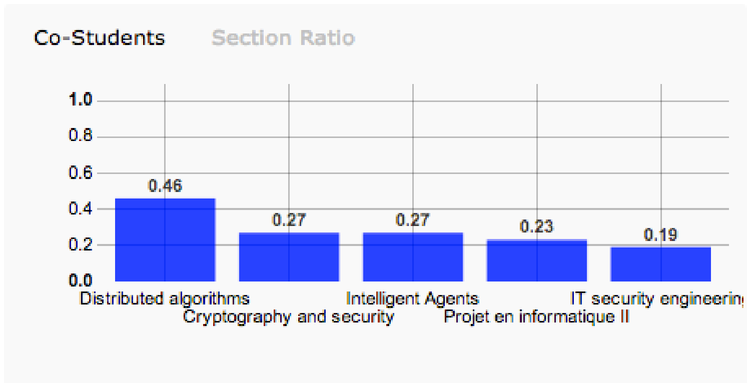
(b) Search



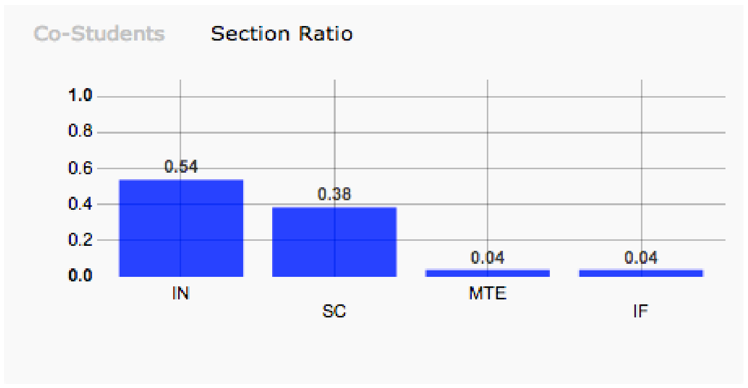
Computer-supported cooperative work
SC section
6 Credits
26 Students

(c) Details

Figure 3.2: Panels in Graph explorer



(a) Co-students



(b) Section of origin

Figure 3.3: Subgraphs in Graph explorer

What can be added In the main graph it is possible to represent data using many parameters.
For nodes we have (if we consider that nodes are circles, but we can also change the shape):

- Radius/Stroke width
- Fill/Stroke color

- Opacity

For edges we have:

- Length/Stroke width
- Fill/Stroke color
- Opacity
- Shape of edge (using arrows and other signs)

The length of an edge is an approximation, as it is often impossible to generate a graph with arbitrary lengths for every edge. The force directed layout, however, provides a great replacement.

3.2.2 SHS explorer/Section explorer

The section explorer considers the repartition of students from different sections in courses as a flow. The goal of this graph is to easily visualize which courses are the most popular, in terms of which sections the students following them are in.

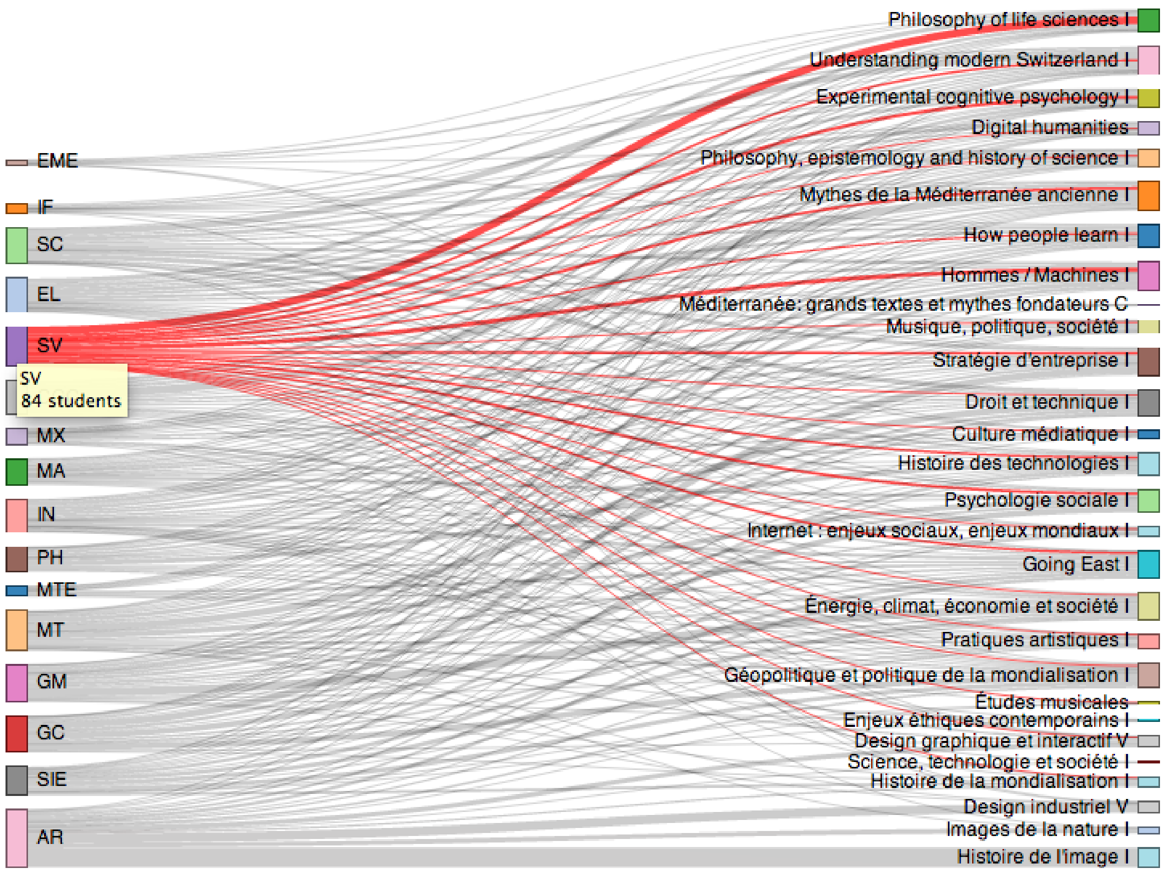


Figure 3.4: Section explorer for SHS

3.3 User study

A user study was conducted among 16 students of the IC faculty, particularly from the Computer Supported Cooperative Work course.
The most recurrent remark was that it was really hard to find a particular course on the graph, leading us to the implementation of the auto-complete search tool.

3.3.1 Quantitative analysis

Table 3.3.1 shows the results for quantitative questions. The first observation that we can make is that users did not see how this application could help them make decisions about their study plan, even if they found the information displayed relevant and easy to access. We can also see that users did not find any particular use in the section/shs explorer.

Category	Question	Mean (1-5)
General	Does the presented information seem relevant to you?	3.54
	Did you find it easy to browse and use the app'?	3.54
Course explorer	Did you identify clusters representing your study experience?	3.46
	Was it easy to find a class?	3.30
	Would it help you choose a speciality/optional class?	2.77
	Are the additional graphs related to specific classes useful?	2.69
Section explorer	Does the representation hold relevant information?	3.62
	Would it help you find an interesting SHS/optional class?	2.69
	Would you choose your SHS based on this kind of graphs?	2.84

Table 3.1: Grades given by testers, from 1 (not a all) to 5 (strongly)

3.3.2 Qualitative analysis

Course Explorer

- *Some data is wrong concerning the years where classes are given*
- *Nodes displayed as circles should be thought again: not enough space for text*
- *I don't understand how it works. There is not enough information.*

Section Explorer

- *I don't really understand what it represents, some information would be great*
- *I can see where my colleagues from my section go*
- *I think that representing section versus courses is good, but I choose my courses given my interests, not given other peoples choices*
- *You should add the description of the class and the grading policy*
- *It would be interesting to have previously chosen SHS, bra size + relationship status.*
- *Relation with hobbies and what people want to do after EPFL*

General remarks

- *The visualizations are great but I am not sure that it is necessary*
- *This application would be more suited for boards and commissions than for students*

3.3.3 Conclusion of the user study

The comments show that the testers had difficulties to understand how the visualizations worked and what they represented. It is thus necessary to implement a *help* button.
The course explorer is appreciated and representative of students experience but students don't see how it can help them in their exploration of the study plan. It would be interesting to get the feedback of board members to know if it can help them better understand the practical structure of the study plan.
The section explorer is less appreciated than the course explorer. When creating it we thought that it would be the most helpful representation: the study however showed us that we were wrong and that people, aside from the fact that it represents some interesting relations, don't find it useful.

Chapter 4

Import

The `import` project is quite small, but holds a few key urls where data can be obtained from is-academia. As the interface to access the data is currently evolving, a straightforward importation method has not been implemented.
Scala allows easy basic XML manipulation, and examples can be found in the `ISAxmlImporter` object.

4.1 ISA interfaces

The current interface IS-Academia is actually working on can be found here: `https://isanwww1.epfl.ch:8443/site-1.0/`. This is currently a Beta and might hence change quite rapidly.

4.1.1 Legacy URLs

These URLs were used at the beginning of the semester to build the first database draft. It is unclear if they will still be available in the future, and they are available even from the outside of EPFL's network.

Course list `http://isa.epfl.ch/wsa/cles/ClesInscr`. When called with an argument like "2012-2013", it returns the list of all courses. Example: `http://isa.epfl.ch/wsa/cles/ClesInscr?invoke=getlistecles&wwXAnneeacad=2012-2013`

Course information Data about a specific course can be obtained through `http://isa.epfl.ch/wsa/fiches/FichesCours`. Course keys obtained from the course list are of the form **M**784635267_**G**1378361727_**P**100304_**P**106177_**P**132574 where the M stands for *Matiere*, the G for *Groupe* and P for *Professeur*. Such a key is used to grab a *Fiche de cours*: `http://isa.epfl.ch/wsa/fiches/FichesCours?invoke=getfiches&wwXMatiere=&wwXAnneeacad=&wwCSection=&wwCNiveau=&wwXCle=M784635267_G1378361727_P100304_P106177_P132574`

Course subscriptions This interface shows the students subscribed to a given course: `http://isa.epfl.ch/wsa/inscr/inscriptions`. Example: `http://isa.epfl.ch/wsa/inscr/inscriptions?invoke=getinscriptionsbymatiere&wwXMatiere=&wwXAnneeacad=&wwXSection=&wwXTypesem=&wwXCle=M784635267_G1378361727_P100304_P106177_P132574`

Chapter 5

Improvements

5.1 Core

The following parts of the project would deserve further improvements:

5.1.1 Database schema

The schema contains entities that could be merged, namely:

`prerequisite` could be merged into `topicrelationmap`

`teaches` and `assists` could be merged into a single entity

5.1.2 Data insertion

Currently, the insertion is done via the Mapper object's `fill()` methods. As objects generally contain many other objects that may also require database insertions or updates, many calls go back and forth between the application and the storage, introducing sensible delays.
This is not a problem as long as import operations do not happen too often: should this requirement change, stored procedures would speed up batch-import operations.

5.2 Import

No proper import procedure exists at the moment. Once IS-Academia proposes a stable and well-defined interface to the required information, a *click-to-import* interface will be a good improvement.
Another solution could be to work directly with ISA to access their data without the need of an import step.
Finally, it would be great to have a deeper access to is-academia's data in order to obtain more details such as course grades, students grades, success rates... This would require some anonymization, but would increase the value of our product. Moreover, it would interesting to compute the semantic distance between courses by using either their description or all the data available on Moodle.

5.3 Visualisation

The user study showed that some tools to help users understand the way the application works are necessary. It would also be appreciated if the interface was globally simpler to lower the

learning time.
The visualization is very powerful, it would now be interesting to try different improvements in order to target the user's needs. This, of course, will be possible when the target audience for this service will be decided between students, teachers and comity members.

Chapter 6

Resources

6.1 Quick Start

1. Clone the `orgarec` repository:
   `git clone https://github.com/Shastick/orgarec.git`
2. Set the database access information in a property file, located in `view/src/main/resources/props`, (ie: the `orgarec` database on `craftsrv5.epfl.ch`'s Postgres database – for which you will need access right)
3. Run sbt, compile the sources and start the application:
   ```
   $ ./run
   > compile
   > project view
   > development:start
   ```
4. access the application through `http://localhost:8080`

We used chromium for our tests, and we recommend to use it to view the application. Other standard-abiding browser seem to do well, but they do not handle the intensive JavaScript usage equally and might interpret the CSS differently from what we intend.

6.2 Links

Code Documentation :

  core `http://shastick.github.com/orgarec/doc/core/`

  view `http://shastick.github.com/orgarec/doc/view/`

Source code `https://github.com/Shastick/orgarec`

JavaScript libraries :

  d3.js `http://d3js.org/`

  nvd3 `http://nvd3.org/`

  sankey `https://github.com/d3/d3-plugins/tree/master/sankey`

  jQuery UI `http://jqueryui.com/`