HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Institute for
Internet Technologies
and Applications

Modul Recommender für HSR Studenten

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2009

Autoren: Corsin Camichel, Léonie Fierz
Betreuer: Prof. Josef M. Joller
Gegenleser: Prof. Peter Sommerlad

1 Projektausschreibung

Studiengang	Informatik (I)
Semester	HS 2009/2010 (14.09.2009-14.02.2010)
Durchführung	Studienarbeit
Fachrichtung	Internet-Technologien und -Anwendungen
Gruppe	Corsin Camichel, Léonie Fierz
Betreuer	Prof. Josef M. Joller
Ausschreibung	Studenten der Hochschule für Technik können jedes Semester ihre Modulvorlesungen selber auswählen. Nachfolgend eine Auflistung der Faktoren, welche für die Modulwahl von Studierenden berücksichtigt werden können: - Modulvoraussetzungen - Bereits besuchte Module - Gesamtbelastung im Semester (Teilzeit/Vollzeit) - Bisheriger Prüfungserfolg - Belastung während Prüfungssession - Interessen - Erfüllung der Punkte in den verschiedenen Kategorien - Gesamtdauer Studium - Erreichung eines bestimmten Notenschnitts (Grades) - Modulangebot im aktuellen Semester (Frühjahr/Herbst) - Stundenplan (Überschneidungen von Vorlesungsterminen, Freitage für Teilzeit) Ziel: Effizientes Studium Alle Faktoren bei der Studiumsplanung einzubeziehen ist für den Studierenden schwierig. Der Musterstundenplan deckt den Normalfall ab, kann jedoch nicht in jeder Lage als Referenz herangezogen werden. Das zu entwickelnde Recommender System, der Recommender, soll daher eine zusätzliche Unterstützung bieten um eine Optimierung des Stundenplans (und des Studiums) an die Bedürfnisse des einzelnen Studierenden zu ermöglichen. Lösungsansatz: Web Modul Recommender Ein Student oder eine Studentin trägt in einer Online-Maske die bereits besuchten und/oder bestandenen Module ein. Zusätzlich wird ausgewählt, welches das nächste Semester ist. Anhand dieser Parametern wird ein Modulplan vorgeschlagen, welcher optimal auf den bisherigen Studiumsverlauf des Studenten/der Studentin angepasst wurde.
Voraussetzungen	Gute Programmierkenntnisse; Bereitschaft sich in neue Verfahren und Techniken einzuarbeiten.

2 Erklärung der Studenten

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.

Rapperswil-Jona, 18.12.2009

.....

Léonie Fierz

.....

Corsin Camichel

Diese Erklärung basiert auf [1].

3 Abstract

Der ”Modul Recommender für HSR Studenten” unterstützt Studierende mit einer Empfehlung bei der Modulwahl für das bevorstehende Semester. Die im System eingesetzten Modelle berücksichtigen unterschiedliche Faktoren einer Modulwahl. Die wichtigsten Faktoren sind der bisherige Verlauf des Studiums, Anforderungen für den Studiumsabschluss und die Voraussetzungen für einen Modulbesuch. Die verschiedenen, kombinierten Modelle sind ein Collaborative Filtering Algorithmus, ein Graphenmodell sowie einfache Filter.

4 Inhaltsverzeichnis

5 Management Summary

5.1 Ausgangslage

Wozu ein Recommender System?
Das Bologna-System erfordert eine grosse Eigenverantwortung des Studierenden[1]. Einerseits in Bezug auf den Vorlesungs- und Übungsbesuch, andererseits hinsichtlich der Modulwahl. Dies kann dazu führen, dass ein Student in einem Semester zu viele Module besucht und teilweise deren Prüfungen nicht besteht. Das kann den Bachelorabschluss gefährden oder verzögern.

Durch das Recommender System soll der Student bei der Modulwahl und der Planung seines Studiums optimal unterstützt werden. Die berechneten Modulempfehlungen dienen als Vorschlag und sind nicht zwingend zu berücksichtigen.

5.2 Vorgehen

Das System aggregiert historische Modulbesuche und wertet diese anhand von Modellen und Algorithmen aus. Dies ermöglicht die Abgabe einer Modulempfehlung für einen Studenten. Durch das Zusammenwirken der Modelle kann die Modulempfehlung verfeinert und optimiert werden.

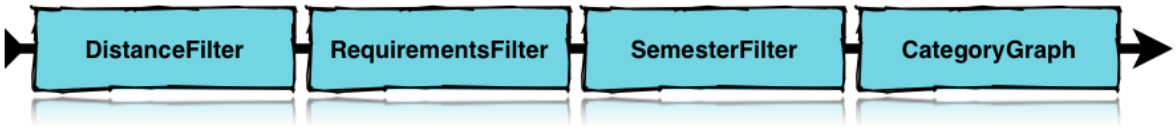

Abbildung 1: Ablauf einer Empfehlung mittels der verschiedenen Modelle

Eingesetzte Technologien und Daten

- Java
- MSSQL Datenbank
- Datenstamm von 440 HSR Informatik-Studenten, anonymisiert

5.3 Ergebnisse

In der Arbeit wurde ein Prototyp entwickelt, welcher Informatikstudenten der HSR für das nächste Semester eine Modulempfehlung anbietet. Folgende Aspekte wurden für die Berechnung berücksichtigt:

- Bisheriger Verlauf des Studiums
- Modulbesuche früherer Studierender
- Voraussetzungen für Modulbesuch
- Modulangebot im jeweiligen Semester
- Minimal erforderliche ECTS-Punkte pro Modulkategorie

[1] Studierende, Student werden geschlechtsneutral verwendet

5.4 Ausblick

Der Recommender-Prototyp hat verschiedene Erweiterungsmöglichkeiten. Dazu gehören im Speziellen:

- Empfehlung über mehrere Semester
- Einbezug von Prüfungsbesuchen
- Optimierung der implementierten Modelle
- Anwendung auf andere Studiengänge
- Integration des Recommender-Systems in die Strukturen der Hochschule

Der Prototyp wird im Frühjahrsemester 2010 im Rahmen einer Bachelorarbeit weiterentwickelt.

6 Konzeptueller Hintergrund

Dieses Kapitel erläutert den theoretischen Hintergrund von Recommender Systemen. Eine kurze Beschreibung der verschiedenen Ansätze soll einen Überblick über das Themengebiet geben. Weiter wird detaillierter die Verwendung von Recommender Systemen zur Modulempfehlung für Studenten aufgezeigt. Im Speziellen wird auf die im Modul Recommender für HSR Studenten verwendeten Algorithmen eingegangen.

6.1 Was ist ein Recommender System?

Ein Recommender System hat das Ziel, einem Benutzer eine Empfehlung abzugeben. Eine solche wird aus einem Datenstamm mit Informationen zu den Benutzern und den zu empfehlenden Objekten gewonnen. Das Einsatzgebiet von Recommender Systemen ist sehr breit. Bekannte Anwendungsgebiete sind Online-Plattformen für Filme, Musik oder Bücher. Weitere Beispiele sind die Vorhersage von Börsenkursen, die Empfehlung von Weblinks, personalisierte Werbung oder die Empfehlung von Modulen für Studenten. Ein Recommender System bietet gegenüber einer klassischen Suche oder Filterung den Vorteil, dem Benutzer eine Auswahl an Objekten zu empfehlen, die er selbst nicht gefunden hätte. Genauer ist damit gemeint, dass beispielsweise bei der Empfehlung von Musik dem Benutzer Stücke einer bisher unbekannten Musikrichtung empfohlen werden, welche seinen Geschmack dennoch treffen könnte. Ein gutes Recommender System wird vom Benutzer nicht als Werbung empfunden, sondern als eine Hilfestellung. Für Firmen sind Recommender Systeme nützlich bei der Generierung von zusätzlichem Umsatz [2, 3]. Ein Beispiel ist Amazon[2], wo einem Kunden für ihn möglicherweise interessante Bücher und andere Produkte speziell angeboten werden. Im Bereich von Filmbewertungen ist Netflix[3] und MovieLens[4] erwähnenswert, welche für Experimentier- und Forschungszwecke Daten von bis zu 100 Millionen Ratings zur Verfügung stellen. Netflix lancierte im Jahr 2006 einen Wettbewerb mit dem Ziel, einen im Gegensatz zu ihrem bestehenden Algorithmus wesentlich verbesserten Recommendation Algorithmus für Filme zu finden. Das Siegerteam arbeitete über drei Jahre an einer Lösung, welche über 100 verschiedene Algorithmen zusammenführt[5].

6.2 Überblick über die Techniken

6.2.1 Daten

Eine solide und aktuelle Datenbasis ist die Voraussetzung für eine aussagekräftige Empfehlung.

Datenbeschaffung Die Daten können entweder explizit oder implizit gesammelt werden. Beispiele für eine explizite Datenbeschaffung sind:

- Vom Benutzer eingegebene Bewertungen zu Objekten
- Benutzerprofil, welches vom Benutzer unterhalten wird
- Metadaten zu den Objekten (z.B. Themengebiet, Dozent)

[2] `http://www.amazon.de/`, zuletzt besucht am 05.12.09
[3] `http://www.netflix.com/`, zuletzt besucht am 05.12.09
[4] `http://www.movielens.org`, zuletzt besucht am 05.12.09
[5] `http://www.netflixprize.com`, zuletzt besucht am 05.12.2009

Beispiele für eine implizite Datenbeschaffung sind:

- Häufigkeit, mit der ein Benutzer ein Objekt auswählt, anhört oder anklickt
- Surfverhalten des Benutzers
- Klickanalyse von Werbebannern
- Speichern virtueller Einkaufskörbe

Expertenwissen Im Gegensatz zu Expertensystemen werden so wenig statische Informationen wie möglich in das System integriert. Ein Beispiel für Expertenwissen in Bezug auf einen Modul Recommender wäre die Einschätzung eines Professors zur Beliebtheit eines Moduls. Ein Recommender System ist nicht auf solche expliziten Experteninformationen angewiesen. Vielmehr können diese durch eine geeignete Analyse und Verknüpfung der gesammelten Daten vom Recommender System automatisch herauskristallisiert werden. Der Vorteil besteht darin, dass die Information nicht auf einzelne Experten abstützt und sich das System flexibel und automatisch auf neue Situationen einstellen kann.

6.2.2 Algorithmen und Modelle

6.2.2.1 Collaborative Filtering

Benutzerbasiert Der Algorithmus sucht für einen Benutzer nach ähnlichen Benutzern, den so genannten Nachbarn. Einem Benutzer X werden Objekte empfohlen, welche von seinen Nachbarn gewählt wurden, von X jedoch noch nicht. Damit wird eine gute Diversifizierung der Empfehlungen erreicht. Der Grund liegt darin, dass ein Benutzer typischerweise nicht nur eine Art (z.B. Musikstilrichtung) von Objekten ausgewählt hat. Beispielsweise könnte sich eine Musikstilrichtung mit der des aktuellen Benutzers decken, von einer zweiten Stilrichtung hat der aktuelle Benutzer bisher noch kein Objekt ausgewählt. Eine Empfehlung eines Objekts dieser zweiten Stilrichtung ist eine Horizonterweiterung für den aktuellen Benutzer. Voraussetzung für diesen Ansatz ist es, Benutzer untereinander vergleichen zu können. Einerseits ist dies mittels eines Benutzerprofiles möglich, andererseits über das Vergleichen der bisher gewählten Objekte. Beim Letzteren ist es schwieriger, Nachbarn für Benutzer mit wenigen Objektbewertungen zu finden.

Objektbasiert Ein anderer Ansatz besteht darin, die Ähnlichkeit zwischen den Objekten zu bestimmen. Einem Benutzer X, welcher Objekt A gewählt hat, werden die Objekte empfohlen, die am besten zu Objekt A passen. Dadurch werden Objekte aus dem Umfeld der vom Benutzer bereits gewählten Objekte empfohlen. Die Ähnlichkeit kann über die Objekteigenschaften bestimmt werden oder anhand der Bewertungen, die für das Objekt von den Benutzern abgegeben wurde. Das Paradigma beim zweiten Ansatz lautet: "Benutzer, die Objekt X gewählt haben, wählten auch Y". Das Objekt X ist demnach dem Objekt Y ähnlich. Objekte mit wenigen Bewertungen können dabei problematisch sein. Mit dieser zweiten Möglichkeit zur Bestimmung der Objektähnlichkeit wird eine bessere Diversifizierung der Empfehlungen gewährleistet.

kNN Ein konkreter, beliebter Algorithmus ist der k-Nearest-Neighbors (kNN). Das "k" steht für eine gewisse Anzahl von Nachbarn, welche für eine Empfehlung massgebend sein sollen. Unter Nachbarn können Benutzer oder Objekte verstanden werden. Mit "Nearest" sind die Benutzer oder Objekte gemeint, welche dem aktuellen Benutzer oder Objekt am ähnlichsten sind. Objekt- beziehungsweise Benutzer-Eigenschaften müssen dabei für die Berechnung der Ähnlichkeit nicht zwingend analysiert werden.

Optimierungen Für die Berechnung von Ähnlichkeiten kann als Verfeinerung auch ein Zeitfaktor berücksichtigt werden. Beispielsweise haben ältere Objekt-Bewertungen eine kleinere Auswirkung auf die Ähnlichkeit als Bewertungen, welche näher am Zeitpunkt der aktuell zu berechnenden Empfehlung liegen.

6.2.2.2 Singular Value Decomposition (SVD)

Singular Value Decomposition[6] Algorithmen gehen von einer Matrix aus, welche in einer Dimension die Benutzer und in der anderen die Objekte auflistet. Die nicht bewerteten Objekte bilden zu Beginn Leerzellen. Mittels der Matrixfaktorisierung mit Singular Value Decomposition werden Vektoren berechnet, aus denen die Originalmatrix näherungsweise zurückberechnet werden kann. Dabei werden dann die Leerzellen gefüllt.

Pseudo SVD Eine mögliche Implementierung ist der Pseudo-SVD Ansatz. Die Vektoren werden in iterativen Lernschritten berechnet. Für einen Teil der in der Originalmatrix leeren Zellen sind die erwarteten Werte bekannt. Dadurch lassen sich die berechneten Vektoren prüfen. Anhand der Abweichung kann in jedem Lernschritt einer der Vektoren korrigiert werden.
Das SVD Modell kann auch mit Zeitaspekten oder Korrekturfaktoren pro Benutzer oder Objekt erweitert werden. Auf diese Optimierungen wird hier nicht weiter eingegangen.

6.3 Anwendung im Bereich Modulempfehlung

Modulempfehlung für Studenten[7] mittels Recommender Technologie ist ein aktuelles Forschungsgebiet. Als Grundlage und Inspiration für unsere Arbeit diente ein an der DePaul University, Chicago veröffentlichte Publikation sowie Publikationen über einen Modul-Recommender für die Stanford University. Im Folgenden wird eine Kurzzusammenfassung dieser Arbeiten gegeben.

DePaul University In der Publikation der DePaul University "AACORN: A CBR Recommender for Academic Advising"[4] wird beschrieben, wie mittels eines Case Based Reasoning Modells einem Studenten Module empfohlen werden können. Dieses Modell funktioniert ähnlich wie ein benutzerorientiertes Collaborative Filtering Modell. Grundsätzlich werden einem Studenten noch nicht besuchte Module empfohlen, welche von ähnlichen Studenten bereits belegt wurden. In der Arbeit wird vertieft auf die Berechnung des Abstands zwischen zwei Studenten eingegangen. Dafür werden die Studienrichtung, die Durchschnittsnote, das Semester sowie die bisher besuchten Module herangezogen. Der Vergleich der Modulbelegungs-Geschichte ist dabei die grösste Herausforderung. Die Modulbelegungs-Geschichte kann als Vektor dargestellt werden und der Abstand zwischen zwei dieser Vektoren über die euklidische Distanz berechnet werden. Dabei wird jedoch vernachlässigt, dass die Reihenfolge der Modulbelegung eine Rolle spielt. Wird die Ähnlichkeit über die beschriebene Edit Distance berechnet, wird die Reihenfolge berücksichtigt. Die Reihenfolge der Modulbelegung im Semester ist jedoch irrelevant, weshalb die Formel mit einem diesbezüglichen Korrekturfaktor erweitert ist.

[6] Einfache Erklärung des SVD Algorithmus: `http://www.puffinwarellc.com/index.php/news-and-articles/articles/30-singular-value-decomposition-tutorial.html`, zuletzt besucht am 19.10.2009

[7] Studierende, Student werden geschlechtsneutral verwendet

Stanford University In den Arbeiten zum Modul-Recommender für die Stanford University wird der grundlegende Ansatz des Collaborative Filtering durch weitere Modelle ergänzt.
In der Arbeit "Recommendation Systems with Complex Constraints: A CourseRank Perspective"[5] wird unter anderem beschrieben, wie die Erfüllung einer Minimalpunktzahl an ECTS-Punkten in den verschiedenen Modulkategorien eines Studiengangs modelliert werden kann.
Das Modell benutzt einen "Directed Acyclic Graph" um die Modulbelegungen und Kategorien abzubilden. Den Kanten werden Kosten und Kapazitäten zugewiesen. Anhand dieser kann mit einem Min-Cost-Max-Flow Algorithmus bestimmt werden, welche Module noch belegt werden könnten. Der Algorithmus stellt sicher, dass mit den empfohlenen Modulen die Minimalpunktzahl in allen Kategorien erfüllt sind.
In einer zweiten Arbeit "Evaluating and Combining Recommendations with Prerequisites"[6] wird ein Modell beschrieben, welches Abhängigkeiten zwischen Objekten berücksichtigt. In Bezug auf einen Modul-Recommender sind dies Modul-Voraussetzungen. Ein Modul kann voraussetzen, dass vor dem Besuch zwingend andere Module bestanden sein müssen. In einem Graphen werden die Module als Knoten und die Abhängigkeiten beziehungsweise Voraussetzungen als Kanten dargestellt. Den Modulen wird eine durch einen vorgeschalteten Recommender berechnete Gewichtung zugewiesen. In der Publikation werden drei verschiedene Algorithmen beschrieben, um eine Modulauswahl mit möglichst grosser Gewicht-Summe zu finden. Der Zweck des Algorithmus ist die Sicherstellung, dass für alle empfohlenen Module die Voraussetzungen erfüllt sind oder durch den Besuch eines empfohlenen Moduls erfüllt werden.

6.4 Anwendung im Modul Recommender für HSR Studenten

Wie in *Kapitel 7* festgelegt, ist das Ziel dieses Recommender-Prototyps, eine Empfehlung für Modulbesuche des nächsten Semesters abzugeben. Die Beschränkung auf eine Empfehlung für das nächste Semester reduziert die Komplexität, bietet jedoch weniger Spielraum in Bezug auf eine Optimierung der Empfehlungen im Rahmen der gesamten Studiumsplanung.

6.4.1 Verwendete Modelle

Als Basis für eine Empfehlung wird von einem Collaborative Filtering Modell (*Kapitel 6.2.2.1*) ausgegangen. Die hier verwendete Implementierung sucht ähnliche Studenten und empfiehlt Module, welche diese bestanden haben (*Kapitel 6.4.3*). Um einem Studenten Module zu empfehlen, die er für die Erfüllung der Punkteanforderungen in den verschiedenen Kategorien brauchen kann, wird ein Graphenmodell (*Kapitel 6.4.4*) eingesetzt.
Weiter wurden zwei einfache Filter implementiert. Der erste Filter entfernt Module, die im gewünschten Semester nicht angeboten werden. Im zweiten Filter wird überprüft, ob der Student für die empfohlenen Module die nötigen Voraussetzungen erfüllt hat.

Die Aufgaben der Filter könnten teilweise auch durch eine Verbesserung des Collaborative Filtering Modells erreicht werden. Der Nachteil dabei wäre allerdings, dass die Bedingungen dann nur noch implizit in die Empfehlung einfliessen und beispielsweise im Fall der Modulvoraussetzungen nicht mehr garantiert wäre, dass der Student für das vorgeschlagene Modul alle empfohlenen Module abgeschlossen hat. Zudem würde die Komplexität des Collaborative Filtering Algorithmus stark erhöht und der Spielraum für Vorschläge schon früh eingeschränkt.

6.4.2 Nicht verwendete Modelle

Objektbasiertes Collaborative Modell Der Kern eines solchen Modells wäre der Vergleich von Modulen anhand einer Analyse der Modulbeschreibungen. Da die HSR eine relativ beschränkte Modulauswahl hat und zudem keine Module angeboten werden, die denselben Stoff beinhalten, ist dieses Modell nicht sehr vielversprechend. Die vorhandenen Modulähnlichkeiten werden durch die bereits verwendeten Modelle teilweise berücksichtigt. Module in demselben Themengebiet (beispielsweise Netzwerk, Software Engineering, Sprachen) bauen meistens aufeinander auf und sind somit durch Voraussetzungen verknüpft, welche durch den Voraussetzungs-Filter berücksichtigt werden. Andererseits werden Studenten alle Module in einem Themenbereich belegen, falls sie sich dafür interessieren, und so werden diese Module bereits durch das Benutzer-Collaborative-Filtering (*Kapitel 6.4.3*) empfohlen.

Graphenmodell für Voraussetzungen Experimentell wurde dieses Modell [6] (*Kapitel 6.3*) erfolgreich implementiert. Mittels Breath First Picking wird die Modulkombination herausgesucht, welche das insgesamt höchste Gewicht hat. Da im entwickelten Prototyp des Modul Recommenders für HSR Studenten die Empfehlungen für nur ein Semester gemacht wird, ist dieses Modell im Moment überflüssig. Eine Möglichkeit bestünde darin, eine Rückkopplung einzubauen. Dadurch würden Module, die Voraussetzung sind für hoch gewichtete Module, höher gewichtet.

SVD Modell Einen sinnvollen SVD-Ansatz (*Kapitel 6.2.2.2*) auf Modul Empfehlungen anzuwenden, stellt sich als schwierig heraus. Der Grund liegt in den fehlenden Modulbewertungen. Deshalb wurde kein SVD Modell entwickelt.

6.4.3 Collaborative Filtering

Der entwickelte Distanz-Filter verwendet das kNN (*Kapitel 6.2.2.1*) Konzept.

Suchen der Nachbarn Im ersten Schritt wird nach Nachbarn des anfragenden Studenten gesucht. Ein Nachbar ist definiert als ein ähnlicher Student, das heisst, dass er möglichst viele Module bestanden hat, welche der anfragende Student ebenfalls bestanden hat. Ein erster Ansatz war es, die Ähnlichkeit mittels der euklidischen Distanz bezüglich der Summe der Punktzahl in den Modulkategorien zu berechnen. Die berechnete Ähnlichkeit hatte jedoch eine zu kleine Aussagekraft, da sich die Modulkategorien als zu grobkörnig erwiesen. Als zweiter Versuch wurde pro Kategorie gezählt, welche Module vom anfragenden und dem zu vergleichenden Studenten bestanden wurden. Als Optimierung werden Studenten, die im gleichen Jahr das Studium begonnen haben oder im gleichen Leistungssemester sind, als ähnlicher eingestuft. Dieser Ansatz stellte sich als sinnvoll und sehr performant heraus.

Bestimmen der Empfehlungen Im zweiten Schritt werden aus den Modulgeschichten der Nachbarn diejenigen Module ermittelt, die der anfragende Student noch nicht bestanden hat. Den Modulen wird ein Gewicht zugeordnet. Ein Gewicht ist umso höher, je mehr Nachbarn ein Modul bestanden haben. Für die Gewichtung wird zudem berücksichtigt, vor wie langer Zeit das Modul bestanden wurde. Der Grund für solch eine Anpassung liegt darin, dass Module über die Jahre verändert werden oder andere Dozenten ein Modul übernehmen.

6.4.4 Graphenmodell für Minimalpunktzahl pro Kategorie

Wie bereits erwähnt, ist der Zweck dieses Modells dem Studenten Module zu empfehlen, deren ECTS-Punkte an eine Kategorie angerechnet werden können, in welcher er die minimale ECTS-Punktzahl noch nicht erreicht hat. Die Empfehlungen werden mittels eines Max-Flow-Min-Cost Algorithmus berechnet. Die Idee lehnt sich an das in [5] beschriebene Modell an.

Aufbau des Graphen Die Module und Kategorien werden als Knoten dargestellt. Zusätzlich wird ein künstlicher Start- und Endknoten eingeführt. In Abbildung 2 ist ein solcher Graph dargestellt. Die Anzahl Module und Kategorien wurde zwecks Übersichtlichkeit klein gehalten. Der Graph ist in drei Schichten unterteilt: Die erste Stufe verbindet den Startknoten mit allen Modul-Knoten. In der zweiten Stufe werden die Modulknoten mit ihrer zugehörigen Kategorie verbunden. Zudem wird die Kategorie "Andere" künstlich hinzugefügt. Zu dieser Kategorie gehören die acht frei wählbaren ECTS-Punkte. Alle Module werden zusätzlich mit dieser "Andere" Kategorie verbunden, da jedes Modul für die Erfüllung dieser Kategorie verwendet werden kann. Im Ausgangsgraph werden den Kanten auf der ersten und zweiten Stufe die Anzahl ECTS-Punkte des entsprechenden Moduls als Kapazität zugewiesen. Den Kanten auf der dritten Stufe, abgehend von den Kategorien, wird die Minimalanzahl ECTS-Punkte, welche in dieser Kategorie erworben werden müssen, als Kapazität zugeordnet. Diese Kapazitäten werden für den Max-Flow Teil des Algorithmus benötigt. Den Kanten auf der ersten Stufe werden zusätzlich Kosten für den Min-Cost Teil des Algorithmus zugewiesen. Wurde ein Modul bereits besucht, so sind die Kosten 0. Noch nicht besuchte Module haben Kosten von 2.

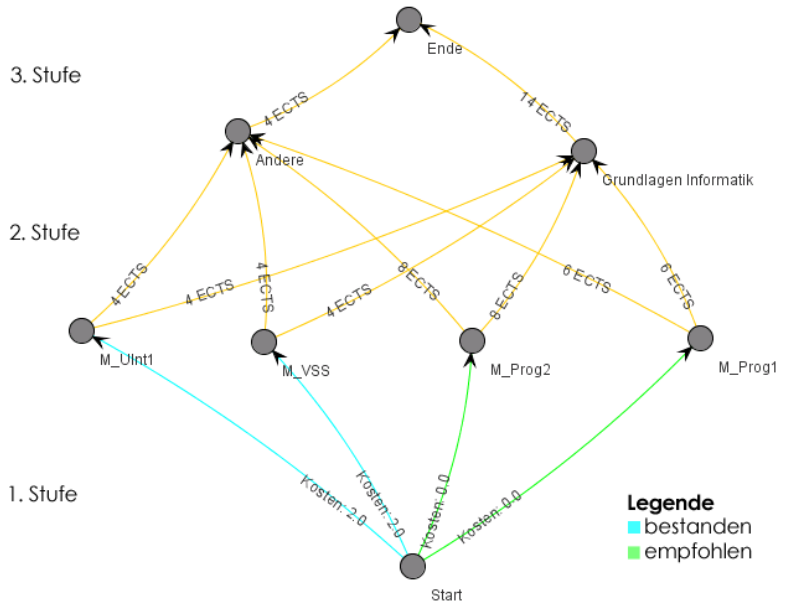

Abbildung 2: Ausgangslage für Min-Cost-Max-Flow Algorithmus (ohne Gewichtungen)

Algorithmus Der Max-Flow-Min-Cost Algorithmus [7] kann nun anhand des Graphen in Abbildung 2 die Module ermitteln, welche benötigt werden, um die Minimalpunktzahlen in allen Kategorien zu erreichen. Der Max Flow Teil des Algorithmus ist nach Edmonds-Karp[8] implementiert: Kern des Algorithmus ist ein iteratives Suchen nach einem Pfad vom Start- zum Endknoten. Die Kapazität jeder Kante im Pfad muss grösser als 0 sein. Die kleinste Kapazität der im Pfad enthaltenen Kanten definiert den Fluss an ECTS-Punkten, welcher über diesen Pfad transportiert werden kann. Nach jeder Iteration wird der Fluss für den gefundenen Pfad gespeichert und die Kapazitäten der Kanten aktualisiert. Der Algorithmus terminiert, wenn kein Pfad mehr gefunden werden kann, auf welchen alle Kapazitäten grösser als 0 sind. Die Summe der Flüsse in den gespeicherten Pfaden ergibt den maximal möglichen Fluss durch den Graphen. Dieser maximale Fluss entspricht der Summe der ECTS-Punkte, welche mit den bereits besuchten und den empfohlenen Modulen erreicht werden kann. Der Max-Flow Algorithmus wird ergänzt durch eine Min-Cost Funktionalität. Für das Bestimmen der Pfade wird der Dijkstra Algorithmus für die Ermittlung eines Pfads mit minimalen Kosten eingesetzt[9]. Der Graph in Abbildung 2 transformiert nach Anwendung des Algorithmus zum Graphen in Abbildung 3.

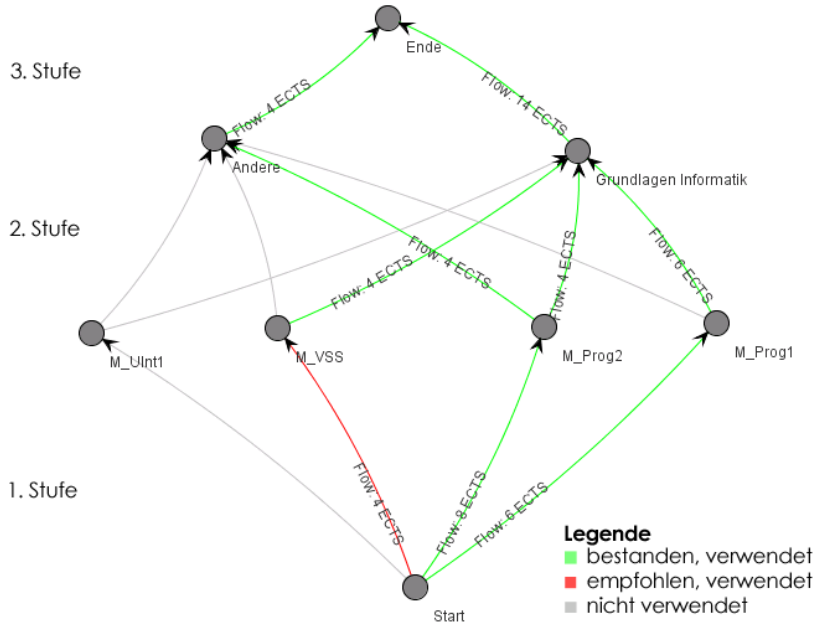

Abbildung 3: Resultat des Min-Cost-Max-Flow Algorithmus (ohne Gewichtungen)

[8] `http://de.wikipedia.org/wiki/Algorithmus_von_Edmonds_und_Karp`, `http://de.wikipedia.org/wiki/Ford-Fulkerson-Algorithmus`, zuletzt besucht am 22.10.09

[9] Vorlesung "Programmieren 2", HSR, Prof. J. Joller, besucht im Frühjahrsemester 2008

In [5] wird die Kapazität pro Modul auf 1 gesetzt. Pro Kategorie müssen somit eine gewisse Anzahl Module berücksichtigt werden. Module können jedoch eine unterschiedliche ECTS-Punktzahl haben. Um dies zu berücksichtigen wird in [5] vorgeschlagen, die Anzahl der zu besuchenden Module entsprechend anzupassen. Diese Korrektur wird in Kategorien vorgenommen, in welchen Module mit unterschiedlichen ECTS-Punktzahlen vorhanden sind. Wie bereits erwähnt, wird im Prototyp für den Modul Recommender für HSR Studenten als Kapazität die Anzahl ECTS-Punkte eingesetzt, damit der erwähnte Aspekt besser modelliert werden kann. Der Nachteil liegt darin, dass so die ECTS-Punkte eines Moduls auf zwei Kategorien aufgeteilt werden können. Im Studiengang Informatik ist ein Modul immer nur einer Kategorie und der künstlichen Kategorie "Andere" zugeordnet. Der Effekt des Aufsplitterns der ECTS-Punkte ist sogar erwünscht.

Einbezug der Pre-Recommendation Das Modell soll Pre-Recommendations von vorgeschalteten Modellen als Ausgangslage verwenden können. Je höher die Gewichtung einer Pre-Recommendation, desto wahrscheinlicher wird das Modul empfohlen. Im obigen Modell werden die Kosten für jedes noch nicht besuchte Modul auf die maximalen Kosten von 2 festgelegt. Diesen maximalen Kosten wird die Gewichtung (Wert zwischen 0 und 2) aus der Pre-Recommendation abgezogen. Dadurch werden Module mit höherer Gewichtung durch den Dijkstra Algorithmus bevorzugt. In den folgenden Abbildungen 4 und 5 ist das Beispiel von oben mit den zusätzlich eingefügten Gewichten dargestellt. Im Resultat ist ersichtlich, dass neu das Modul "M_UInt" vom Algorithmus vorgeschlagen wird. Dies entspricht der Erwartung, da von den vorgeschalteten Algorithmen für dieses Modul eine höhere Gewichtung berechnet wurde.

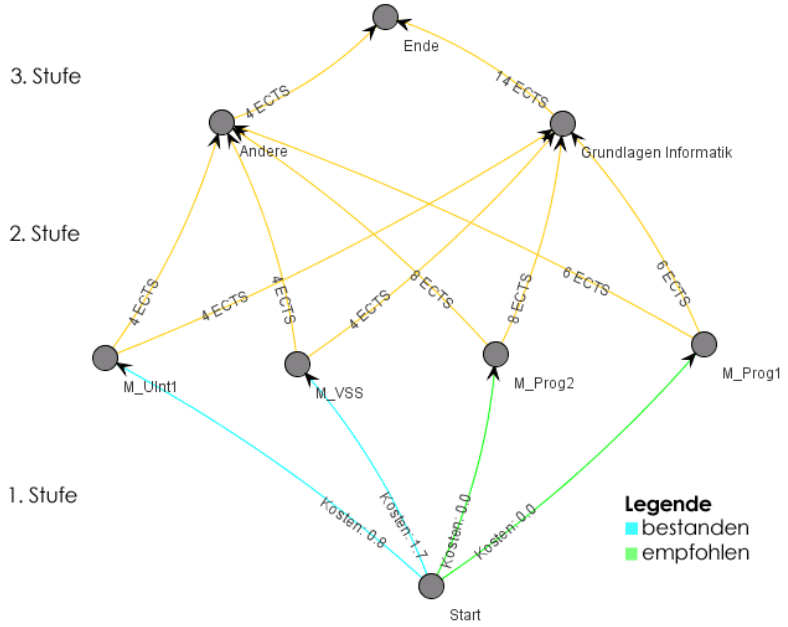

Abbildung 4: Ausgangslage für Min-Cost-Max-Flow Algorithmus (mit Gewichtungen)

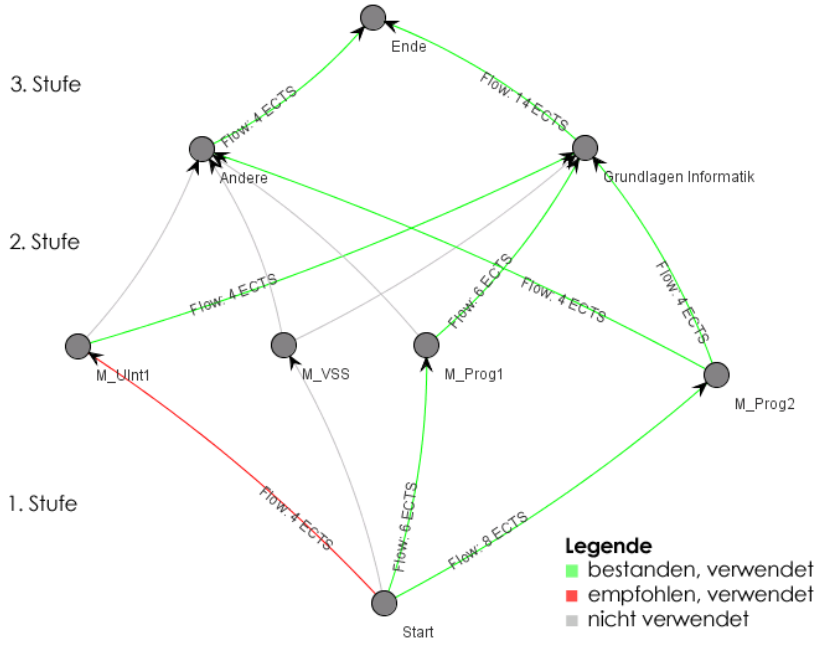

Abbildung 5: Resultat des Min-Cost-Max-Flow Algorithmus (mit Gewichtungen)

7 Anforderungen

7.1 Funktionale Anforderungen

Der Modul Recommender für HSR Studenten soll für eine für den Studenten möglichst ansprechende Empfehlung abgeben. Der Prototyp soll für Studenten des Studiengangs Informatik Modulvorschläge für das nächste Semester anbieten.

7.1.1 Use Case 1: Student wünscht Modulempfehlung

Use Case	UC1-Query - Student wünscht Modulempfehlung
Scope	Client
Level	Anwenderziel
Stakeholders & Interests	Student: will ansprechende Empfehlung für Modulbesuche im nächsten Semester
Preconditions	Student muss im System erfasst sein.
Success Guarantee	- Student erhält mindestens 2 Modulempfehlungen - System ist in konsistentem Zustand
Main Success Scenario	1. Eingabe des Studenten und des Semesters 2. System berechnet mögliche Modulbesuche 3. Empfehlung wird dem Studenten übergeben
Extensions	1.a. Das System erkennt den Studenten automatisch mittels eines Login 2.a. Der Student besitzt noch keine Modul-Historie 2.a.1. Das System empfiehlt die meistbesuchte Modulkombination 2.b. Der Student hat alle Module bereits besucht 2.b.1. Das System gibt keine Empfehlung ab
Special Requirements	-
Frequency of Occurence	Beliebig oft, je nach Bedürfnis eines Studenten.

Tabelle 1: UC1-Query

7.1.2 Use Case 2: Vorberechnen von Modulempfehlungen

Use Case	UC2-Automation - Vorberechnen von Modulempfehlungen
Scope	System
Level	Subfunction Level
Stakeholders & Interests	Student: will schnelle Antwort vom System
Preconditions	System besitzt genügend Daten zu Studenten und Modulbesuchen
Success Guarantee	- System ist in konsistentem Zustand - Empfehlungen wurden berechnet
Main Success Scenario	1. System lädt alle Daten zu den Studenten und Modulbesuchen *Schritte 2 und 3 für jeden Studenten wiederholen, welcher Modulempfehlungen abfragen kann* 2. Die für eine Empfehlung nötigen Vorberechnungen durchführen 3. Vorberechnung speichern
Extensions	2.a. Statt der Vorberechnungen werden direkt die Modulempfehlungen berechnet 2.b. Für einen Studenten sind noch keine Modulbesuche vorhanden 2.b.1. System führt nötige Vorberechnungen für die Bestimmung der meistbesuchten Module aus
Special Requirements	Die Änderung der zugrundeliegenden Daten der Modulbesuche sollten möglichst periodisch sein, so dass die Vorberechnungen nicht oft durchgeführt werden müssen.
Frequency of Occurence	Mindestens einmal pro Semester, wenn die Modulanmeldungen der Studenten und somit die Daten ändern.

Tabelle 2: UC2-Automation

7.2 Nicht funktionale Anforderungen

Richtigkeit der Modulempfehlungen Richtlinien für die absolute Richtigkeit der Modulempfehlungen sind nicht möglich. Nachfolgend erwähnte Punkte sollen jedoch für jede Empfehlung zutreffen. Für die vorgeschlagenen Module müssen die Vorbedingungen durch den Studenten erfüllt sein. Dies weicht von den Bedingungen der HSR ab, da dort in der Regel keine Bedingung für einen Modulbesuch vorgegeben sind. Weiter sollen vom Recommender nur Module vorgeschlagen werden, welche im gewünschten Semester stattfinden.

Anonymität der Daten Die dem Recommender zugrundeliegenden Daten über die Studenten, Modul- und Prüfungsbesuche dürfen zu keinem Zeitpunkt für den Endbenutzer ersichtlich sein. Dem Anwender ist es nicht möglich herauszufinden, welche Daten für eine Empfehlung verwendet wurden. In der Datenbank müssen die Studenten mit einer anonymisierten Identität abgelegt werden.

Anpassbarkeit Das System soll so konzipiert sein, dass einfach neue Algorithmen eingefügt, Algorithmen angepasst und die Reihenfolge des Aufrufs von Algorithmen verändert werden kann.

Schnittstelle zu Daten Die Daten stammen aus der Studentenverwaltung der HSR. Die Daten müssen manuell in die Datenbank des Recommender Systems importiert werden können.

Zeitverhalten Eine Modulempfehlung soll so schnell berechnet werden können, dass ein Benutzer eine Empfehlung interaktiv abrufen kann. Der Start des Systems sowie allfällige Vorberechnungen unterliegen keinen genauen Zeitlimiten. Der Systemstart kann im Rahmen von mehreren Minuten liegen, die Vorberechnungen können im Extremfall im Zeitrahmen von mehreren Tagen oder Wochen sein.

7.3 Einschränkungen

Modulangebot Das Modulsystem der HSR bietet eine relativ kleine Auswahl an Modulen. Dadurch ist die Auswahlmöglichkeit je nach Kategorie beschränkt. Zum Beispiel müssen für die 64 Grundlagen ECTS-Punkte beinahe alle angebotenen Module besucht werden. In einer solchen Kategorie sind die Unterschiede zwischen den einzelnen Studierenden gering.

Datenstamm Die in *Kapitel 9.2* beschriebenen verfügbaren Daten müssen für die Berechnung einer Modulempfehlung ausreichen.

7.4 Studiumsstruktur an der HSR

Die HSR bietet verschiedenste Studiengänge an. Der Aufbau des Studiums ist in jedem Studiengang unterschiedlich. Da der Prototyp für den Bachelor Studiengang Informatik erstellt wurde, wird nachfolgend nur der Aufbau dieses Studiengangs (Stand Dezember 2009) erläutert.

Struktur und Anforderungen Das Studium ist nach dem Bologna-Modell aufgebaut. Für Modulbesuche mit erfolgreichem Abschluss werden dem Studenten ECTS-Punkte gutgeschrieben. Ziel des Studiums ist die Erreichung des Bachelor-Abschlusses. Dafür sind folgende Bedingungen zu erfüllen:

- Erreichen von 180 ECTS-Punkten
- Abschluss der Pflichtmodule (Semester- und Bachelorarbeit)
- Erreichen der erforderlichen Minimalpunktzahl pro Modul-Kategorie

Pro Semester darf sich ein Student für maximal 40 ECTS-Punkte einschreiben. Der Musterstundenplan der HSR sieht eine durchschnittliche Belastung von 30 ECTS-Punkten pro Semester vor.

Module und Kategorien Ein Modul ist beispielsweise Programmieren 1, Analysis 1, Verteilte Softwaresysteme oder Englisch 3. Auch die Studienarbeit und Bachelorarbeit werden als Modul bezeichnet. Jedem Modul ist eine Kategorie zugewiesen. Kategorien gruppieren gleichartige Module, beispielsweise Mathematik, Grundlagen Informatik, Aufbau Informatik oder Kommunikation und Sprache. Um die geforderten 180 ECTS-Punkte zu erreichen, sind neben der Erfüllung der Minimalpunktzahlen pro Kategorie mindestens 8 zusätzliche Punkte in einer beliebigen Kategorie zu erwerben.

Modulvoraussetzungen Viele der Module bauen auf anderen Modulen auf. Dies wird in den Modulbeschreibungen der HSR als "Empfohlene Module" bezeichnet, deren Besuch jedoch nicht zwingend ist. Eine Ausnahme bildet die Bachelorarbeit, welche nur nach erfolgreichem Abschluss der Semesterarbeit geschrieben werden kann.

8 Software Architektur Design

8.1 Einführung

Im ersten Teil wird die Gesamtstruktur (*Kapitel 8.2*) beschrieben. Im zweiten Teil (*Kapitel 8.3*) wird detailliert auf die verschiedenen Layer und deren Packages eingegangen. Ebenfalls werden Designentscheide hier dokumentiert. Die Beschreibung der Interfaces und Klassen samt deren Methoden sind in der Javadoc dokumentiert.

8.2 Übersicht Architektur

8.2.1 Wesentliche Merkmale

Der Aufbau der Datenstruktur mit allen zur Verfügung stehenden Datenfeldern und den Beziehungen der einzelnen Objekte ist in Abbildung 6 illustriert.

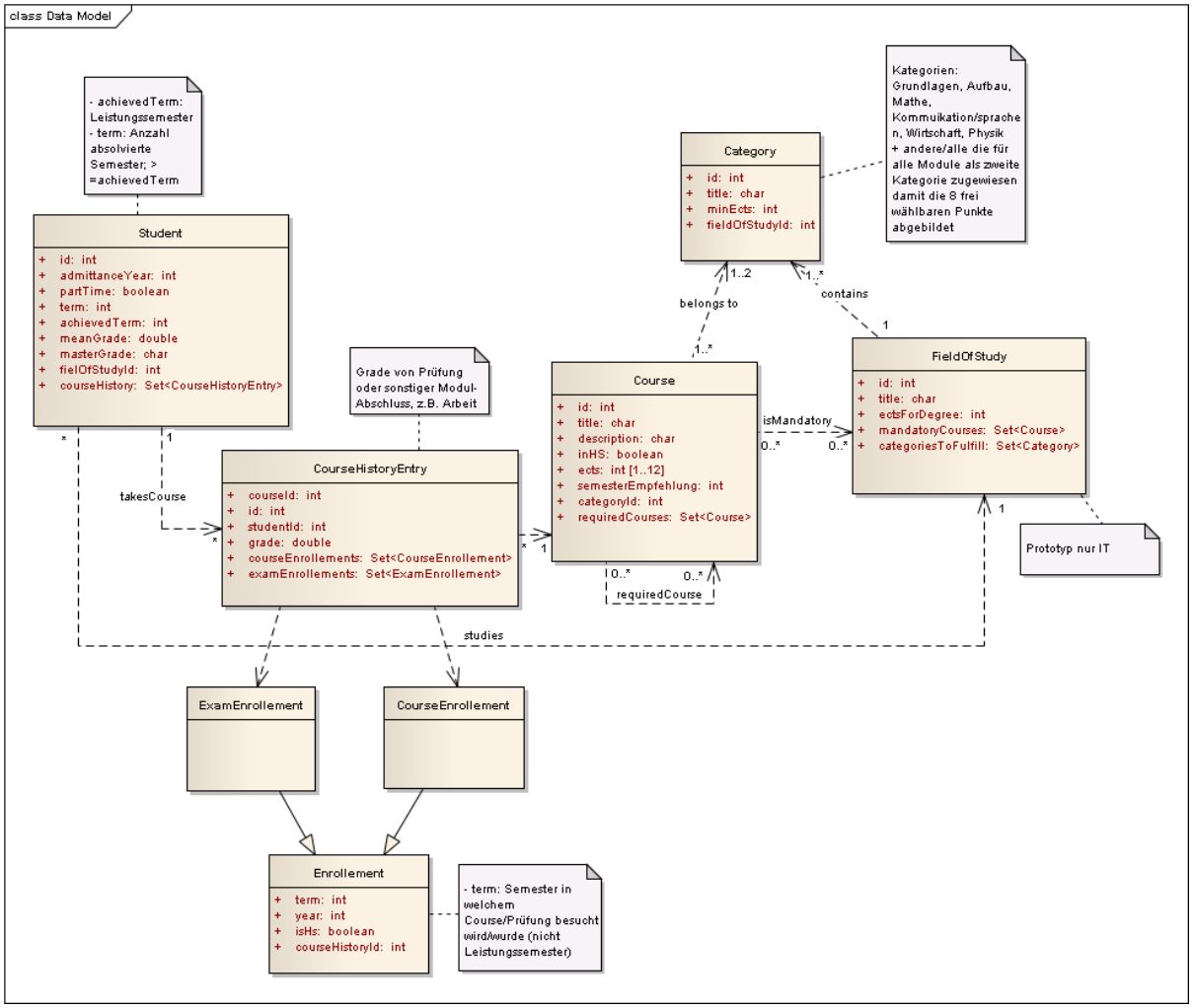

Abbildung 6: Domainmodell

8.2.2 Architektonische Ziele

Der Kern des Recommender Systems bestehend aus den verschiedenen Algorithmen unterliegt einer fortlaufenden Weiterentwicklung und Optimierung. Demzufolge soll das System eine Änderung und Kombination von Algorithmen mittels geeigneter Schnittstellen unterstützen. Weiteres Ziel ist es, den Zugriff auf die Datenbank von der Domain Schicht zu entkoppeln. Dies soll eine einfache Änderung der Datenbanktechnologie oder auch des Datenbankaufbaus ermöglichen.

8.3 Systemstruktur

8.3.1 Logische Schicht

Das Softwaresystem wurde in unterschiedliche Schichten unterteilt. Dies sind die Datenbankschicht (*Kapitel 8.4.5*), die Domain-Schicht (*Kapitel 8.4.6*), die Logik-Schicht (*Kapitel 8.4.1*) sowie eine minimale Applikations-Schicht (*Kapitel 8.4.4*). Diese Unterteilung der Komponenten in einzelne Schichten ist ein verbreitetes Vorgehen in der Softwareentwicklung und war auch in diesem Projekt sehr hilfreich. So konnten die einzelnen Schichten unabhängig voneinander getestet und entwickelt werden.

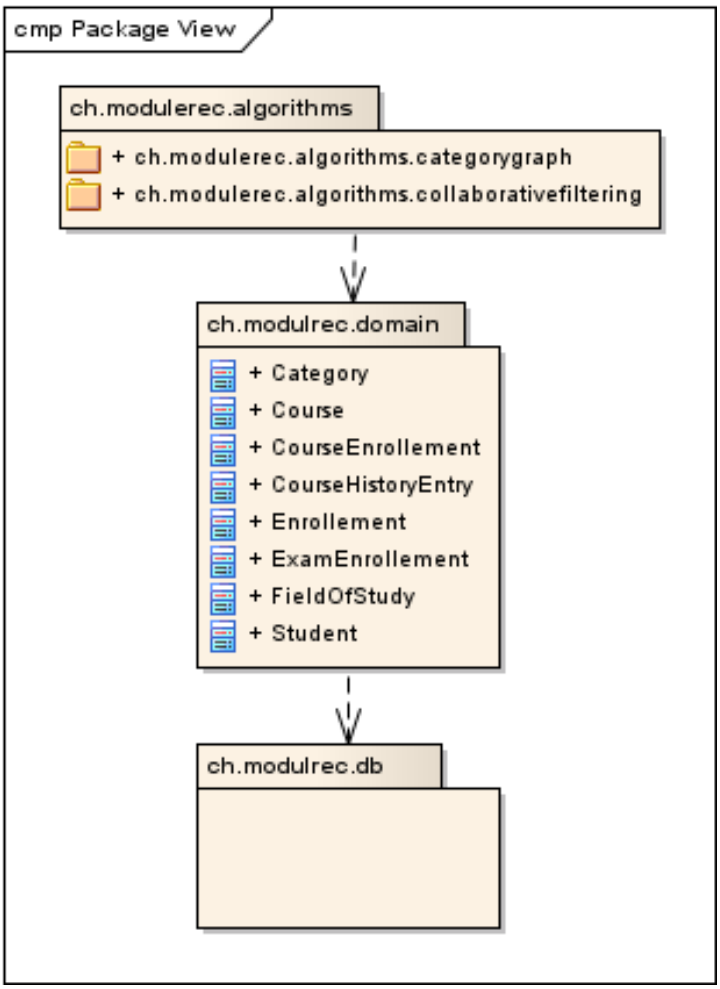

Abbildung 7: Sicht auf Packages

8.3.2 Architekturkonzepte

8.3.2.1 Modularisierung Algorithmen

Wie in *Kapitel 8.2.2* beschrieben, ist eine einfache Möglichkeit für die Weiterentwicklung der Algorithmen wichtig. Um dies zu gewährleisten, wurden die Algorithmen modular aufgebaut, mit dem Ziel, die verschiedenen Algorithmen in einer beliebigen Reihenfolge ausführen zu können. Dazu werden alle Algorithmen gegen das Interface *IAlgorithm* implementiert. Jeder Algorithmus kann für einen Studenten eine Liste empfohlener Module generieren, wobei den Modulen eine Gewichtung (Wert zwischen 0 und 2) zugeordnet wird. Je höher die Gewichtung, desto mehr wird das Modul empfohlen. Die ausgegebene Liste kann als Eingabe für einen nächsten Algorithmus verwendet werden, welcher diese als Pre-Recommendation verwendet. Dadurch können alle Algorithmen beliebig zusammenarbeiten. Der erste Algorithmus in der Kette muss mit der noch leeren Pre-Recommendation Liste zurechtkommen, das heisst neue Modulempfehlungen abgeben. Dies wird vom *DistanceFilter* und vom *CategoryGraph*, wenn konfiguriert, gewährleistet. In der folgenden Abbildung ist die im Prototyp verwendete Verkettung der Algorithmen dargestellt.

Abbildung 8: Kette der aufgerufenen Algorithmen

8.3.2.2 Datenbank-Abstraktion

Das Recommender System greift nicht direkt auf die Datenbank zu. Sämtliche Kommunikation erfolgt über das Hibernate Framework[10]. Hibernate ist ein mächtiges Werkzeug, um mittels Objekt-Relationaler Mappings die Datenbank und die Domain Objekte zu verbinden.
Das System benötigt nur Lesezugriff auf die Datenbank. Aufgrund dieser einseitigen Kommunikation wurde auf eine weitere Abstraktion der Domain Objekte mittels Data Access Objects verzichtet.

8.3.2.3 Exceptionhandling

Die grösste Fehlerquelle liegt bei der Kommunikation zwischen System und Datenbank. Das verwendete Framework Hibernate reagiert sehr sensible auf Fehler, welche durch Exceptions gemeldet werden. Das System kann ohne die Daten aus der Datenbank nicht arbeiten. Deshalb werden die Exceptions von Hibernate nicht vom System verarbeitet, sondern bis ganz nach oben zur Applikation weitergeleitet. Grundsätzlich werden keine Exceptions erwartet, weder von den Algorithmen noch von der Datenbankkommunikation. Da nur eine minimale Benutzerschnittstelle angeboten wird, werden die Exceptions nicht benutzerfreundlich dargestellt.

[10] `https://www.hibernate.org/`, zuletzt besucht am 8.12.2009

8.3.2.4 Eingesetzte Frameworks und Bibliotheken

Wie in *Kapitel 8.3.2.2* bereits beschrieben, wird Hibernate für die Kommunikation mit der Datenbank verwendet. Für die Realisierung des Kategorien-Graphenmodells (*Kapitel 8.4.2*) wurde das Java Universal Network/Graph Framework (JUNG, Version 2.0)[11] eingesetzt. Dieses bietet einige grundlegende Algorithmen sowie eine einfache Visualisierungsmöglichkeit für Graphen. Für das Testing wird JUnit 3.8[12] eingesetzt.

8.4 Logische Architektur

8.4.1 Package ch.modulrec.algorithms

Dieses Package enthält die Logik des Systems. Als Grundlage für die Algorithmen ist das Interface *IAlgorithm* definiert.

8.4.2 Package ch.modulrec.algorithms.categorygraph

Das Package "categorygraph" enthält die Implementierung des in *Kapitel 6.4.4* erläuterten Modells. Für die Handhabung des Graphen wird die JUNG Library [13] verwendet. Für den Max-Flow-Min-Cost Algorithmus konnte der Dijkstra-Algorithmus in die vorhandene Implementation des Edmonds-Karp Max-Flow Algorithmus integriert werden. Die Visualisierung des Modells konnte ebenfalls mithilfe der Library realisiert werden.
Der Algorithmus implementiert das Interface *IAlgorithm*. Damit der geforderten Ausgabe, einer Liste mit Modulen und Gewichtungen, entsprochen werden kann, leitet der Algorithmus die Modulgewichte der Eingabe unverändert weiter. Sinnvollerweise wird dieser Algorithmus am Ende der Modell-Kette (*Kapitel 8*) aufgerufen, weil die Auswahl der vorgeschlagenen Module unter Umständen stark eingeschränkt wird, falls nur noch wenige Punkte für die Erfüllung der Minimalpunktzahlen pro Kategorie benötigt werden. Als Ausgangslage können entweder nur die Module aus der Pre-Recommendation verwendet werden oder alle Module des Studiengangs, wobei die Gewichtungen der in der Pre-Recommendation vorhandenen Module in beiden Fällen beachtet werden.

8.4.3 Package ch.modulrec.algorithms.collaborativefiltering

Das Package *collaborativefiltering* enthält Klassen zur Filterung von Modulen anhand unterschiedlicher Kriterien. Zuerst erwähnt sei an dieser Stelle das Interface *IFilter*. Es erweitert das Interface *IAlgorithm* aus *Kapitel 8.4.1* und abstrahiert die von allen Filtern benötigten Schnittstellen.

DistanceFilter Diese Klasse implementiert das in *Kapitel 6.4.3* beschriebene Modell. Für die Berechnung werden 50 Nachbarn berücksichtigt. Die Ähnlichkeit zweier Studierender wird normalisiert, so dass die Zahl im Wertebereich zwischen 0 und 2 liegt. Der Distance Filter generiert eine komplett neue Liste an Modulen; Pre-Recommendations werden verworfen.

[11] `http://jung.sourceforge.net/`, zuletzt besucht am 8.12.2009
[12] `http://www.junit.org/`, zuletzt besucht am 8.12.2009
[13] `http://jung.sourceforge.net/`, zuletzt besucht am 8.12.2009

RequirementsFilter Das System setzt zwingend voraus, dass vorausgesetzte Module bestanden sind, bevor ein darauf aufbauendes Modul besucht werden kann (*Kapitel 7*). Diese Art von Filterung geschieht im *RequirementsFilter*. Als Input wird die von anderen Algorithmen berechnete Modulliste verwendet. Danach wird geprüft, ob der Student die vorausgesetzten Module für die empfohlenen Module bestandenen hat. Schliesslich werden nur die Module zurückgegeben, für welche die Voraussetzungen erfüllt sind. Die Gewichtung der Modulempfehlung wird nicht berücksichtigt oder verändert; in der zurückgegebenen Map jedoch eingetragen.

SemesterFilter An der HSR werden viele Module nur jährlich angeboten. Das bedeutet, dass gefiltert werden muss, ob die empfohlenen Module im gewünschten Semester angeboten werden. Das Resultat dieser Filterung ist wiederum eine Map mit Modulempfehlungen und einer Gewichtung. Wie im *RequirementsFilter*, wird die Gewichtung der Modulempfehlungen nicht modifiziert.

8.4.4 Package ch.modulrec.application

Dieses Package enthält eine Klasse *ConsoleApplication*, welche benutzt wird, um das System aus der Konsole abzufragen. Das Interface wurde bewusst einfach gehalten. Zurzeit wird es nur für Testzwecke verwendet, um das System einfach mittels weniger Eingaben bedienen zu können. Für spätere Versionen ist es durchaus möglich, ein komplexeres Interface zu entwerfen oder den Modul Recommender in ein bestehendes System einzubinden.

```
Please enter a student to predict: 868
Please enter the Semester:
5
1: Andere
2: Naturwissenschaften
3: Kommunikation und Sprache
4: Mathematik
5: Bachelor-Arbeit Informatik
6: Grundlagen Informatik
7: Aufbau Informatik
8: Gesellschaft, Wirtschaft und Recht
7
```

Abbildung 9: Konsoleninterface

8.4.5 Package ch.modulrec.db

Sämtliche Klassen für die Kommunikation mit der Datenbank sind in diesem Package abgelegt. Es beinhaltet unter anderem die Session-Factory für die Verbindung mit der Datenbank durch Hibernate. Zusätzlich befindet sich hier eine Klasse *DbConnector*, welche anhand statischer Methoden Abfragen auf die Datenbank ermöglicht. Zusätzlich sind die benötigten EnumTypen *EnumUserType* und *MasterGradeEnumUserType* in diesem Package enthalten.

8.4.6 Package ch.modulrec.domain

Dieses Package enthält sämtliche Domain-Klassen, insbesondere *Student* für Studenten, *Course* für Module sowie *CourseHistoryEntry* und *Enrollment* für Modulbesuche. Alle diese Klassen werden auf die Datenbank abgebildet. Ausgenommen der Vererbungshierarchie für *Enrollment* entsprechen alle Klassen einer Datenbanktabelle. Jede einzelne Klasse besitzt logische Verknüpfungen zu anderen Klassen, beispielsweise sind einem Studenten-Objekt bekannt, welche Module es besucht und bestanden hat sowie zu welchem Studiengang es gehört. Teilweise sind diese Verknüpfungen von beiden beteiligten Objekten aus navigierbar.

8.4.7 Package ch.modulrec.exceptions

Wie in *Kapitel 8.3.2.3* erläutert, sind wenige Exceptions nötig. Die einzige Exception *AlgorithmNotInitializedException* wird geworfen, wenn eine Implementation von *IAlgorithm* nicht alle benötigten Parameter und Variablen erhält. Diese Exception wird benötigt, da für eine Vorhersage beispielsweise zwingend der Student bekannt sein muss.

8.5 Testing

Dank der Schichtenarchitektur können der Datenbankzugriff und die Business Logik beziehungsweise die Algorithmen unabhängig voneinander getestet werden. Als Testing Framework wird JUnit 3.8[14] eingesetzt. Die geschriebenen Tests befinden sich im Quellordner *src/test/java*, die Packagestruktur entspricht derjenigen des Systems selbst. Dies erleichtert die Orientierung und erlaubt eine Trennung von Produktiv- und Testcode.

Datenbank Test Geprüft wird die korrekte Funktion des Datenbankzugriffs und des Mappings auf die Domainklassen. Ebenfalls getestet wird die Verknüpfung der Domainobjekte.

Business Logik Test Damit die Algorithmen unabhängig von der Datenbank getestet werden können, werden für die Domainschicht Mockobjekte erstellt. Die von den Mockobjekten bereitgestellte Datenbasis wird bewusst sehr klein gehalten, so dass manuell nachvollzogen werden kann, welchen Output die Algorithmen liefern sollten. Die Tests sichern die grundlegende Funktionalität der Algorithmen. Um alle möglichen Varianten konstruieren und testen zu können, wäre eine grössere Mockdatenbasis nötig. Die Tests würden dadurch komplexer, insbesondere die Vorhersage der Ergebnisse, woraus eine geringere Wartbarkeit resultierte. Für einen wichtigen Algorithmus wurde ein einfacher Test geschrieben, welcher die Integration mit der Datenbank prüft.

Systemtest Damit die Richtigkeit des Gesamtsystems geprüft werden könnte, wäre es notwendig, ohne Hilfe des Systems vorhersagen zu können, welche Module einem Studenten empfohlen werden sollen. Dazu ist jedoch das Gesamtsystem zu komplex und die Datenbasis zu gross.

[14] `http://www.junit.org/`, zuletzt besucht am 10.12.2009

9 Infrastruktur

9.1 Applikation

Der entstandene Prototyp verwendet als Programmiersprache Java in der Version 1.6. Um eine Empfehlung für einen spezifischen Studenten und ein Semester zu erhalten, wurde ein einfaches Konsoleninterface programmiert. Es erlaubt mit wenigen Eingaben (Studenten-ID, Semester und Kategorie) eine Empfehlung zu erstellen. Dabei werden die einzelnen Filterungs- und Algorithmenschritte auf der Konsole ausgegeben. Dies erlaubt es nachzuvollziehen, welche Module in einem Schritt entfernt wurden.
Als Dokumentation der einzelnen Schritte sei auf *Kapitel A.2* verwiesen. Dort finden sich die Beschreibungen der Interfaces und einige grundlegende Klassen.

9.2 Datenbestand

Die verwendeten Daten stammen aus einem Datenbankexport vom 25. November 2009 aus dem Studenten- und Modulverwaltungssystem der HSR (Adunis). Die exportierten Daten beinhalten sämtliche Informatikstudenten seit dem Jahr 2002. Dadurch sind Daten von Diplom-Studenten sowie Studenten, welche das Studium nach dem neuen Bologna-Modell absolvieren vorhanden. Die Module haben sich seit der Einführung des Bologna-Modells wesentlich geändert. Um eine möglichst gute Vorhersage für aktuelle Studenten zu erreichen, werden nur die Daten ab der Einführung des Bologna-Modells im 2005 verwendet. Der jetzige Prototyp verwendet somit einen Datenstamm aus 440 Studenten.
Aufgrund von Daten- und Persönlichkeitsschutz ist es nicht möglich, den Datenbestand an Dritte auszuhändigen.

9.3 Datenbank

Als Datenbank wird ein Microsoft MSSQL Server 2008 verwendet. Die Installation befindet sich auf einem von der Hochschule zur Verfügung gestellten Computer. Für den Import und die Anpassung auf das Datenbankschema des Modul Recommenders für HSR Studenten wurden SQL Skripts erstellt. Diese Skripte finden sich im *Kapitel A.1*.
Da das System nur Lese-Zugriffe erlaubt, war es problemlos möglich, mit mehreren Clients gleichzeitig auf der Datenbank zu arbeiten. Transaktionsprobleme mussten deswegen nicht berücksichtigt werden.

9.4 Entwicklungsumgebung

Entwickelt wurde auf Windows sowie Mac OS X. Als Entwicklungsumgebung wurde Eclipse IDE[15] eingesetzt. Für die Codeverwaltung und -versionierung wurde Subversion[16] verwendet.

[15] `http://eclipse.org/`, zuletzt besucht am 17.12.2009
[16] `http://subversion.tigris.org/`, zuletzt besucht am 17.12.2009

10 Schlussfolgerungen

Der Prototyp zeigt die Machbarkeit eines Modul Recommenders für HSR Studenten basierend auf der Aggregierung und Analyse von historischen Modulbesuchsdaten.

Korrektheit der Empfehlungen Über die Korrektheit der Modulempfehlungen kann grundsätzlich keine genaue Aussage gemacht werden. Um eine grobe Abschätzung zu erhalten, könnte ein Gütemass für eine Empfehlung definiert werden. Dieses würde die vom Modul Recommender für HSR Studenten empfohlenen und die tatsächlich von einem Studenten besuchten Module vergleichen. Ein solches Mass wurde im Rahmen der Prototypentwicklung nicht ausgearbeitet. Soweit nachvollziehbar, waren die getesteten Empfehlungen sinnvoll. Noch schwieriger zu quantifizieren ist, ob diese Empfehlungen für einen Studenten auch nützlich sind. Dies lässt sich nur anhand einer Testphase mit Studenten verifizieren.

Diversifizierung der Empfehlungen Die Datenbasis von 440 Studenten ist relativ klein. Zudem ist der Spielraum für die Auswahl von Modulen an der HSR klein. Dies ist für einige Modulkategorien verstärkt der Fall. Eine sehr geringe Diversifizierung bezüglich der historischen Modulbesuche zeigt sich beispielsweise für die Fächer in der naturwissenschaftlichen Kategorie. Von den fünf angebotenen Modulen haben rund 95% der Studenten *Physik 1* sowie *Physik 4 für Informatiker* besucht. Der Anteil an Modulbesuchen für *Physik 2* und dem *Naturwissenschaftlichen Praktikum* ist verschwindend klein, für *Physik 3* sind gar keine Modulanmeldungen verzeichnet. Daraus lässt sich schliessen, dass der Modul Recommender für HSR Studenten in den betroffenen Kategorien der Mehrheit der Studenten dieselbe Modulkombination empfehlen wird. Die Herausforderung besteht darin, den wenigen Studenten, für welche eines dieser selten belegten Module interessant sein könnte, dieses zu empfehlen.

Studenten ohne Modulbesuche Studenten, welche bisher noch keine Module besucht haben, soll auch ein Vorschlag für Modulbesuche abgegeben werden können. Der Modul Recommender für HSR Studenten empfiehlt in solchen Fällen die meistbesuchte Modulkombination für das erste Semester. Diese empfohlenen Module haben alle keine Voraussetzungen.

Zeitverhalten In den Anforderungen (*Kapitel 7*) ist die Möglichkeit für Vorberechnungen aufgeführt. Da alle Algorithmen eine kurze Laufzeit haben (< 1 Sekunde), ist eine solche Vorberechnungsphase nicht nötig. Der Systemstart benötigt circa 80 Sekunden, was gemäss Anforderungen (*Kapitel 7.2*) im Toleranzbereich liegt. Durch die Einführung von Lazy Loading [8] könnte die Systemstartzeit optimiert werden.

Graphenmodell für Kategorien Der *CategoryGraph* ist ein vorausschauendes Modell. Es optimiert die Erfüllung der verschiedenen Modulkategorien über den gesamten Studienverlauf. Da momentan die Empfehlung nur für das nächste Semester ausgegeben wird, kann das volle Potenzial des Modells noch nicht ausgeschöpft werden.

Neue Module Zu der Funktionalität eines Recommender Systems gehört, dass neu hinzugefügte Objekte für eine Empfehlung berücksichtigt werden. Der *CategoryGraph* kann so konfiguriert werden, dass nicht nur die Module aus der Pre-Recommendation beachtet werden, sondern alle Module. Dadurch wird, falls die Punkte für eine Kategorie mit den Modulen aus der Pre-Recommendation nicht erreicht werden, eine Kategorie mit beliebigen Modulen dieser Kategorie aufgefüllt. Dadurch werden potenziell auch neue Module empfohlen. Für das Gesamtsystem ist

diese Funktion jedoch ausgeschaltet. Da nur Module für das nächste Semester empfohlen werden, würden durch den *CategoryGraph* zu viele Module ausgewählt. In der aktuellen Berechnungskette der Modelle steht der *CategoryGraph* ganz am Ende, wodurch diese zu umfassende Modulliste direkt dem Benutzer präsentiert würde. Um die neuen Module auch im Gesamtsystem aufnehmen zu können, ist einerseits an eine Optimierung der Zusammenarbeit der Modelle zu denken, andererseits an die Einführung von Modulähnlichkeiten.

Bestehende Arbeiten für amerikanische Universitäten Die dem Projektteam bekannten, wissenschaftlichen Dokumente beziehen sich auf amerikanische Universitäten und deren Modulsystem. Der Studiumsaufbau ist zwischen den Hochschulen der beiden Länder vergleichbar. Für die Implementierung des Modul Recommender für HSR Studenten konnte daher auf den bestehenden Arbeiten aufgebaut werden, unter Berücksichtigung der vorhandenen Unterschiede der Systeme. Diese sind unter anderem auf den Grössenunterschied zwischen der HSR und den amerikanischen Universitäten zurückzuführen. Durch diese Arbeit konnten erste spezifische Erkenntnisse in Bezug auf die Implementierung eines Modul Recommenders für eine kleinere Schweizer Hochschule gewonnen werden.

11 Zusammenfassung

Der vorliegende Prototyp des Modul Recommenders für HSR Studenten arbeitet mit verschiedenen Modellen, wovon ein Collaborative Filtering Modell die Grundlage bildet. Durch das Finden von Studenten mit ähnlichen Studienverläufen wird eine erste Modulauswahl ermittelt. Diese Liste wird mit weiteren Modellen optimiert und gefiltert. Im letzten Schritt wird durch ein Graphenmodell berechnet, welche Module für die Erreichung der minimalen ECTS-Punktzahl pro Kategorie dienen. Ein einfaches Konsoleninterface dient zur Demonstration der Abfrage einer Modulempfehlung für einen Studenten.
Zusammengefasst fliessen in die Berechnung folgende Faktoren ein:

- Bisheriger Verlauf des Studiums
- Frühere Modulbesuche anderer Studenten
- Voraussetzungen für einen Modulbesuch
- Modulangebot im Semester, für welches eine Empfehlung berechnet wird

Die erarbeitete Lösung hat die zu Beginn festgelegten Ziele erreicht. Mit dem Prototyp ist es möglich, für einen Informatikstudenten der HSR für das nächste Semester eine Empfehlung zu erstellen. Dazu werden die historischen Daten von 440 Informatikstudenten der HSR analysiert und ausgewertet. Die daraus gewonnenen Empfehlungen können Studenten bei der optimalen Auswahl von Modulen unterstützen.

12 Ausblick

Der entwickelte Prototyp bietet ein stabiles Grundsystem für die Weiterentwicklung und Erweiterung des Modul Recommender für HSR Studenten. Während der Erarbeitung der Modelle entstanden Ideen für verschiedene Optimierungs- und Erweiterungsmöglichkeiten. Diese Möglichkeiten sind beinahe unbegrenzt. Die nachfolgenden Auflistungen haben daher keinen Anspruch auf Vollständigkeit.

Optimierungen für die bestehenden Modelle

- Verbesserung der Berechnung der Ähnlichkeit zweier Studenten.
- Noten der Studenten einbeziehen. Einerseits bei der Berechnung der Ähnlichkeit zweier Studenten, andererseits bei der Modulempfehlung selber.
- Minimalpunktzahl als Voraussetzung für einen Modulbesuch (z.B. für Semesterarbeit) berücksichtigen.
- Die aktuell belegten Module als bestanden annehmen, damit diese bei der Prüfung der Voraussetzungen bereits berücksichtigt werden.
- Falls der *CategoryGraph* nicht genügend Module empfehlen kann, um die Minimalpunktzahl für eine Kategorie zu erreichen, sollte eine entsprechende Meldung ausgegeben werden.
- Über das Konsoleninterface wird die Empfehlung für eine bestimmte Modulkategorie abgefragt. Besser wäre es, eine Empfehlung für alle Kategorien zusammen zu liefern.

Mögliche Erweiterungen des Prototyps

- Modulempfehlung über mehrere Semester.
- Empfehlung von Prüfungsbesuchen.
- Gütemass für die Prüfung einer Empfehlung. Könnte zum Vergleichen verschiedener Algorithmen verwendet werden.
- Die Anzahl der empfohlenen Module an der Belastung, welche ein Student wünscht, anpassen.
- Weitere Auswertung der Informationen zu Modulähnlichkeiten und Abhängigkeiten, welche im HSR Informatiksystem für die Modul- und Stundenplanverwaltung (Adunis) vorhanden sind. Besonders interessant ist die Zuordnung "verwandt". Anhand dieser Information könnten neue Module, welche bisher noch von keinem Studenten belegt wurden, einfach einbezogen werden. Die Abhängigkeiten "empfohlen" und "erforderlich" werden als Modulvoraussetzungen bereits berücksichtigt.

EId	EWert	Anzeige_D
1001	entspricht	Übergangsmodul entspricht Kurs
1002	wirdzu	Modul wird zu Übergangsmodul
1003	haengtAn	Übergangsmodul hängt an Modul
1004	istZugeteilt	Übertrittsmodul ist Modul zugeteilt
1005	erforderlich	erforderlich
1006	empfohlen	empfohlen
1007	aequivalent	äquivalent
1008	verwandt	verwandt (Versionen)
1009	anschliessend	Anschlussmodul

Abbildung 10: Modulabhängigkeiten im HSR Informatik System

- Anwendung und Anpassung auf andere Studiengänge der HSR.
- Integration des Recommender-Systems in die Strukturen der HSR.

13 Persönliche Berichte

13.1 Léonie Fierz

Das Themengebiet "Recommender Systems"der Studienarbeit ist sehr vielseitig. Die Einarbeitung in das für mich neue und spannende Gebiet wurde ergänzt durch den parallelen Besuch des Moduls "Challenge Projekte". In diesem Modul befassten wir uns mit der Anwendung und Entwicklung verschiedener Algorithmen für die Empfehlung von Filmen auf dem Dataset von Netflix. Dadurch ergab sich die Möglichkeit, mich vertieft in die Thematik einzuarbeiten. Besonders interessant war es, die Gemeinsamkeiten und Unterschiede der beiden Anwendungsgebiete von Recommender Systemen vergleichen zu können.

Themenwahl Schon früh legten wir uns fest, unsere Arbeit im Gebiet Recommender Systems zu schreiben. Der Spielraum für ein konkretes Recommender System ist gross. Unser Anspruch war es, ein nützliches, interessantes und doch überschaubares System entwickeln zu können. Aus den verschiedenen entstandenen Ideen, bei welchen teilweise die Datenbeschaffung gar nicht oder nur schwer möglich gewesen wäre, entschieden wir uns für einen Modul Recommender für HSR Studenten.

Zeitplanung Die Arbeit beinhaltete neben der Softwareentwicklung hauptsächlich die Einarbeitung in das Themengebiet und die Entwicklung und Ausarbeitung der verschiedenen Modelle. Aus diesem Grund konnten wir zu Beginn der Arbeit keine genaue Zeitplanung vornehmen. Wir beschränkten uns auf die Festlegung von 5 Meilensteinen. Leider fiel unser Betreuer durch einen Unfall einige Wochen aus, weshalb sich die Datenbeschaffung verzögerte und damit die geplanten Meilensteine vollkommen durcheinander gerieten. Wir entschlossen uns vorerst anhand von manuell generierten Testdaten die Modelle zu entwickeln. Nach dem Erhalt der Daten in Woche 11 und deren Import und Anbindung Stand der grosse Test an, ob die Modelle den richtigen Daten standhalten können - sie funktionierten erwartungsgemäss!
Rückblickend kann ich sagen, dass die Einhaltung einer groben Planung eine gewisse Sicherheit geben würde. Zudem wäre ein Arbeitsprotokoll hilfreich, um den Ablauf und Aufwand verfolgen zu können. Dank der guten Kommunikation im Team war eine agile Planung jedoch möglich und erfolgreich, sodass wir einen funktionierenden Prototypen und die Dokumentation fristgerecht fertigstellen konnten.

Teamarbeit Die Arbeit im Team mit Corsin Camichel war für mich sehr wertvoll. Besonders schätzte ich es, Entscheidungen im Bereich des Projektmanagements gemeinsam fällen zu können und bei der Programmierung und Entwicklung der Algorithmen Ideen zusammen zu besprechen und auszuarbeiten zu können.

In der gesamten Studienarbeit gewährte uns unser Betreuer Prof. J. Joller viel Freiheit, sei es in Bezug auf die genaue Festlegung des Arbeitsinhalts, der Zeitplanung oder der Erarbeitung der Modelle und Algorithmen. Diese Eigenverantwortung war motivierend und lehrreich. Ich freue mich die Arbeit im Rahmen der Bachelorarbeit, betreut von Prof. J. Joller, zusammen mit Corsin Camichel weiterführen zu können. Dies bietet uns die Möglichkeit, die Modelle zu optimieren und die entstandenen, noch nicht realisierten Ideen umzusetzen.

13.2 Corsin Camichel

Bereits ziemlich früh war für mich klar, dass ich meine Studienarbeit gerne bei Prof. Josef Joller machen möchte. Einer der Gründe lag darin, dass ich Prof. Joller als Mensch und als Dozent äusserst schätze. Auch habe ich mir erhofft, möglichst viele Freiheiten zu haben, sei es bei der Einreichung einer eigenen Idee wie auch bei der Umsetzung. Diese Punkte wurden vollständig bestätigt und es war insgesamt eine grosse Freude mit Herrn Joller aber auch mit Frau Léonie Fierz zu arbeiten.
Bereits vor der eigentlichen Ausschreibung über das AVT habe ich mich mit Frau Fierz und Herrn Joller besprochen, in welche Richtung unsere Studienarbeit gehen soll. Mich haben grosse Datensätze, komplizierte Systeme und Lösungen mit Algorithmen immer schon fasziniert. Deswegen war ich für mich auch klar, dass ich mich für das parallel stattfindende Challenge Projekt Modul anmelde. Aus diesem Grund haben wir uns dann auch eine Studienarbeit ausgedacht, welche in einem ähnlichen Gebiet angesiedelt ist. Zusätzlich war mir auch wichtig, etwas zu erstellen, was später auch verwendet wird.
Dank desselben Themengebietes von Challenge Projekt und Studienarbeit mussten wir uns in die allgemeinen Algorithmen und Vorgehensweisen nur einmal einlesen.

Erste Projektwochen In den ersten Wochen stand das Einlesen und Kennenlernen der Algorithmen im Vordergrund. Es gab viele Arbeiten von Universitäten und Konferenzdokumente zu lesen.
Leider verunfallte Prof. Joller in der zweiten Semesterwoche. Zu diesem Zeitpunkt und einige Wochen darüber hinaus stand unser Projekt auf der Kippe. Niemand wusste genau, was mit Herrn Joller passiert ist und wann er wieder einsatzfähig ist. In dieser Zeit haben wir uns mit anderen Dozenten unterhalten, um an die nötigen Datensätze zu kommen. Uns wurde jedoch von den Dozenten immer wieder mitgeteilt, dies sei nicht möglich und ob wir nicht eine andere Richtung für unsere Studienarbeit einschlagen möchten. Da wir das jedoch nicht wollten, haben wir uns nach Absprache mit Prof. Joller dazu entschlossen, einen Prototypen zu entwickeln und diesen mit einigen wenigen, selbst definierten Daten zu betreiben.

The Computer Says "Yes" Bereits nach sehr kurzer Zeit war es möglich, eine erste, vom System erstellte Empfehlung auszugeben. Zu diesem Zeitpunkt mussten wir definieren, wie das System im Einzelnen aussieht (einzelne Algorithmen definieren) sowie die Interkommunikation zwischen den Komponenten definieren.
Am 25. November 2009 erhielten wir endlich die gewünschten Daten der Modulbesuche und konnten unser bis anhin "dummes" System mit echten Daten testen. Ausserdem wurden zum ersten Mal alle Komponenten zusammengeschaltet. Das klappte besser als ich es jemals gedacht habe. Ich sehe hier drin den Beweis für eine ausgezeichnete Zusammenarbeit zwischen Léonie Fierz und mir. Innert weniger Tage wurde nun das System auf den Stand eines brauchbaren Prototypen gebracht. Bereits kurze Zeit später war eine minimale Applikation verfügbar, welche einem Benutzer die Möglichkeit bietet, das System zu bedienen.

Schlussfolgerungen Neben all diesen erfreulichen Geschehnissen gab es auch einige Punkte, welche ich mir anders vorgestellt habe oder ich verbessern würde. Zuerst hätte ich mir gewünscht, dass der nötige Datenbestand früher verfügbar gewesen wäre. Es war oft nicht sicher, wie gut ein Algorithmus funktioniert mit nur vier einfachen Studenten im System. Auch hätten wir früher die Datenbank ins System integrieren müssen.
Etwas mühsam waren die ersten drei Wochen nach dem Unfall von Herrn Prof. Joller. Niemand wusste genau Bescheid oder konnte uns konkret Auskunft geben. Durch direkten Kontakt mit

Herrn Joller waren wir zwar etwas informiert, aber dennoch war es schwierig abzuschätzen, wie wir weiter machen sollen oder ob wir uns von einem anderen Dozenten betreuen lassen sollen.
Im Gegensatz zu anderen Projektarbeiten, sei es im Studium oder früher als Angestellter, hatte ich dieses Mal keine Durchhänger oder langweilige Phasen. Immer gab es etwas Neues auszuprobieren, zu lesen oder zu testen. Dies schreibe ich dem äusserst interessanten Thema aber auch den Freiheiten, die uns Herr Joller eingeräumt hat, zu.
Selbstverständlich habe ich noch viele Ideen, von denen ich hoffentlich einige in einer späteren Arbeit umsetzen kann.

Ergebnis Ich bin mit dem jetzigen Stand der Arbeit sehr zufrieden und es freut mich, dass ich das System zusammen mit Frau Fierz in meiner Bachelor-Arbeit erweitern darf. Auch freut es mich, dass gemäss Aussage von Herrn Joller die Hochschule sehr an diesem System interessiert ist. Es ist ein spannendes und äusserst aktuelles Thema. Weiter bietet es Studenten eine Hilfestellung und wird ihnen hoffentlich einen Mehrwert bieten.
Ich habe in dieser Arbeit viel gelernt. Nicht nur in Bezug auf die Entwicklung des Systems, sondern auch über die Arbeiten um das System herum. Sei es Bewerten und Evaluieren von fachlichen Publikationen oder Integration von verschiedenen Komponenten.

14 Glossar

Begriff	**Erklärung**
Adunis	Studenten- und Modulverwaltungssystem der *HSR*
Algorithmus	Lösungsverfahren und Auswertung der in einem *Modell* abgebildeten Problematik
Ähnlich/Ähnlichkeit	In Bezug auf Studenten: Studenten deren Attribute wie Semester, Notenschnitt oder die *Modul-Geschichte* gut übereinstimmen. Im Modul Recommender für HSR Studierende wird die Ähnlichkeit zweier Studenten nur über die *Modul-Geschichte* definiert.
Bachelor	Internationaler Hochschulabschluss; Angeboten an Hochschulen und Universitäten
Bologna-Modell	Vorhaben zur Vereinheitlichung des europäischen Hochschulwesens
Breath First Picking	Algorithmus zur Findung von Knoten in einer Reihenfolge
CBR	*Content Based Reasoning*
Content Based Reasoning	Filtermethode ähnlich zu Collaborative Filtering
Collaborative Filtering	Recommendation Algorithmus basierend auf der Suche nach ähnlichen Objekten oder Benutzer
Data Access Object	Erlaubt den Zugriff auf unterschiedliche Arten von Datenquellen durch Kapslung[17].
Dijkstra Algorithmus	Algorithmus zur Berechnung des kürzesten Pfad zwischen zwei beliebigen Knoten in einem Graph
ECTS	European Credit Transfer System; Mass zur Bewertung von *Modulen*
Framework	Programmgerüst; Vereinfacht und unterstützt die Entwicklung von Softwaresystemen.
Hibernate	Ein standardisiertes Framework zur Kommunikation mit einer Datenbank aus einer Java-Applikation
HSR	Hochschule für Technik Rapperswil
Java	Objektorientierte Programmiersprache
JUNG	Java Universal Network/Graph Library; Eine offene Java-Library für Graphendarstellung und -berechnung
kNN	k-Nearest-Neighbors; Algorithmus zur Bestimmung der k-Ähnlichsten Objekten/Personen
Lazy Loading	Pattern nach [8]; Daten sollen erst geladen werden, wenn diese benötigt werden
Leistungssemester	Einen gemäss ECTS-Punkte definierten Studiumsfortschritt.
Library	Sammlung von Code
Modell	Abstraktion einer Problematik
Map	Datenstruktur in welcher zu Schlüsseln Werte abgelegt werden können. Die Schlüssel sind einmalig.
Mockobjekte	Statische Nachbildung von Objekten für *Unit-Tests*
Modul	Eine Vorlesung an der Hochschule

[17] `http://de.wikipedia.org/wiki/Data_Access_Object`, zuletzt besucht am 15.12.2009

Begriff	**Erklärung**
Objekt	Beispielsweise ein Buch, ein Musikstück, ein *Modul*, ein Weblink
Objektrelationales Mapping	Erlaubt in einer objektorientierten Programmiersprache die Objekte in eine relationale Datenbank zu speichern.
Pre-Recommendation	Von einem Modell/Algorithmus berechnete Modulempfehlung mit Gewichten; Dient als Input für andere Modelle
Recommender System	Empfiehlt anhand Algorithmen und Modellen eine Auswahl an passenden oder ähnliche Objekte
Singular Value Decomposition	Matrixfaktorisierung
Studenten	Studenten und Studentinnen
Studierende	*Studenten*
Subversion	Tool zur Verwaltung und Versionierung von Code.
SVD	*Singular Value Decomposition*
Unit Test	Testcode der eine einzelne Einheit unabhängig von der gesamten Applikation testet.

15 Literaturverzeichnis

[1] C. Furrer, Muster einer Studentenerklärung, http://www.hsr.ch, 2009.

[2] J. B. Schafer, J. Konstan, and J. Riedl, Recommender systems in e-commerce, 1999.

[3] A. Das, C. Mathieu, and D. Ricketts, CoRR **abs/0908.3633** (2009).

[4] J. J. Sandvig and R. Burke, DePaul University Report No. 05-015, 2005 (unpublished).

[5] Stanford University Report No., , 2009 (unpublished).

[6] A. Parameswaran and H. Garcia-Molina, Recommendations with prerequisites, in *ACM Conference on Recommender Systems*, Stanford InfoLab, 2009.

[7] L. Suhl and T. Mellouli, *Optimierungssysteme: Modelle, Verfahren, Software, Anwendungen (Springer-Lehrbuch)* (Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007).

[8] M. Kirchner and P. Jain, *Pattern-Oriented Software Architecture: Patterns for Resource Management*, Wiley Series in Software Design Patterns Vol. 3 (John Wiley & Sons, 2004).

16 Abbildungsverzeichnis

A Anhang

A.1 SQL Skripte

Im Ordner *sql/* sind alle für den Datenbankaufbau und Datenimport nötigen SQL-Skripte enthalten.

Tabellenstruktur In einem ersten Schritt ist es nötig, anhand der *createTable**-Skripte die Datenbankstruktur zu erstellen.

Import Nachfolgend sind die Skripte für den Import der Daten aus dem Informatiksystem für die Studenten- und Modulverwaltung der HSR (Adunis) aufgelistet. Die Daten liegen in der 1. Normalform vor und werden auf die Tabellenstruktur des Modul Recommenders für HSR Studenten konvertiert.

- insertStudentsQuery.sql: Löscht Diplomstudenten, importiert Studenten und modifiziert Übergangs-Module
- insertCoursesQuery.sql: Import von Category, Course, CourseHistoryEntry, Enrollements und deren Abhängigkeiten
- insertRequirements.sql: Import für Module-Voraussetzungen

A.2 Package ch.modulerec.algorithms

Package Contents *Page*

Interfaces

INTERFACE **IAlgorithm**

Interface for any kind of Recommendation Algorithms. Implementations of IAlgorithms can be cascaded in any order by setting the output of `Map<Course, Double>run()` as a Pre-Recommendation (`setPreRecommendation(Map<Course, Double>preRecommendations)`) for the following algorithm.

METHODS

`public String` **info**`( )`

- **Usage**
  - Method to get a String message which explains what the algorithm or filter does. Useful to explain why a module has not been recommended or a general explanation of the chain of filters/algorithms.
- **Returns** - A message as string

`public Map` **run**`( )`

- **Usage**
  - Run the algorithm to calculate the course recommendation. Should check if all necessary information has been set by the client. The returned course list represents the recommendation calculated by this algorithm. Courses have a weight associated with a value between 0 (not recommended) and 10 (highly recommended) If the algorithm just calculates if a course is wise to take or not and does not calculate a differentiated weight it should pass through the weight associated with the pre-recommendation, if one exists. Return value can be used as a pre-recommendation for another algorithm.
- **Returns** - Map<Course, Double>with recommended courses

`public void` **setPreRecommendation**`( Map` **preRecommendations** `)`

- **Usage**
  - Set courses already recommended by another algorithm. Each course should have an weight associated with a value between 0 (not recommended) and 10 (highly recommended). The map can be empty, e.g. if it is the first algorithm run.

`public void` **setStudent**`( Student` **student** `)`

- **Usage**
  - Set the student for which to get course recommendations

A.3 Package ch.modulerec.algorithms.categorygraph

Package Contents *Page*

Classes

Classes

Class **CategoryGraph**

Recommends courses for a student based on the "Max Flow Min Cost" Algorithm. The goal is to recommend courses such that all category requirements (minimal ECTS-points per category) are fulfilled. As input the algorithm needs all courses visited by the student and all courses he could possibly visit. To get more accurate recommendation the not jet visited courses can be weighted using another algorithm, e.g. some collaborative filtering algorithm. The weights can then be passed to the "Max Flow Min Cost" Algorithm, which sets the costs according to these weights.

extends ch.modulerec.algorithms.BaseAlgorithm

Fields

- private static double MAX_COST
- private MaxFlowMinCostAlgorithm maxFlow
- private HashMap flows
- private DirectedGraph graph
- private Node startNode
- private Node sinkNode

Constructors

`public` **CategoryGraph**`( Student s )`

`public` **CategoryGraph**`( Student s, Map` **preRecommendations** `)`

`public` **CategoryGraph**`( Student s, Map` **preRecommendations**`, boolean` **considerAllCourses** `)`

- **Parameters**
  - `s` - Student for which to get the recommendation
  - `preRecommendations` - list of modules with associated weights
  - `considerAllCourses` - take all courses to make recommendation instead of only considering courses out of the pre-recommendation list.

METHODS

`private Node` **findNode**`( Node n, Graph g )`

`public HashMap` **getFlows**`( )`

- **Returns** - HashMap representing the flows calculated for each edge. Available after calling recommend().
- **See Also**
  - `ch.modulerec.algorithms.categorygraph.CategoryGraph.recommend()`

`public DirectedGraph` **getGraph**`( )`

- **Returns** - the graph for which the max flow min cost algorithm is calculated

`public int` **getMaxFlow**`( )`

- **Returns** - The total flow calculated through the graph. Available after calling recommend().
- **See Also**
  - `ch.modulerec.algorithms.categorygraph.CategoryGraph.recommend()`

`private Map` **getRecommendedCourses**`( )`

`public Node` **getSinkNode**`( )`

- **Returns** - the sink/end node of the graph

`public Node` **getStartNode**`( )`

- **Returns** - the start node of the graph

`public String` **info**`( )`

`private DirectedGraph` **initializeGraph**`( Map` **preRecommendations** `)`

`public Map` **recommend**`( )`

- **Usage**
  - Runs "Max Flow Min Cost" algorithm. The result flows for each edge are available in the HashMap returned by {@Code getFlows()}
- **Returns** - the total flow calculated
- **See Also**
  - `ch.modulerec.algorithms.categorygraph.CategoryGraph.getFlows()`
  - `ch.modulerec.algorithms.categorygraph.CategoryGraph.getMaxFlow()`

`public Map` **run**`( )`

Class **MaxFlowMinCostAlgorithm**

Implements the Edmonds-Karp maximum flow algorithm for solving the maximum flow problem. After the algorithm is executed, the input {@code Map} is populated with a {@code Number} for each edge that indicates the flow along that edge.
Extended by lfierz:
Added min cost functionality. Convertet to use our Node and Edge implementation instead of a generic type. Generizity is not needed, though it would add unnecessary complexity.

An example of using this algorithm is as follows:

```
EdmondsKarpMaxFlow ek = new EdmondsKarpMaxFlow(graph, source, sink, edge_flows);
ek.evaluate(); // This instructs the class to compute the max flow
```

extends edu.uci.ics.jung.algorithms.util.IterativeProcess

Fields

- private static boolean DEBUG_AUGMENTING_PATH
- private DirectedGraph flowGraph
- private DirectedGraph originalGraph
- private Node source
- private Node target
- private int mMaxFlow
- private Map residualCapacityMap
- private Map parentMap
- private Map parentCapacityMap
- private Map edgeFlowMap
- private Factory edgeFactory

Constructors

`public` **MaxFlowMinCostAlgorithm**`(` `DirectedGraph` **directedGraph**`,` `Node` **source**`,` `Node` **sink**`,` `Map` **edgeFlowMap** `)`

- **Usage**
  - Constructs a new instance of the algorithm solver for a given graph, source, and sink. Source and sink vertices must be elements of the specified graph, and must be distinct.
- **Parameters**
  - `directedGraph` - the flow graph
  - `source` - the source vertex
  - `sink` - the sink vertex
  - `edgeFlowMap` - the map where the solver will place the value of the flow for each edge

METHODS

`private void` **addBackEdgeToFlowGraphFor**`( Edge` **edge** `)`

`private boolean` **checkForAugemntingPath**`( )`

`private void` **checkSourceSink**`( DirectedGraph` **directedGraph**`, Node` **source**`, Node` **sink** `)`

`private void` **clearParentValues**`( )`

`private void` **copyToFlowGraph**`( DirectedGraph` **directedGraph** `)`

`protected void` **finalizeIterations**`( )`

`public DirectedGraph` **getFlowGraph**`( )`

- **Usage**
  - Returns the graph for which the maximum flow is calculated.

`public int` **getMaxFlow**`( )`

- **Usage**
  - Returns the value of the maximum flow from the source to the sink.

`protected boolean` **hasAugmentingMinCostPath**`( )`

`protected void` **initializeIterations**`( )`

`private void` **initQueueWithGraphNodes**`( PriorityQueue` **prioQueue** `)`

`private void` **printFlowGraphNodes**`( )`

`private void` **relaxEdge**`( Node` **originVertex**`, Integer` **originVertexCapacity**`, Edge` **neighboringEdge**`, Integer` **edgeResidualCapacity**`, PriorityQueue` **prioQueue** `)`

`private void` **setEdgeFlowMap**`( )`

`public void` **step**`( )`

`private void` **tidyUpGraph**`( )`

`private void` **updateResidualCapacities**`( )`

`private void` **updateShortesPath**`( Node` **originVertex**`, Integer` **originVertexCapacity**`, Integer` **edgeResidualCapacity**`, Node` **neighboringVertex** `)`

A.4 Package ch.modulerec.algorithms.collaborativefiltering

Package Contents *Page*

Interfaces

INTERFACE **IFilter**

Interface for the collaborative filtering algorithms in the module recommender.

implements ch.modulerec.algorithms.IAlgorithm

METHODS

`public void` **setCategory**`( Category` **category** `)`

- **Usage**
  - Category for which to recommend courses.
- **Parameters**
  - `category` -

`public void` **setSemester**`( int` **semester** `)`

- **Usage**
  - Semester for which to recommend courses.
- **Parameters**
  - `semester` -

`public void` **setStudents**`( List` **students** `)`

- **Usage**
  - List of Students which can be used as a history to search for neighbors.
- **Parameters**
  - `students` -

Classes

Class **BaseFilter**

extends ch.modulerec.algorithms.BaseAlgorithm
implements IFilter

Constructors

`public` **BaseFilter**`( )`

Class **DistanceFilter**

Distance filter seeks for a list of similar modules. This list is generated on a number of neighbors and calculated with a K-Nearest-Neighbors algorithm.

extends ch.modulerec.algorithms.collaborativefiltering.BaseFilter

Fields

- private final int MAX_NEAREST_NEIGHBORS
- private List allKNeighbors
- private TreeMap allNeighbors
- private final DecimalFormat DOUBLE_FORMAT
- private final int MAX_ADMITTANCE_YEAR_DELTA
- private final double MIN_DISTANCE_VALUE
- private List workStudents

Constructors

`public` **DistanceFilter**`( )`

Methods

`public double` **calcCustomDistance**`( Student` **other**`, Category` **category** `)`
`private HashMap` **createWeightedCourses**`( Set` **courses** `)`
`private void` **fillKNeighborsList**`( )`
`private void` **fillNeighborsList**`( )`
`private Set` **getCoursesFromKNeighbors**`( )`
`public String` **info**`( )`
`private Set` **removeBlockedCourses**`( Set` **courses** `)`
`public Map` **run**`( )`

Class **RequirementsFilter**

Filter that checks if the student has all required courses for the recommended courses. If a course is found where not all requirements have met, the course is removed.

extends ch.modulerec.algorithms.collaborativefiltering.BaseFilter
implements IFilter

Class **SemesterFilter**

Filter that returns a list of courses that are offered in the requested semester.

extends ch.modulerec.algorithms.collaborativefiltering.BaseFilter