

Flexible Recommendations over Rich Data

Georgia Koutrika, Robert Ikeda, Benjamin Bercovitz, Hector Garcia-Molina
Computer Science Department, Stanford University
California, USA
{koutrika, berco}@stanford.edu,
{rmikeda, hector}@cs.stanford.edu

ABSTRACT

CourseRank is a course planning tool aimed at helping students at Stanford. Recommendations comprise an integral part of it. However, implementing existing recommendation methods leads to fixed recommendations that cannot adapt to each particular student's changing requirements and do not help exploit the full extent of the available learning opportunities at the university. In this paper, we describe the concept of a flexible recommendation workflow, i.e., a high-level description of a parameterized process for computing recommendations. The input parameters of a flexible recommendation process comprise the “knobs” that control the final output and hence generate flexible recommendations. We describe how flexible recommendations can be expressed over a relational database and we present our prototype system that allows defining and executing different, fully-parameterized, recommendation workflows over relational data. Finally, we describe a user interface in *CourseRank* that allows students customize recommendations.

Categories and Subject Descriptors

H.3.3 [Information Systems]: Information Search and Retrieval; H.3.4 [Information Storage and Retrieval]: Systems and Software; H.5.2 [Information Interfaces and Representation]: Graphical user interfaces (GUI)

General Terms

Design, Algorithms, Human Factors

Keywords

flexible recommendations, workflows, CourseRank

1. INTRODUCTION

Recommendation systems, such as Google News [11], Amazon [16] and MovieLens [18], are popping up everywhere, to provide advice on movies, travel, and leisure activities.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

RecSys'08, October 23–25, 2008, Lausanne, Switzerland.
Copyright 2008 ACM 978-1-60558-093-7/08/10 ...\$5.00.

The systems help users sort through the huge amounts of available data, and receive content and services that are of interest to them. Since the appearance of the first recommendation systems [12, 20, 22], many approaches have been proposed both by the industry and academia. However, most recommendation methods *perform on the basis of assumptions ‘hard-wired’ into the system and they support only a predefined and fixed set of recommendations which may not always capture the real-time user information needs* [5]. Giving the end-user the ability to customize their recommendations may be very important in order to provide more accurate, personalized recommendations. For example, *CourseRank* is a course planning tool for Stanford University students that helps students choose courses. Recommendations comprise an integral part of the system. Providing only fixed recommendations does not help build personalized plans that are customized to each particular student's changing requirements and exploit the full extent of the available learning opportunities the university offers.

1.1 The CourseRank System

Traditionally, university students base their schedules on word-of-mouth knowledge and the brief course descriptions found in bulletins or academic guides, rather than on more comprehensive assessments. *CourseRank* [1], under development in Stanford's Infolab, is a social tool for course planning that helps students make informed choices and take advantage of the available learning options. It displays official university information and statistics, such as bulletin course descriptions, grade distributions, and results of official course evaluations. Furthermore, students can anonymously rank courses they've taken, add comments, and rank the accuracy of each others' comments. They can also shop for classes, get personalized recommendations, and organize their classes into a quarterly schedule or devise a four year plan. *CourseRank* also functions as a feedback tool for faculty and administrators, ensuring that information is as accurate as possible. Faculty can also modify or add comments to their own courses, and can see how their class compares to other classes. To support this functionality, *CourseRank* maintains a database that stores rich information, such as the courses offered, the instructors, the students, comments and ratings given by the students to courses and instructors, course material, and so forth. Figure 2 provides a small snapshot of the database schema.

A little over a year after its launch, *CourseRank* is already used by more than 6,200 Stanford students, out of a total of about 14,000 students. The vast majority of *CourseRank* users are undergraduates, and there are only about



Figure 1: Classical recommendation paradigms.

Departments(DepID, DepCode, Name)
 Courses(CourseID, DepID, Title, Description, Units, Url)
 CourseSched(CourseID, Year, Term, InstrID, Location, TimeSlot, Days)
 Instructors(InstrID, Name, Url)
 Students(StudID, Name, Class, GPA)
 StudentStudies(StudID, StudyPrgID)
 StudyPrograms(StudyPrgID, ProgramName, Classification, DepID)
 StudentHistory(StudID, CourseID, Year, Term, Grade, Rating)
 Comments(StudID, CourseID, Year, Term, Text, Rating, Date)

Figure 2: Extract from *CourseRank*'s database

7,000 undergraduates at Stanford. Thus, it is safe to say that *CourseRank* is already used by a very large fraction of Stanford undergraduates.

Supporting Recommendations. Existing recommendation approaches can be categorized on the basis of how recommendations are generated: (a) *content-based* methods recommend to the user items similar to the ones the user preferred in the past, and (b) *collaborative filtering* recommend to the user items that people with similar preferences liked in the past.

Following a *content-based approach* in *CourseRank*, we could consider that each course is represented by the set of topics that it covers (`Courses.Description`), and each student is represented by a list of the topics from the courses she has attended. When a student, say Alice, logs into the system, the system matches courses, based on their topics, to the topics that interest this student. The result would be a list of courses, each one with a score showing how closely the course matches Alice's interests and the top courses would comprise the system's recommendations. Standard functions for computing such scores include the cosine similarity, the Jaccard metric, and so forth. A high-level representation of this recommendation process is illustrated in Figure 1(a).

Adopting a *collaborative approach*, we could take into account the ratings students assign to courses (`Comments.Rating`). In order to recommend courses to Alice, the system will first find students that are similar to Alice, i.e., with similar ratings for the same courses. The output of this first step is a list of (top N) users ordered on their similarity scores. A typical function for computing user similarity is the Pearson correlation [20]. Then, these users collectively determine the ratings for the courses not taken by Alice. For example, course ratings could be computed as the weighted average of the students' ratings for this course. Figure 1(b) illustrates this recommendation process, which has been implemented in *CourseRank*. Figure 3 shows a screen from *CourseRank* that displays recommendations based on this approach.

Limitations. Current recommendation methods operate on a set of fixed assumptions that shape the desired recommendations but the user cannot specify or modify any of them. Below, we highlight these assumptions and give examples from *CourseRank* of how they may generate inaccurate or inflexible recommendations for the students.

- *The system always assumes that the target of the recommendations is the current user.*

Recommendations explicitly (content-based approach) or implicitly (collaborative approach) match this user's rep-

resentation in the system. For example, if a student has attended many Computer Science courses, a content-based recommendation system can recommend other CS courses. However, if this student is also interested in taking some Sociology courses, the system cannot generate any recommendations for such courses that would match the system's past knowledge on the student (i.e., her past CS courses). In some cases, students may also want to see the recommendations the system would give their best friend, not themselves.

- *The system always assumes a fixed pool of courses, from which recommendations will be drawn.*

For example, in Figure 3, *CourseRank* provides a general list of recommendations for our hypothetical student, Alice, containing a course on Robotics and a course on Spanish language. Other interesting courses on Spanish have not surfaced in this general list of recommendations. Moreover, Alice cannot request recommendations targeted to Spanish language courses. Taking this one step further, since the database stores information about many entities apart from courses, a student may wish to ask for recommendations for different entities, such as instructors, course material for CS courses, etc.

- *In collaborative filtering, recommendations are based on the opinions of a fixed set of users.*

For example, in our initial *CourseRank* implementation, all students act as potential recommenders. In many cases, a student may not seek recommendations from the broad student community, because their shared interests may not be related to the user's current information need, but rather from a selective subset of experts w.r.t. to her current needs. For example, Alice would rather ask for recommendations on Spanish courses from Spanish Language students rather than any student in general, whereas for CS courses she is only interested in the opinions of her colleagues in Computer Science.

- *The system always assumes a fixed set of attributes for comparing courses or students.*

Courses and students are compared on a fixed set of their attributes, ratings in collaborative filtering and keywords in the content-based case. Since *CourseRank*'s database contains a lot of attributes that describe the various entities, assuming that comparisons are performed only on ratings or keywords (or anything that is predefined in the system) is very restricting. For example, Alice may want recommendations for CS courses from CS students with similar grades (i.e., with similar performance) and for dance classes from students with similar ratings (i.e., with similar tastes).

Consequently, supporting recommendations following one of the classical approaches allows *CourseRank* to offer only fixed, 'canned' recommendations that are generated on the basis of a number of assumptions hard-coded in the system. Little (if any) control is given to the users. The recommen-

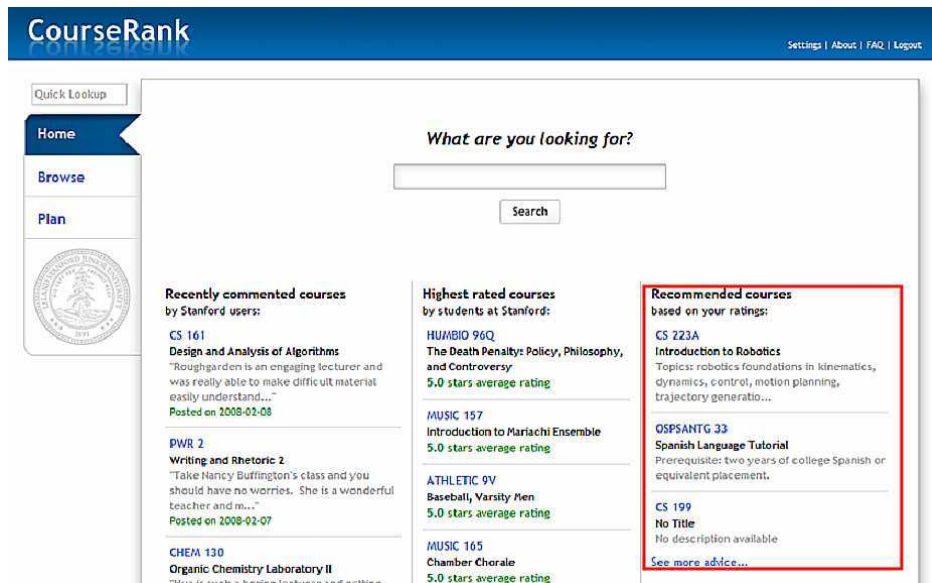


Figure 3: Fixed recommendations in *CourseRank*

dations offered comprise standard advice, which cannot accurately capture a student’s evolving information needs, and they may change when there is a significant change in the information the system keeps for the users or the courses offered. Furthermore, traditional recommendation approaches model the world as having two types of entities, users (e.g., students) and items (e.g., courses), represented as sets of ratings or features. Many systems rely on richer data models. For example, *CourseRank* stores structured data in a relational database. Providing recommendations for different entities (e.g., courses, instructors, etc) taking into consideration different attributes (e.g., grades, ratings, major, etc) is not straightforward. As a consequence of these limitations, although the initial version of *CourseRank* has been very popular with students (see editorial in the Stanford student paper [2]), we received many requests, from students and administrators, for more flexible recommendations.

In this paper, we propose *flexible recommendations*, i.e., recommendations that have a set of “knobs” that can be used to specify or “tune” the desired output of the recommendation process, and can be dynamically defined *over structured data*. To illustrate, in the same sense that a system can offer advanced search, where users can customize their queries by combining different parameters, conditions, and entities, a system should provide advanced recommendations, where users can tweak a number of “components” in order to explore different recommendations.

1.2 Contributions

In brief, the contributions of this paper are the following:

- We define the concept of a flexible recommendation workflow, i.e., a high-level description of a parameterized process for computing recommendations. The input parameters of a workflow allow one to control externally the final output of the recommendation process, and hence, generate *flexible recommendations* (Section 2).
- We describe how flexible recommendations can be expressed over a relational database (Section 3) and we present our prototype flexible recommendation engine

that allows defining and executing recommendation workflows over relational data (Section 4).

- We describe a preliminary user interface in *CourseRank* that allows students to make use of two flexible recommendation workflows, a content-based one and a collaborative filtering one, defined and executed with the help of the prototype system in order to shape the course recommendations provided by the system (Section 5).

2. FLEXIBLE RECOMMENDATIONS

We define a *flexible recommendation workflow* $RW(I, P)$ as a high-level description of a process for computing recommendations for a set of objects I (e.g., courses, books, etc) based on a set of input parameters P . A recommendation workflow comprises a series of interconnected operators that describe how a recommendation is computed. In a workflow, sets of objects flow from one operator to the next, and are filtered, ranked, etc. in the process. The workflow is a “high-level” description in the sense that it does not contain actual code, but rather, it describes what code (operators) to call upon. The input parameters P of a workflow are essentially operator inputs that are exposed outside the workflow in order to allow one to control externally the final output of the recommendation process, and hence, generate *flexible recommendations*.

There are several possible types of operators that one could define and combine in a recommendation workflow. For example, one could use filters in order to exclude objects, blend operators for combining sets of objects or recommendations, and top-k operators that output only the top k objects based on some attribute of them. The core operator of any recommendation workflow is a *recommend operator* that rates the objects of a set by comparing them to the objects of another set.

Figures 1(a) and 1(b) could actually be seen as two recommendation workflows, except that the inputs and parameters are hardwired by the system. Furthermore, in classical recommendation systems there is usually no high-level description; the recommendation processing is implemented in

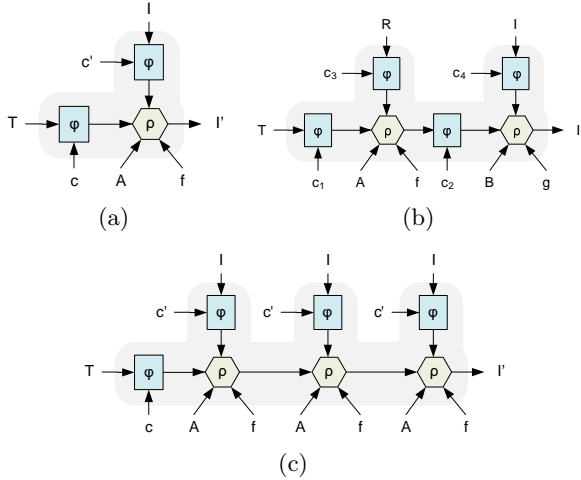


Figure 4: Flexible recommendation workflows.

low-level code that is hard to change. Below we illustrate some flexible recommendation workflows.

Example. Assume that we define two operators: (a) a filter operator $\phi(X, c)$ that takes as input a set of objects X and outputs only objects that meet the condition c , and (b) a recommend operator $\rho(X, Y, f, A)$ that rates the objects in X by comparing them to the objects in Y on some common attribute A with the help of a function f . These operators can be combined in several ways to build different flexible recommendation workflows. Figure 4 illustrates three example workflows. The workflow depicted in Figure 4(a) generates flexible recommendations for a set of objects I , which may be filtered based on some criterion c' , by comparing its objects with the objects of a set T on some common attribute A . Objects in T may be also filtered on some criterion. Consequently, this workflow represents a flexible content-based recommendation process with several parameters to shape the recommendations, i.e., $RW_{cont}(I, c', T, c, A, f)$. By dynamically changing its inputs, this process can provide different recommendations. For example, a student (i.e., T : students) with id ‘1432’ (c : id=‘1432’) could ask for recommendations for courses (I : courses) on programming (c' : “on programming”) whose topics match her topics of interest (A : topics). In similar way, she could ask for course recommendations for her friend (c') or for dance (c) classes (I). Going one step further, T could be a set of objects, such as a group of friends and function f could compute the average score for each course over all students. In this way, the system could provide recommendations for courses that would match the interests of a group of students that want to take a class all together.

Figure 4(b) shows a workflow that provides flexible recommendations for a set of objects I in a collaborative-filtering fashion. For this purpose, the objects in I are rated w.r.t. the objects of a set R , which have been already rated w.r.t. a set T . Consequently, this workflow represents a flexible recommendation process $RW_{coll}(I, c_4, R, c_3, B, g, c_2, T, c_1, A, f)$. Different recommendations can be obtained by changing the process inputs. For example, a student (T) named Alice (c_1) may want to see books (I) recommended based on the ratings (B) of students (R) with a CS major (c_3) that match with her on course ratings (A). Similarly, Alice could ask for sociology (c_4) courses (I) recommended based on the ratings

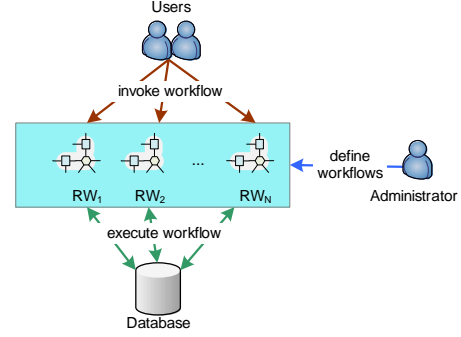


Figure 5: Flexible system workflow.

(B) of sociology (c_3) students (R) that have similar grades (A) with a friend student in this department (c_1).

It is also possible to define workflows that represent new types of recommendations, which can be tested using the same infrastructure. To illustrate, Figure 4(c) shows a sequence of interconnected recommend operators that compute friends-of-friends that have up to three degrees of separation. The first recommend operator computes the set of similar students to an individual based on a set of attributes A (e.g., ratings, grades, GPAs, etc). We call these students “friends” of the student. The second recommend operator takes as input the “friends” found by the first recommend operator and computes “friends” of them, and so on. Consequently, this workflow represents a different flexible recommendation process $RW_{friend}(I, c', T, c, A, f)$.

As illustrated in Figure 5, a *flexible recommendation system* contains a set of recommendation workflows, RW_1, RW_2, \dots, RW_N . These are defined by an administrator or designer using the set of operators that are supported by the system. At run time, users can invoke any of these workflows (through some user interface) with the set of objects, for which recommendations are sought, and the rest of the parameters that are required by the process in order to generate customized recommendations. The system then executes the process over the underlying data and returns a set of recommendations.

3. FLEXIBLE RECOMMENDATIONS OVER RELATIONAL DATA

Since most structured data used by production systems (including *CourseRank*) is stored in relational databases, it is desirable to have a workflow data model that is “close” to a relational model. If the objects that flow between operators are relations (i.e., sets of tuples), then we can use classical relational operators like joins, selections and projections to operate on the data. Reading and storing objects would be straightforward, since we would only need to read and write relations into the database. We would need to define new operators like recommend and blend, but they would also be defined over relations.

Unfortunately, relying on a “pure” relational data model for recommendation workflows is restrictive. The problem is that information about a single entity (e.g., student, course) may be dispersed over different relations due to database structuring and normalization. For example, we may want to compare students to the current user based on their ratings for the courses that they have in common. In *CourseRank*, as Figure 2 shows, there is a relation *Students*, where

each tuple corresponds to a single student while information about the courses each student has taken is stored in a different relation, *StudentHistory*. A recommend operator that compares students to the current user would need to join together multiple relations, making it cumbersome to define the operator. Ideally, we would like to represent our application entities with a single “extended” relation. For instance, a tuple in an extended relation could contain base information on a student (e.g., name, GPA), plus the set of courses a student has taken. In this way, our recommend operator would operate on attributes of entities, irrespective of whether these are base attributes, such as the GPA, or extended attributes, such as the sets of ratings.

On the basis of the above observations, we consider an *extended relational model*, that lets us represent our application entities with a single relation.

Extended relations. A relational database comprises a set of stored relations. A stored relation has a set of stored tuples described by a set of attributes. An attribute can be instantiated to a single, numerical or categorical, value. We define an *extended relation*, which has a set of extended tuples described by a set of stored attributes and a set of extended attributes. An *extended attribute* is instantiated to a relation derived from another stored relation. Consequently, under our semantics, an attribute value can be scalar or a relation. Note that only stored (flat) relations can be part of a tuple, allowing only one level of nesting. We think that one level of nesting is sufficient to capture the needs of most recommendation operators, so we avoid the complexities of a full-fledged nested relational model.

Extended relations can be therefore thought of as “views” that collect and group together information related to an individual entity and represent it as a single tuple that can be easily handled by other operators. For example, in a workflow, students may be extended with their courses, so that the set of courses for each student can be “viewed” as an additional attribute of the student by subsequent operators in the workflow irrespective of the database structure.

Extend operator. Since extended relations are not stored in the database, we define an *extend operator* that can be used in a recommendation workflow. This operator allows “extending” each tuple from one relation with the set of joining tuples from a different relation. In other words, for each tuple t in the first relation, the operator creates an extended attribute. The t attribute is instantiated to a relation containing tuples from another relation that join with t . For example, each student can be extended with an attribute that describes the courses she has taken. Similarly, a department can be extended with the courses it offers, etc.

Figure 6 shows an instance of the *Students* relation and an example of an extended relation, where the set of courses for each student is “viewed” as an additional attribute of the student. A recommend operator can now take as input this extended relation, and for each tuple generate a score indicating how well the student matches the target user (as discussed in the example of Section 2).

In the following section, we present our prototype flexible recommendation engine that allows defining and executing relational recommendation workflows.

4. FLEXIBLE RECOMMENDATION ENGINE

Our goal is to build an engine for flexible recommendation processing and optimization over relational databases.

| Students | StudID | Name | Class | Reputation |
|----------|--------|-------------|-------|------------|
| | 1 | Paul Little | 2009 | A |
| | 2 | John Doe | 2010 | A |

| Ext_Students | StudID | Name | Comment(CourseID, Rating, Date) | | |
|--------------|--------|-------------|---------------------------------|-----|-------------|
| | 1 | Paul Little | C1 | 5 | 2 Feb 2008 |
| | | | C2 | 6 | 3 Dec 2007 |
| | 2 | John Doe | C1 | 5 | 15 Mar 2007 |
| | | | C2 | 6 | 12 Dec 2007 |
| | | | C5 | 6.6 | 22 Jun 2007 |
| | | | C7 | 7 | 22 Jun 2007 |

Figure 6: Relation *Students* extended with courses.

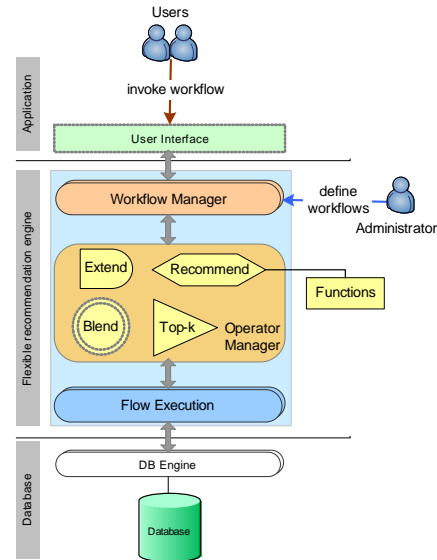


Figure 7: Prototype flexible recommendation engine

There are generally two options: (i) build a middleware layer, i.e., implement the processing outside the core of the database system or (ii) extend the database query engine with the processing capabilities required for flexible recommendations. We chose the middleware solution for portability and implementation ease. Figure 7 shows the architecture of our current *CourseRank* working system. It is implemented in Java on top of MySQL.

Workflow Manager. This component allows an administrator to define different recommendation workflows and end-users (through the user interface, see Section 5) to invoke any of the defined workflows with different inputs and receive customized recommendations. This component hides the details of how flexible recommendations are generated, such as the details of the algorithms and structures used to implement the functionality of the operators that comprise a recommendation workflow, the execution order of operators, which can be different from the workflow definition in order to optimize the process, and so forth.

Operator Manager. The Operator Manager implements the operators that can be used for defining relational recommendation workflows. Apart from exposing the classical relational operators, i.e., selections, projections and joins, it implements additional operators: the extend operator, as described in the previous section, and the recommend and top-k operators described in Section 2. We also plan to add a blend operator that will unify recommendations gen-

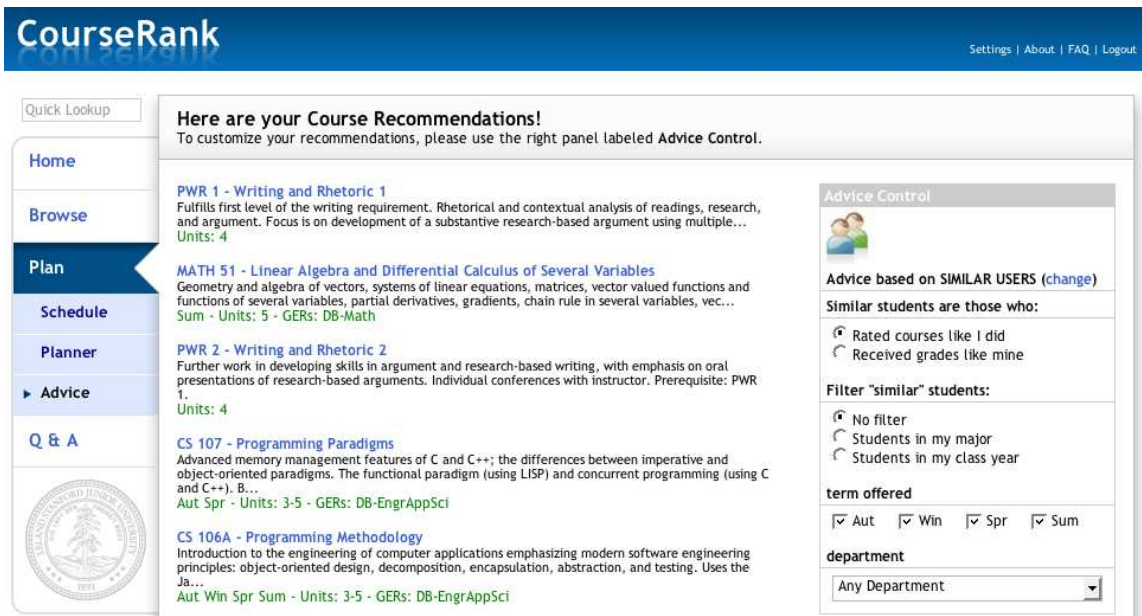


Figure 8: Snapshot of advice page’s collaborative recommendations. (Default)

erated from different workflows or different instantiations of the same workflow.

The extend operator allows “extending” in a transparent way any tuple from one relation with its set of joining tuples from a different relation. For example, students can be extended with their courses. In this way, the set of courses for each student can be “viewed” as an additional attribute of the student that can be handled by the recommend operator irrespective of the database structure. Note that the extend operator does not materialize a new relation. The joins implied by the extended relation are only executed only when tuples are actually requested by some downstream operator.

The recommend operator comes with a library of functions for performing correlations (e.g., Pearson), similarity comparisons (e.g., Jaccard), and aggregations, such as average and weighted average. The operator is responsible for providing the right inputs when calling these functions. The library is extensible, so new types of recommendations can always be supported.

Flow Execution. The Flow Execution component is responsible for the execution of a recommendation workflow and it contains the code for implementing the recommend, the extend and the top-k operators. This component constructs the SQL queries that are required in order to: (i) bring into memory tuples that need to be processed, (ii) realize the extension of tuples required by the extend operator, and (iii) materialize any results that are shared in a particular recommendation workflow. The SQL queries are executed by the underlying DB engine. The Flow Execution component assembles results in memory and performs any extra processing required, such as applying any library function used by a recommend operator.

In the current *CourseRank* version, a workflow is executed exactly as defined. We are currently working on workflow optimization, where operators can be reordered and executed using different run-time strategies (e.g., exploiting an index). Such optimization is analogous to what a query engine does in a traditional database system, except that now

we have to handle new operators and extended relations.

5. A FLEXIBLE USER INTERFACE

There are several challenges in creating such a flexible recommendation interface. Flexible recommendations are a new concept, so a user interface that provides this customizability could lead to confusion. The interface should make clear that a single workflow is active at a time, and given a particular workflow, the user should be able to set the relevant parameters and understand how recommendations can be tailored. Furthermore, supporting flexible recommendations may be performed through a separate ‘advice’ interface, but we anticipate that the integration of advice with search will reveal interesting challenges. For example, we may want to order search results in a manner consistent with the user’s preferences.

Taking into consideration the above, we have designed an *advice panel* that allows a single workflow to be active at a time, and given a particular workflow, the user can set the relevant parameters. We currently offer two recommendation workflows: a collaborative and a content-based workflow described below.

Overview of Current Prototype. To receive recommendations the user clicks a top-level (tab) link *Advice*. After clicking the link, the user is shown recommendations created with the collaborative recommendation workflow using some default parameter values. Along with the recommendations, the system presents the advice panel as shown in Figure 8. The user experience thus far is consistent with the typical user experience in most recommendation systems. The user can use the advice panel to customize the recommendations. In the advice panel, there is an icon of two people and explanatory text reading:

Advice based on SIMILAR USERS

This indicates that the collaborative workflow is currently active. To switch between recommendation workflows, the user can simply click the *change* link. Now, the icon will

Figure 9: Panel for content-based recommendations.

display a stack of books (Figure 9). The text under this icon will also change to read:

Advice based on COURSE HISTORY

In both workflows, the user can control recommendations through the advice panel, which has two sets of parameters: a set for “pure” recommendation operations and a set for filtering returned recommendations. The first set allows currently the user to specify for the collaborative workflow:

- how similarity between two students should be calculated, i.e., using the users’ course ratings or based on grades, and
- how to filter the set of similar students. For example, the user may be interested in recommendations based on students in her major.

In the content-based workflow, the advice panel allows the user to specify:

- how similarity between courses should be calculated: using both the course titles and course descriptions or only course titles, and
- the part of the user’s course history that should be considered. For example, the user may want recommendations within her major and hence, specify that only the courses taken within the major should be examined.

The second set of parameters in the advice panel allows the user to filter the recommendations (generated either by the content-based or the collaborative filtering workflow) by term offered and by department.

Evolving Interface. Designing the interface is an ongoing operation. There are many parameters that we would like the user to be able to control in order to customize recommendations. However, we did not want to confuse the user or overwhelm him with complexity. Thus, we have only chosen a subset of the possible parameters to include in our advice panel.

As users work with the system, we will track the usage of the advice panel to see which parameters are most popular and which could be replaced by others we have been considering. To improve recommendations, we need some way to measure the quality of our suggestions. A user can implicitly indicate that a recommended course is good by adding it to her *CourseRank* schedule or even by simply clicking the recommending item. We save for future work exploring different evidence of recommendation quality.

6. RELATED WORK

Since the appearance of the first papers on collaborative filtering [12, 20, 22], a lot of work has been done from improving and evaluating recommendation methods [3, 14, 19, 23] to designing trustworthy recommender systems [7, 17].

Recommendation approaches are broadly classified into the following categories [6]:

- *Content-based recommendations:* The user is recommended items similar to the ones the user preferred in the past;
- *Collaborative recommendations:* The user is recommended items that people with similar tastes and preferences liked in the past;
- *Hybrid approaches:* These methods combine collaborative and content-based methods.

Traditionally, recommendation systems deal with applications that have two types of entities, users and items (e.g., movies, Web pages) and they recommend top N items to a user following a content-based or a collaborative filtering paradigm. For example, the content-based component of the Fab system [6], which recommends Web pages to users, represents Web page content with the 100 most important words. Similarly, the Syskill & Webert system represents documents with the 128 most informative words [19]. This is a very restrictive view of the world. Many applications use much richer data with large amounts residing in databases. Different types of entities may co-exist in a single database, such as authors, books, customers, publishers, represented by rich sets of attributes. Therefore, being able to ask recommendations that can dynamically incorporate different parts of a database and be targeted to different entities is very useful. However, existing methods do not allow such flexibility and they provide canned recommendations.

Furthermore, using a fixed approach to recommendations can also be restrictive, since a single approach may not be appropriate for every occasion. For example, in content-based methods, the user is limited to see items that are similar to those already rated. Methods to address this problem include the use of genetic algorithms [23] or filtering out items if they are too similar to those seen before [9]. On the other hand, collaborative methods, grouped in memory-based [10, 13, 20] and model-based ones [8, 15], provide serendipitous recommendations. However, collaborative filtering has its own problems, such as the inability to recommend new items [6]. Several recommendation systems use a hybrid approach by combining collaborative and content-based methods, which helps to avoid certain limitations of content-based and collaborative systems [6, 8, 21]. Still, these methods may not provide a complete solution, since in many cases, different recommendations may be required under different circumstances.

The inherent limitations of recommendation systems have been acknowledged [5] and some extensions have been recently proposed, such as incorporating multi-criteria ratings into recommendation methods. In order to give to users the ability to shape their recommendations, the language RQL has been the first proposal that allows the end-users to formulate recommendations of interest in a flexible manner [4]. However, RQL queries follow a data warehouse approach and they are formulated on a pre-specified multidimensional cube of ratings. Our flexible recommendation workflows are more expressive as they can be defined over relational data using combinations of various traditional re-

lational operators as well as new operators and do not assume any pre-specified setting or data structure, such as a cube of ratings. Entities, attributes for comparing entities, attributes to be used for predicting recommendations, even the functions used in the computations may vary on demand. It is possible to dynamically define different recommendation strategies that move away from the typical content-based and collaborative filtering processes of computing recommendations and then allow the users to shape them through a set of knobs.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have seen through our course planning tool, *CourseRank*, how existing recommendation methods lead to fixed, pre-specified recommendations that cannot adapt to each particular student's changing requirements and do not help exploit the full extent of the available options. These observations have shown the need to support flexible recommendations. We have defined the concept of a flexible recommendation workflow, i.e., a high-level description of a parameterized process for computing recommendations. The input parameters of a workflow allow one to generate flexible recommendations. We have seen how flexible recommendations can be expressed over a relational database and we have presented our prototype system that allows defining and executing recommendation workflows over relational data. We built a user interface in *CourseRank* that allows students to make use of two workflows defined and executed with the help of this system. We plan to expose students to other kinds of recommendation workflows as well, such as the friends of friends workflow briefly sketched in this paper.

Building a flexible recommendation engine over relational data presents many challenges. We are currently working on adding a blend operator and on workflow optimization, where operators can be reordered and executed using different run-time strategies. We are interested in making an extensible platform that will allow expressing and experimenting with different types of flexible recommendations. Furthermore, building user interfaces for flexible recommendations is an ongoing operation. Finally, expressing flexible recommendations in a declarative language would be another interesting research direction.

Acknowledgements We would like to thank the other members of the *CourseRank* team, Filip Kaliszyn and Henry Liou, who have worked hard on other parts of the system.

8. REFERENCES

- [1] CourseRank: url:<http://courserank.stanford.edu>.
- [2] The Stanford Daily: url:
<http://stanforddaily.com/article/2007/12/5/-editorialcourserankalongoverduesuccess>.
- [3] G. Adomavicius and Y. Kwon. New recommendation techniques for multi-criteria rating systems. *IEEE Intelligent Systems*, 22(3), 2007.
- [4] G. Adomavicius and A. Tuzhilin. Multidimensional recommender systems: A data warehousing approach. In *WELCOM*, 2001.
- [5] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.
- [6] M. Balabanovic and Y. Shoham. Fab: Content-based, collaborative recommendation. *C. of ACM*, 40(3):66–72, 1997.
- [7] R. B. Bamshad Mobasher, Robin Burke and C. Williams. Towards trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Transactions on Internet Technology*, 7(2), 2007.
- [8] D. Billsus and M. Pazzani. Learning collaborative information filters. In *ICML*, 1998.
- [9] D. Billsus and M. Pazzani. User modeling for adaptive news access. *User Modeling and User-Adapted Interaction*, 10(2-3):147–180, 2000.
- [10] J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *14th Conf. Uncertainty in Artificial Intelligence*, 1998.
- [11] A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, pages 271–280, 2007.
- [12] D. Goldberg, D. Nichols, B. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *C. of ACM*, 35(12):61–70, 1992.
- [13] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *ACM SIGIR Conf.*, 1999.
- [14] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *TOIS*, 22:5–53, 2004.
- [15] T. Hofmann. Collaborative filtering via gaussian probabilistic latent semantic analysis. In *ACM SIGIR Conf.*, 2003.
- [16] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, Jan/Feb 2003.
- [17] M. O. Mahony, N. Hurley, N. Kushmerick, and G. Silvestre. Collaborative recommendation: A robustness analysis. *ACM Transactions on Internet Technology*, 4(4):344–377, 2004.
- [18] B. Miller, I. Albert, S. Lam, J. Konstan, and J. Riedl. MovieLens unplugged: Experiences with an occasionally connected recommender system. In *Int’l Conf. Intelligent User Interfaces*, 2003.
- [19] M. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27:313–331, 1997.
- [20] P. Resnick, N. Iakovou, M. Sushak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Conf. on Computer Supported Cooperative Work*, 1994.
- [21] A. Schein, A. Popescul, L. Ungar, and D. Pennock. Methods and metrics for cold-start recommendations. In *ACM SIGIR Conf.*, 2002.
- [22] U. Shardanand and P. Maes. Social information filtering: Algorithms for automating word of mouth. In *Conf. Human Factors in Computing Systems*, 1995.
- [23] B. Sheth and P. Maes. Evolving agents for personalized information filtering. In *IEEE Conf. Artificial Intelligence for Applications*, 1993.