Social Sites Research Through CourseRank

Benjamin Bercovitz, Filip Kaliszan, Georgia Koutrika,
Henry Liou, Aditya Parameswaran, Petros Venetis,
Zahra Mohammadi Zadeh, Hector Garcia-Molina
Computer Science Department, Stanford University, California, USA
{berco, kaliszan, koutrika, liouh, adityagp, venetis, zahram, hector}@stanford.edu

ABSTRACT

Social sites such as FaceBook, Orkut, Flickr, MySpace and many others have become immensely popular. At these sites, users share their resources (e.g., photos, profiles, blogs) and learn from each other. On the other hand, higher education applications help students and administrators track and manage academic information such as grades, course evaluations and enrollments. Despite the importance of both these areas, there is relatively little research on the mechanisms that make them effective. Apart from being both a successful social site and an academic planning site, CourseRank provides a live testbed for studying fundamental questions related to social networking, academic planning, and the fusion of these areas. In this paper, we provide a system overview and our main research efforts through CourseRank.

1. INTRODUCTION

A growing number of social sites can be found on the Web enabling people to share different kinds of resources, such as: photos (e.g., Flickr [4]), URLs (e.g., Del.icio.us [3]), blogs (e.g., Technorati [17]), and so forth. These sites differ from the open Web in that they tend to foster communities of registered users that contribute regularly and are controlled by some entity that can set up "rules" of engagement.

The increasing popularity of these systems has motivated a number of studies (e.g., [2, 5, 6]) that have mainly focused on understanding the usage and evolution of these systems as well as a number of efforts on harvesting social knowledge for tasks, such as resource recommendations [13, 14, 18], expert and community identification [10, 19] and ontology induction [16]. Still, there are many unanswered questions about how people interact and what services should and can be offered in social sites. During the summer of 2007, we decided that if we wanted to investigate social sites, we needed to have our own site. The result of our effort is CourseRank, a social site where Stanford students can review courses and plan their academic program.

By focusing on an academic site, not only can we study social sharing and networking, but we can also study an important application area for database systems that has seen little research: *higher-education applications*. There are over 6000 Universities in the USA alone, with over 15M college students, most of whom use software to track courses that they take. Several companies, including Red Lantern (DARS), Jenzabar, Datatel, and PeopleSoft (Oracle), have products for course planning (but not centered on a student community). Thus, our work "kills two birds with one stone", investigating not just social sites, but an interesting and important application beyond "sharing videos and chatting about movies" as in many current social sites.

In the next section we describe the current CourseRank system and what sets it apart from other social sites. Then, in Section 3, we give an overview of our research work in CourseRank. In Section 4, we discuss evaluating social aspects of the system.

2. COURSERANK SYSTEM DESCRIPTION

Using CourseRank, students can search for courses of interest, evaluate courses taken and receive personalized recommendations based on what other students have taken and liked. Faculty can also add comments to their own courses, and can see how their class compares to other classes. CourseRank has been successful (it is already used by more than 10,000 students at Stanford, out of a total of about 14,000 students) because in addition to features common to other social sites, it provides special tools geared to the academic domain. For instance, Figure 1 shows two CourseRank screen shots: on the left is part of a course description page, while on the right is the 4-year course planner that helps students structure their courses over multiple years.

In addition, unlike other social sites, it provides access to three types of information:

- *Official Stanford Data.* We have access to official information, including course descriptions and schedules, and results of the official course evaluations conducted by the university.
- *Personal Information.* Students provide personal information (e.g., their class, major), the courses they have already taken and their grades.
- *Evaluations.* Students can evaluate courses they have taken and enter comments. This component is similar to what commercial sites offer, e.g., for rating and reviewing books at Amazon.com.

3. DATA-CENTERED SERVICES

In CourseRank, unlike other public course evaluation sites (e.g., RateMyProfessors.com) and social sites, we have access to much richer data: In addition to basic information (what courses a student took and which ones he or she liked), we have other types of information: what grade the student received, how the courses interrelate (needed for major, pre-requisites), and user profiles (major, class, ...). This rich data gives us the opportunity to develop novel types of data-centered services, where the user interacts mainly with the data that the system has collected from all users. These services include: a recommendation engine that lets students personalize their recommendations; a requirements service that checks if students have met program requirements, and incorporates missing courses into recommendations; a novel interface that facilitates discovery of new courses. For each service we explore models for representing information (e.g., course requirements), algorithms and options for implementing them, desirable user interfaces, expected performance, and social implications.

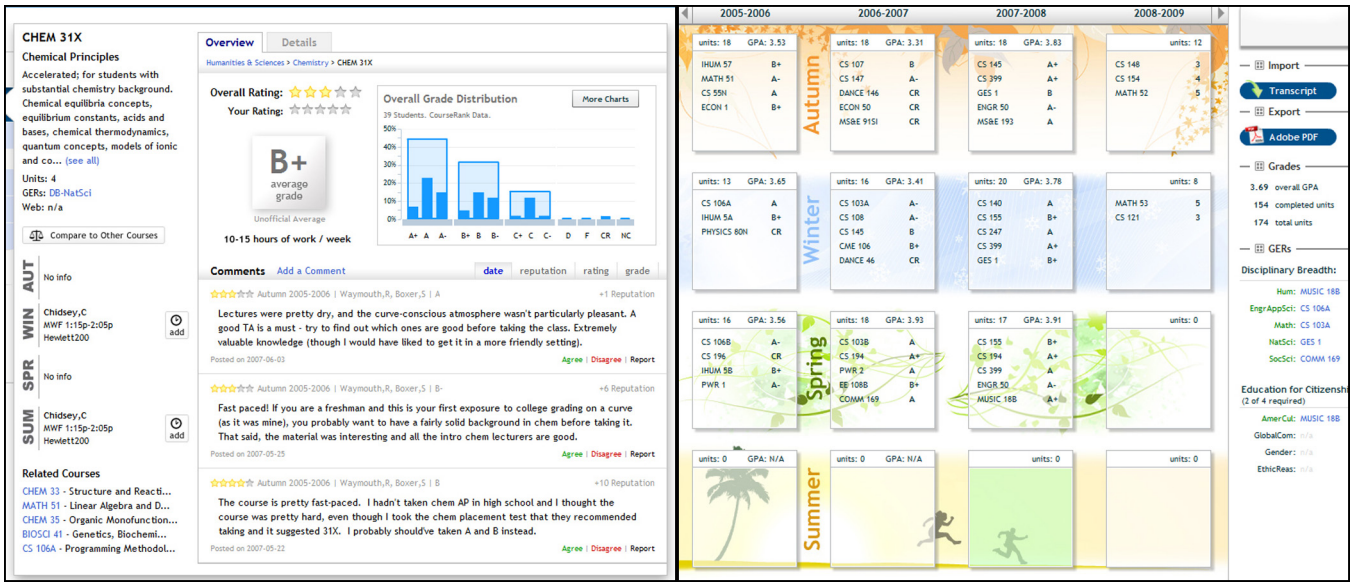

Figure 1: CourseRank Screen Shots: course description (left), course planner (right).

In this section, we drill-down in more detail into these services, to illustrate the types of challenges we address and the types of solutions that emerge from our work.

3.1 Flexible Recommendations

Social networking and commerce sites often provide recommendation services. For instance, MovieLens [12] recommends movies to watch, while Amazon [11] recommends books to buy. Recommendations are also important in CourseRank, and there are many more recommendation challenges than in a traditional site. In CourseRank, there are multiple dimensions to recommend (courses, quarters, majors, instructors), and there are multiple ways to recommend them. For example, a course recommendation can be based on what "similar" students have taken, where similarity is based on liking the same courses, or getting the same grades, or being in the same major. The recommendations can also be based on what courses are needed for graduation, or on what is available when the student has free time in the week. The same course can be offered at different times, with different instructors, teaching assistants, and textbooks. In addition, courses (unlike books or movies) often need to be taken in a certain order or must satisfy certain constraints.

For this purpose, we have developed a flexible recommendation service that allows recommendations to be easily defined, customized, and processed. The engine gives not one canned recommendation (as most current systems do [1]), but the flexibility to specify what is desired. Our goal is to allow a student to specify their goal (courses, quarters, instructors, ...), the basis of recommendations (grade similarity, evaluation similarity, ...) and filtering conditions (e.g., I am looking of a biology class that satisfies my science requirement).

A given recommendation approach can be expressed declaratively as a high-level workflow over structured data and then executed by the underlying engine. Our view is that a site administrator declaratively expresses a suite of workflows; then students can select a workflow, and provide parameters to it. For example, parameters may specify courses the student is interested in or peer students with whom to get compared.

We now describe our flexible recommendation model, FlexRecs (presented at the SIGMOD Conference [7].) Since CourseRank data is currently relational, we start with a relational algebra representation of workflows, containing *traditional relational operators* such as select, project and join, *plus new recommendation operators* that generate or combine recommendations. At the heart of these new operators is a special recommend operator that takes as input a set of tuples and ranks them by comparing them to another set of tuples. The operator may call upon functions in a library that implement common tasks for recommendations, such as computing the Jaccard or Pearson similarity of two sets of objects. The operator may be combined with other recommendation and traditional relational operators.

To illustrate FlexRecs, suppose that our information on courses, students and evaluations is stored in the following three relations. (Even though we continue to focus on academic planning, our model is generic and can be used in any recommendation scenario.)

Courses(CourseID, DepID, Title, Description, Units, Url)
Students(SuID, Name, Class, GPA)
Comments(SuID, CourseID, Year, Term, Text, Rating, Date)

In order to build recommendations, we have a library of comparison functions (e.g., to compare course ratings, course topics, student names, etc), such as Pearson's, and Jaccard index, and for aggregations, such as average and weighted average.

Assume that we want to compute course recommendations for a student with id 444 based on the ratings of similar students. In this case we assume that two students are similar if their course ratings are similar. For our example, we will compute similarity between two students by taking the inverse Euclidean distance of their course ratings. We compute the final course ratings by taking the weighted average of course ratings by students who are similar to student 444. The desired recommendations are captured by the workflow shown in Figure 2 (left). As we can observe, it is composed of traditional select, project, join operators, and it also contains some new operators that we describe below.

Ideally, we would like to represent our application entities with a single relation. For instance, a tuple in such a relation could contain base information on a student (e.g., name), plus the courses a student has taken. For this purpose, we use an *extend operator* (ε) that generates a virtual 2-level nested relation. This operator allows

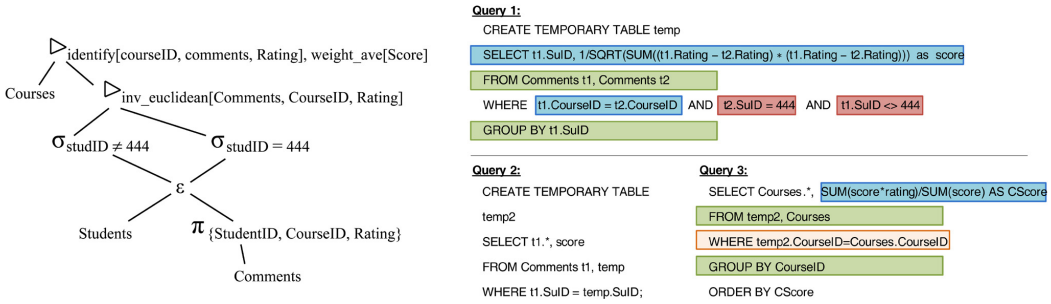

Figure 2: Sample workflow (left) and generated SQL plan (right).

"extending" each tuple from one relation with the set of joining tuples from a different relation. In our example workflow, students are extended with their course ratings, so that the set of ratings for each student can be "viewed" as another attribute of the student by subsequent operators in the workflow irrespective of the database schema. Hence, it would be just as easy to compare students based on their name (a normal attribute) or based on their ratings (an extended attribute). In a sense, *extended relations* can be thought of as "views" that group together information related to an individual entity and represent it as a single tuple.

We observe that recommendations are based on comparisons (e.g., courses are rated against student ratings, students are compared to a student based on their ratings in order to find similar students, and so forth). For this purpose, we use the *recommend operator* ($\triangleright_{cf}$), which rates the tuples of a set by comparing them to the tuples of another set using a comparison function cf. Our example workflow has two recommend operators. The lower one finds similar students to the student with id 444 using the inverse Euclidean distance of their ratings. The upper one finds courses recommended by these students taking a weighted average of their ratings.

We have built a FlexRecs engine in Java that compiles and executes workflows on top of the CourseRank MySQL database. To illustrate, Figure 2 (right) shows the compiled plan (Queries Q1, Q2, Q3) for our sample workflow. We will first see the set of queries for implementing the lower recommend operator in Figure 2 (left). Query Q1 has several shaded parts: (a) the select operators have been included as conditions in the WHERE clause, (b) the recommend operator, which compares students using the inverse Euclidean comparison function on their course ratings, is implemented by combining the aggregation functions that are supported by the underlying database, (c) the extend operator is implemented by a GROUP BY clause. This query creates a temporary in-memory table that contains two attributes for each student: the student id and a score. Q2 combines for each student the score and the student ratings into one relation. Then, the higher recommend operator computes course recommendations based on the ratings provided by the similar users based on their scores. The computations required are again realized by leveraging the database's existing aggregation capabilities as shown in Q3.

Handling the full suite of FlexRecs operators is more challenging than what this simple example illustrates. In summary, FlexRecs is a way to express recommendation strategies more compactly and clearly than using, say SQL or Java. FlexRecs makes it possible to offer users (in any social site) a variety of recommendation strategies, that can be easily tailored to their interests. In CourseRank we have been able to quickly modify existing workflows to experiment with a variety of recommendation strategies. There is, of course, still much remaining work, for instance:

- *Optimization.* Implementing the new operators inside the database engine will enable the implementation of special workflow optimization schemes. For example, we may be able to push down selections and change the order of recommendation operators or dynamically define what comparisons should be performed in order to achieve a good trade-off between recommendation efficiency and effectiveness.
- *User Interface.* Developing appropriate user interfaces that will allow users to specify the kind of recommendations they want is also very challenging. We plan to develop and evaluate different interfaces for students to select workflows and provide parameters. Such an interface will allow users to specify their target (e.g., courses, instructors, majors, quarters), filtering conditions (e.g., biology courses, engineering majors), and the basis for the recommendation (students with similar grades, with similar tastes, and so forth).
- *Other Applications and Non-Relational Data.* We will explore how FlexRecs can be extended to support non-relational data (e.g., XML, Jason) and other recommendation applications.

3.2 Course Requirements

In order to graduate, students must satisfy a set of requirements. For example, a Computer Science (CS) major at Stanford must satisfy a set of sub-requirements, one of which is the math requirement (simplified):

- The student must complete Math 41 and 43, or as an alternative Math 19, 20, 21.
- The student must complete either CS 103X or the pair CS 103A, 103B.
- The student must complete two electives out of the set Math 51, 103, 108, ... CS 156, 157, ... Completion of Math 52 and Math 53 will together count as one Math elective. Restrictions: Math 51 and 103, or Math 51 and CME 100, or ... may not be used in combination to satisfy the Math electives requirement.
- The total units for Math courses should be 23 or greater.

A system like CourseRank needs to understand such relationships in order to (*a*) help students manage their courses (e.g., am I done with the foreign-language requirement?), and (*b*) improve recommendations (e.g., course x is highly recommended for you because it helps you complete your major requirements faster).

Achieving this functionality we need: (*a*) a language for describing the requirements commonly seen at universities; (*b*) algorithms for efficiently checking if requirements have been satisfied (and for explaining what parts have not been satisfied); and (*c*) ways to translate our knowledge of what courses help a student complete requirements into recommendations for the student.

University requirements are quite diverse, so they represent the ideal testing ground to understand recommendations in the face of

complex constraints. Simply capturing the requirements in a succinct and usable form is one of the challenges. (Several commercial products provide ways of capturing academic requirements, but their models are so complex that they are not widely used.) Furthermore, it turns out that efficiently checking requirement satisfaction is not trivial. One complexity is that a course a may appear in multiple sub-requirements, and it can only be used to satisfy one of the sub-requirements (see example below). Furthermore, there are often exceptions to the rules, pre-requisites for courses, and so on.

In general, we have shown that checking such complex constraints is NP-hard [15]. However, we have identified a class of requirements that in practice is at the core of most actual requirements and that can be checked efficiently. We next illustrate this class and one efficient checking algorithm that can form the basis of a more general scheme.

In the class that we study, requirements are expressed as conjunctions of sub-requirements, where each sub-requirement R_i is of the form `take` k_i `from` S_i. Here, S_i is a set of courses, and $k_i > 0$. Note that $S_i \cap S_j$ need not be empty, i.e., there could be courses that are common to the two sub-requirements as well, e.g., the database systems principles course could be both in the theory and systems sub-requirements. However, a taken course can be counted towards only one sub-requirement.

To illustrate requirements and our algorithm, say students must satisfy these three sub-requirements:

- R_1: `take 1 from` $\{a, p\}$
- R_2: `take 2 from` $\{p, d, i\}$
- R_3: `take 1 from` $\{i, o\}$.

A student who has taken courses $\{a, p, i, o\}$ has satisfied the requirement. Another student, say Bob, who has only taken $\{p, d, o\}$ has not satisfied the requirement.

Our checking (and recommendations) algorithm is based on building a flow graph, as illustrated in Figure 3(left). The courses are divided into two groups representing the courses taken by Bob (left top oval), and the remaining courses (left bottom oval). Each course is connected to the sub-requirements it helps satisfy. Each link has two numbers associated with it. The first is a maximum capacity. The capacity is 1 for all links except those connecting to the target t, in which case the capacity is the "take k" value associated with the sub-requirement. The second number is a cost (in square brackets), which is used only for links from the source s to a course from the ones that Bob has not taken yet. For now, assume these costs are 1.

It turns out that if we run a min-cost max-flow algorithm on this graph we can not only check if Bob has satisfied the requirements, but we can also obtain the smallest set of additional courses that are needed towards this end. In particular, if there is a feasible flow of magnitude $\sum_j k_j$, then there is an assignment of courses to sub-requirements such that each sub-requirement is satisfied. (The converse is also true.) If no additional courses are used (i.e., have a non-zero flow in the solution), then Bob has satisfied the requirements. If not, then the used courses represent the smallest set of courses needed, i.e., courses we can recommend to Bob. In this example, Bob has only taken 3 courses, so some of the remaining courses are also needed. The algorithm uses course i, achieving a max-flow of 4 to t, at a min-cost of only 1. Thus, Bob is recommended course i. It can be shown [15] that the complexity of checking/recommending courses in this fashion is $O((c*r+m)^2 \sum_j k_j)$, where c is the number of courses, m is the number of sub-requirements, and each course appears in at most r sub-requirements. Thus, this approach is relatively efficient.

We can take this approach one step further by assigning to all courses that have not been taken by a student costs that reflect their "inverse desirability". That is, using traditional recommendation schemes, we can assign to each course c a score $sc(c)$ (between 0 and 1) that represents its utility based on grades of similar students, popularity, ratings, prerequisites being satisfied, etc. Then we can use $1 - sc(c)$ as the cost of a course, and the min-cost max-flow algorithm will give us a set of candidate courses that (*a*) contains the smallest feasible number of courses, and (*b*) among the smallest feasible sets, has the highest aggregate score. Thus, we can now recommend courses that both help meet requirements and are desirable. To complete our example, say given Bob's grades and tastes, course a has a score of 0.9 (very desirable), while i has a score of 0.5 (less desirable). In this case, the recommendation changes from i to a. (When the costs were equal, R_1 was satisfied by p and R_2 by i, d. Now, R_1 is satisfied by a and R_2 by p, d.)

Our network flow solution can be extended to handle additional types of constraints [15]. It can also be used as an initial filtering step when more complex constraints exist. That is, we can generate solutions that satisfy the constraints we can handle efficiently, and then check if the resulting assignments of courses to sub-requirements also satisfy the more complex constraints. If the complex constraints are not met, then we can do more sophisticated (and expensive) searching (which we believe will be rare).

Clearly, there is still substantial work to be done:

- *Recommendation Evaluation*. Using actual requirements from a variety of programs and the courses taken by students, we will determine how efficient each recommendation approach is.
- *Prerequisites*. We will incorporate prerequisites into our framework. For instance, we may not want to recommend course a if course b needs to be taken first, unless we can also incorporate b into our recommendation.
- *Other domains.* We will apply/extend our requirement model and algorithms to other settings. For example, say a banker wants to recommend an investment portfolio to a customer, with constraints on the type of investments and amounts. What constraints appear here (perhaps similar to course constraints)? How can we make recommendations with such constraints?

3.3 Course Cloud

In order to facilitate course planning, CourseRank offers two traditional interfaces: one for browsing courses based on department and a keyword-based search interface. Keywords are searched in the title and the description of courses.

When browsing courses based on department, students have to sift through long lists of courses and read their descriptions in order to discover courses of interest. Many courses may cover common topics and different departments may offer courses on similar topics making locating and sorting out the available options very tedious. On the other hand, keyword searching offers more flexibility but users still need to figure out the right search keywords, not always an easy task. Furthermore, users often want to search beyond the immediate course description. For example, if a student searches for "Java", he may be interested not only in courses that explicitly mention this word in their title or description, but also in courses with implicit references to "Java", such as in their comments.

CourseCloud (presented at EDBT 2009 [9]) is an improved search service, especially targeted at the *discovery of unexpected but useful* courses or other resources (as opposed to searching for a specific course with known characteristics). *CourseCloud* uses three main ideas: (*a*) in addition to search results, the service presents a "tag cloud" where users can see unexpected terms that may be of interest. In this case, the "tags" are not traditional tags added by users,

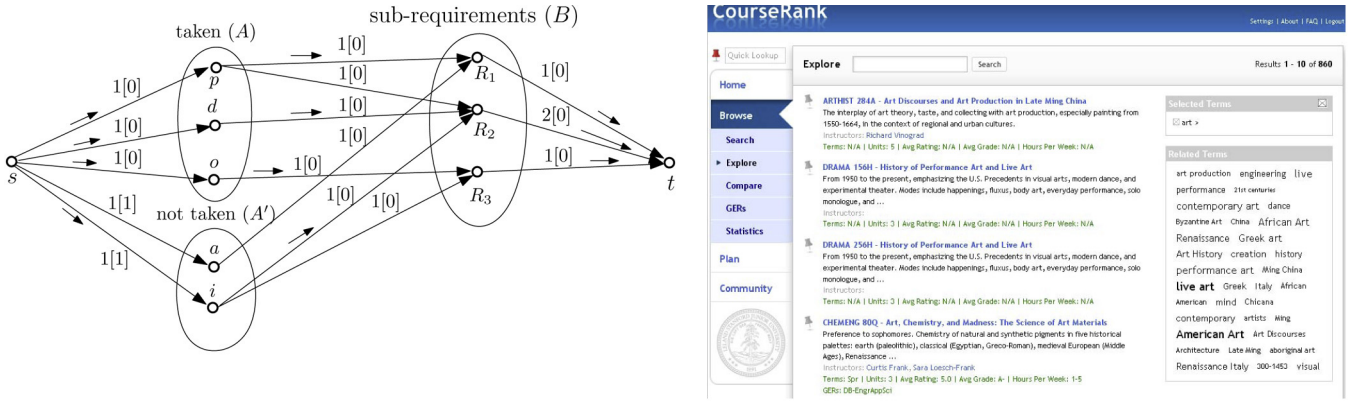

Figure 3: Example flow graph (left); Example Course Cloud (right).

but terms from the database that are explicitly or implicitly connected to the search results. (*b*) CourseCloud searches for terms (keywords entered by user or tags) in data related to courses (e.g., student comments), not just in the course records. (*c*) CourseCloud uses the tags for navigation and search refinement. Our initial experience with this prototype shows that for some students it provides a very useful service.

Figure 3(right) illustrates the CourseCloud interface, after the student has typed in the keyword "art". The left display shows courses related to the search, that have "art" in their description or in "nearby" records (e.g., in comments). On the left is the tag cloud, providing many diverse concepts related to "art" that are found in the matching courses, such as "performance", "art production", and "Renaissance". For example, the term "performance" is found in many user comments that refer to "art" courses with live performances. The data cloud conveniently categorizes courses in a digestible way under different concepts. Thus the student can find out that there are courses offered not only by the ART HISTORY program (identified by the course code in the results) but also from other programs that address other aspects, such as the DRAMA or HUMANITIES programs.

When the student clicks on a tag, say "architecture", the system adds the term to the search and displays new results and a new cloud, allowing the student to drill-down. Some of the new tags, like "Byzantine art" or "religious art" may be unexpected to the student, allowing the discovery of courses the student might not have thought of.

Even though tag clouds are popular on some web sites, there has been little research on them, and to our knowledge, no work on using them to explore non-tagged content. There are many important questions to investigate related to this type of service:

- *Tag selection.* In [9] we explore some initial techniques that show promise in identifying the terms users find most useful. There are many other options that need to be evaluated in terms of coverage, diversity, overlap, and other aspects.
- *Personalization.* Displaying the most popular tags is not hard, but displaying tags that are "personalized" to a user is much more challenging. We will study the performance of various tag selection schemes that adapt to the needs and preferences of a particular user.

4. SOCIAL IMPACT

In a social networking site, users interact not just with the computer, but with each other. CourseRank provides a living laboratory for studying or revisiting human interactions in the specific context of an electronic academic community.

We briefly summarize two results we have obtained using CourseRank. Details on how these two results were obtained, plus additional results can be found in [8], published in ICWSM 2009.

Are users of social sites truthful? Users at social sites provide a lot of information about themselves, e.g., their gender, age, interests, and so on. How reliable is all this information? There is seldom a way to verify the authenticity of user provided data, and anecdotally we know the some users lie about their age or gender. However, CourseRank gives an opportunity to verify some of the information users provide. For example, Figure 4(left) shows two grade distributions for students in the Engineering School at Stanford. The top one is the official distribution, provided to us by the registrar. It shows how many students received a particular grade in a period. The bottom distribution uses self-reported grades, which the students enter into CourseRank as they plan their academic program. We have fewer self-reported grades, but overall the distributions follow very similar patterns. This result suggests that on aggregate users are giving us very accurate grade information. Of course, users have an incentive to give good data, since the data helps them plan their program. Additional experiments are needed to understand if there is a grade bias in some circumstances (e.g., in popular or large courses) or for other type of information.

Are raters objective? Many universities use course evaluations to evaluate and promote professors, and university administrators frequently argue that such evaluations are unbiased. That is, a professor cannot improve his ratings by giving out higher grades. However, Figure 4(right) shows a clear correlation between the grade a student receives and the rating he gives to the course. For each possible grade (horizontal axis), the vertical axis shows the average course rating given by students that earned that grade. (User ratings range from the lowest rating of 1 to the highest of 5.) In this case the grades are self reported, as discussed above. Of course, this result does not prove that instructors can improve their ratings by giving out higher grades, it only shows a correlation. But we plan to shed more light on this issue by examining the written comments left with the course evaluations, and by comparing the ratings and grades of a student in different courses.

There are several interesting questions to study, such as:

- *Community effect*. What role does community size and corpus size play in user behavior? In CourseRank, we have communities with varying characteristics (e.g., departments with few or many students, departments that offer many or few courses). Do students in different communities behave differently?

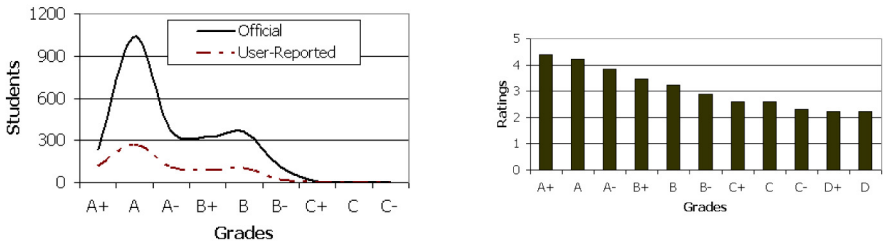

Figure 4: Sample results: Truthfulness (left); Objectiveness (right).

- *Incentives*. How can we incentivize students to evaluate more courses, or to give comments that are more constructive? In the early stages of CourseRank, we raffled an iPod to get users to try the site. Can similar techniques be used? What is their impact?

5. CONCLUSIONS AND FUTURE WORK

We will continue to use CourseRank as a live testbed for fundamental research into social systems, developing and evaluating the algorithms and services that drive such systems. There are many interesting questions that can be addressed, regarding how students use the site, the veracity of user provided data, the usefulness of our course recommendations, and so on. At the same time, CourseRank has been spun out of our lab as a company, and the system is being deployed at other universities.

One of the strengths of our project is that we have a concrete application (student academic planning) with many actual users as well as rich and interesting data. In spite of our focus on this application, we believe that much of our work will be applicable to other applications, and indeed, a concurrent goal of our project is to explore other domains and applications. For instance, several companies have expressed interest in using CourseRank as a starting point for a "corporate social site" where employees discover resources and plan projects. Many of the services we explore (e.g., recommendations) would be applicable in such a system.

6. REFERENCES

[1] 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.

[2] C. Brooks and N. Montanez. Improved annotation of the blogosphere via autotagging and hierarchical clustering. In *Proceedings of the 15th International Conference on World Wide Web*, 2006.

[3] Del.icio.us: url: http://del.icio.us/.

[4] Flickr: url: http://www.flickr.com/.

[5] S. Golder and B. A. Huberman. Usage patterns of collaborative tagging systems. *Journal of Information Science*, 32(2):198–208, 2006.

[6] V. Gomez, A. Kaltenbrunner, and V. Lopez. Statistical analysis of the social network and discussion threads in slashdot. In *Proceedings of the 17th International Conference on World Wide Web*, 2008.

[7] Georgia Koutrika, Benjamin Bercovitz, and Hector Garcia-Molina. Flexrecs: Expressing and combining flexible recommendations. In *SIGMOD Conference*, 2009.

[8] Georgia Koutrika, Benjamin Bercovitz, Filip Kaliszan, Henry Liou, and Hector Garcia-Molina. Courserank: A closed-community social system through the magnifying glass. In *Third International Conference on Weblogs and Social Media (ICWSM)*, 2009.

[9] Georgia Koutrika, Z. Mohammadi Zadeh, and Hector Garcia-Molina. Data clouds: Summarizing keyword search results over structured data. In *12th International Conference on Extending Database Technology (EDBT)*, 2009.

[10] X. Li, L. Guo, and Y. Zhao. Tag-based social interest discovery. In *Proceedings of the 17th International Conference on World Wide Web*, 2008.

[11] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, Jan/Feb 2003.

[12] B.N. Miller, I. Albert, S.K. Lam, J.A. Konstan, and J. Riedl. Movielens unplugged: Experiences with an occasionally connected recommender system. In *IntŠl Conf. Intelligent User Interfaces*, 2003.

[13] G. Mishne. Autotag: collaborative approach to automated tag assignment for weblog posts. In *Proceedings of the 15th International Conference on World Wide Web*, 2006.

[14] T. Ohkura, Y. Kiyota, and H. Nakagawa. Browsing system for weblog articles based on automated folksonomy. In *Proceedings of the WWW 2006 Workshop on the Weblogging Ecosystem: Aggregation, Analysis and Dynamics*, 2006.

[15] Aditya Parameswaran, Petros Venetis, and Hector Garcia-Molina. Recommendation systems with complex constraints: A courserank perspective. In *Stanford InfoLab Technical Report, available at http://ilpubs.stanford.edu:8090/909/*, 2009.

[16] P. Schmitz. Inducing ontology from flickr tags. In *Collab. Web Tagging Workshop in conj. with WWW2006*.

[17] technorati: url: http://www.technorati.com/.

[18] Z. Xu, Y. Fu, J. Mao, and D. Su. Towards the semantic web: Collaborative tag suggestions. In *Collab. Web Tagging Workshop in conj. with WWW2006*.

[19] J. Zhang, M.S. Ackerman, and L. Adamic. Expertise networks in online communities: Structure and algorithms. In *WWW*, 2007.