Recsplorer: Recommendation Algorithms Based on Precedence Mining

Aditya Parameswaran[1] Georgia Koutrika[2] Benjamin Bercovitz[1] Hector Garcia-Molina[1]

[1]Computer Science Dept., Stanford University
Stanford, CA - 94305, USA
{adityagp, berco, hector}@cs.stanford.edu

[2]IBM Almaden Research Center
San Jose, CA - 95120, USA
gkoutri@us.ibm.com

ABSTRACT

We study recommendations in applications where there are temporal patterns in the way items are consumed or watched. For example, a student who has taken the Advanced Algorithms course is more likely to be interested in Convex Optimization, but a student who has taken Convex Optimization need not be interested in Advanced Algorithms in the future. Similarly, a person who has purchased the Godfather I DVD on Amazon is more likely to purchase Godfather II sometime in the future (though it is not strictly necessary to watch/purchase Godfather I beforehand). We propose a *precedence mining* model that estimates the probability of future consumption based on past behavior. We then propose Recsplorer: a suite of recommendation algorithms that exploit the precedence information. We evaluate our algorithms, as well as traditional recommendation ones, using a real course planning system. We use existing transcripts to evaluate how well the algorithms perform. In addition, we augment our experiments with a user study on the live system where users rate their recommendations.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information Filtering*

General Terms

Algorithms, Experimentation, Measurement

Keywords

recommendations, precedence mining, temporality

1. INTRODUCTION

Recommender systems provide advice on products, movies, web pages, and many other topics, and have become popular in many sites, such as Amazon and Netflix. Many systems use collaborative filtering (CF) methods (e.g., [26]), where users similar to the target user are identified, and then items are recommended based on the preferences of the similar users. Typically, users are considered similar if there is overlap in the items watched or consumed, and similar ratings have been given. The order in which items were consumed is not taken into account.

In this paper, we consider a complementary recommendation strategy, based on temporal (precedence) access patterns. These patterns may encompass user preferences (e.g., students may take a course on Convex Optimization following an Advanced Algorithms course but not the other way around), may encode some logical order of options (e.g., you first take an introductory class in social dance before taking an advanced one), and may capture how interests evolve (e.g., one may watch and like a popular movie by some director and subsequently watch some lesser known films by the same director). With our proposed *precedence mining model*, we estimate the probability that a given user will consume an item in the future, based on past behavior. We then use these probabilities to make recommendations.

Our experimental results show that precedence mining can significantly improve recommendations in an application with temporal patterns. Since precedence mining does not look at user ratings, it does not suffer from the sparsity of ratings problem faced by collaborative filtering (few items are rated by few users). Furthermore, it can exploit patterns across all users, not just similar users.

1.1 Motivation and Outline of Work

Scenario: Our work has been motivated and implemented as part of CourseRank, a commercial site for course planning, originally developed by us at the Infolab at Stanford University. CourseRank helps students make their course plans by providing information about courses, such as bulletin course descriptions, grade distributions, and results of official course evaluations, as well as unofficial information, such as user ratings and comments. Recommendations comprise an important system component that helps students sort out available choices. The current production system uses collaborative filtering based on course ratings.

Outline of Approach: To illustrate our approach and the differences with collaborative filtering, consider the example of Figure 1. We assume (for simplicity) that each quarter a student signs up for just one course, and rates it from 1 to 5. Part (a) of the figure shows five transcripts, where q_1, q_2, ... are the quarters. For example, user u_1 took

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

User	q1	q2	q3	q4
u_1	a: 5	b: 5	d:5	
u_2	a: 1	e: 2	d:4	f: 3
u_3	g: 4	h: 2	e: 3	f: 3
u_4	b: 2	g: 4	h: 4	e: 4
u	a: 5	g: 4	e: 4	

(a) User transcripts

User	q1	q2	q3	q4
u_1	a: 5	b: 5	d:5	
u_2	a: 1	e: 2	d:4	f: 3
u_3	g: 4	h: 2	e: 3	f: 3
u_4	b: 2	g: 4	h: 4	e: 4
u	a: 5	g: 4	e: 4	

(b) Collaborative Filtering

User	q1	q2	q3	q4
u_1	a: 5	b: 5	d:5	
u_2	a: 1	e: 2	d:4	f: 3
u_3	g: 4	h: 2	e: 3	f: 3
u_4	b: 2	g: 4	h: 4	e: 4
u	a: 5	g: 4	e: 4	

(c) Precedence relationships

Figure 1: Motivating example.

course a during quarter q_1 and rated it 5. User (student) u is our target user who needs recommendations.

Let us assume that collaborative filtering matches users that share at least two common courses (and have similar ratings). Hence, users that are similar to u are u_3 and u_4. Inevitably, we ignore potentially useful information contained in the transcripts of u_1 and u_2 because they do not match with u (u_1 has only one course in common with u, and u_2's ratings for the common courses do not agree with u's). Users u_3 and u_4 have course h in common (which is not already taken by u), and we aggregate their ratings (taking for example their average) to predict u's rating for h and make a recommendation accordingly. Figure 1(b) illustrates the part of the transcripts exploited in this case.

With precedence mining we look for patterns where one course follows another. For our small example, say we consider as significant those patterns that are supported by at least two transcripts. (We also had a minimum support of 2 for both the user similarity and rating prediction.) Figure 1(c) illustrates various patterns with shading. For example, it appears that course a is followed by d, and e is followed by f. We can take these relationships into account to make recommendations to u based on the courses already taken. For instance, given that u has taken a, g and e, we can recommend d, f and h, i.e., courses that tend to follow. The different size of the shaded areas in Figures 1(b) and 1(c) illustrates that we exploit a larger portion of user histories, and we are potentially able to make more and better recommendations to more users.

Our approach is related to *sequence mining*, but different in several ways. First, the goal of sequence mining is to extract the temporal patterns themselves, while our goal is to make recommendations. Second, we focus on simple temporal precedence, while sequence mining may look at more complex (and more expensive to identify) multi-item sequences. Third, we assume a course only appears once in a transcript, which impacts the statistics and conditional probabilities we use. There is also related work that tries to predict the *next* item in a sequence. We, on the other hand, find courses that the target user may be interested in taking at some future time, not necessarily immediately.

Prediction problems like ours can be solved using a probabilistic framework that computes the various conditional probabilities given the user's current status or choices (e.g., [6, 25]). Intuitively, this also holds for our problem: given that an item a has been selected by the target user, we want

Transcript #	Containing
1	-
2	a
3	x
4	$a \rightarrow x$
5	$x \rightarrow a$

Table 1: Motivating example on probabilities.

to compute the probability that x will follow, i.e., $Pr[x|a]$. However, there are two complications we need to address.

The first complication is that $Pr[x|a]$ is not exactly what we need. To illustrate, consider five transcripts that may or may not include a or x, as shown in Table 1. For example, Transcript 2 contains a only, while Transcript 4 contains a followed by x. Each course can occur only once in a transcript. The conditional probability $Pr[x|a]$ is equal to $2/3$, which is the number of transcripts containing a and x divided by the number of transcripts containing a. In our context, we are deciding if x is a good recommendation for a target user that has taken a. Thus, we know that our user's transcript does not (and will not) have x before a. Thus, we should ignore transcripts like number 5. That is, the probability we want is $1/2$, the number of transcripts where x follows a divided by the number with a but no x before a. We call this probability $Pr[x|a^{\neg x}]$ (see Section 3.1). The larger this value (compared to other not-taken courses), the stronger will be our recommendation for x.

The second complication is that the target user has taken a list $\mathcal{T} = \{a, b, c, \ldots\}$ of courses, not just a. Thus, what we really need is $Pr[x|\mathcal{T}^{\neg x}]$. Unfortunately, we would need to estimate an exponential number of such probabilities, one for each possible list $\mathcal{T}$. A standard technique in machine learning is to estimate $Pr[x|\mathcal{T}]$ from the estimates $Pr[x|a]$, $Pr[x|b], \ldots$. We use the same idea, but applied to the special conditional probabilities we need.

Evaluation: In the discussion above, we implicitly assumed that we need to optimize *predictability*, i.e., how well the algorithm predicts which courses (books, electronics) the user will take (purchase) later on. In our context, predictability is important because it enables us to present a set of "likely" options to the student who can then easily populate her course plan by selecting the courses that she will take. In some other domains (e.g., Netflix), predictability might be how well the algorithm predicts the rating that a user assigns to a given item.

In addition to predictability, there are other (not necessarily independent) aspects that can be used to judge recommendation "quality" [13]:

• *Coverage:* Coverage is the number of cases for which the algorithm is able to make recommendations. In our setting, when an algorithm is asked for k recommendations, coverage is the number of students who can get k recommendations.

• *Goodness:* Goodness is a measure of how "interesting" a recommendation is to a user. Note that this metric is distinct from predictability. For example, a CS student may say that she is interested in "Contemporary Art", even though it is unlikely that she will ever take it (due to time constraints, course requirements, etc.).

• *Unexpectedness:* Unexpectedness is a measure of how "surprising" the recommendation is to the user. For a student in the CS major, we could recommend Compilers or Biocomputation — both of which may be relevant to the student. However, Biocomputation may be an unexpected course, while Compilers, being a core CS course, is expected.

We evaluate our algorithms on all four aspects in this paper. Note that the last two aspects (and related notions such as serendipity and novelty) are typically not studied [13] but we can do so because we have a live system at our disposal. Although our algorithms are designed to achieve high predictability and coverage (by exploiting temporal patterns), our experiments will show that they do not compromise on goodness and unexpectedness.

1.2 Contributions

In summary, the contributions of this work are the following:

- We propose a *precedence mining* model that estimates the probability of future consumption based on temporal patterns.
- We propose the Recsplorer suite of recommendation algorithms that exploit the mined precedence information to recommend the top-k choices that could follow the past choices of a particular user.
- We propose a new algorithm *Reranked* that gives preference to items with high predictability but low popularity in order to improve the goodness and unexpectedness of recommendations.
- We evaluate our algorithms, as well as traditional recommendation ones, using CourseRank, a real course planning system. We show that our precedence mining-based recommendations outperform even collaborative filtering both in *predictability as well as coverage*.
- We augment our experiments with a user study on the live system where users rate their recommendations on goodness and unexpectedness. We find that our algorithms perform just as well on both goodness and unexpectedness, with the *Reranked* algorithm being the best.

2. RELATED WORK

Recommender systems can be classified into two categories: content-based filtering and collaborative filtering. Content-based filtering analyzes the associations between user profiles and the descriptions of items and recommends items similar to those the user liked in the past [4, 21, 23]. Collaborative filtering methods rely only on past user behavior (e.g., transactions or ratings) to make recommendations and they are of two types: memory-based and model-based.

Memory-based algorithms employ statistical techniques to find a set of users, called neighbors, who tend to agree with the target user (e.g., they rate items similarly), and then combine the preferences of neighbors to produce recommendations for the active user [5, 12, 26]. Model-based approaches use past user behavior to build off-line models (e.g., Bayesian [6], probabilistic relational [11] and probabilistic LSA models [14]) that capture item or user relationships. These models are subsequently used to compute recommendations on-line. Recent model-based methods use pattern discovery techniques. Patterns that are mined can be: *association rules*, i.e., rules for groups of co-occurring objects, or *sequential patterns* capturing strict orders of items.

Association rules are different from our precedence patterns because they contain no ordering relation between the items and they are useful for recommending items related to a particular item [10, 17, 27]. A classical example is the discovery of sets of products usually purchased together by many independent buyers, which is applied to recommend other products to a shopper related to the product currently being viewed (as in the "Customers who bought this item also bought" recommendations in Amazon [1]).

The body of work on sequence mining deals with extracting temporal patterns from data [3, 19, 24]. The patterns obtained are similar to the ones we have described, but they also consider more general patterns, for example, a customer who bought pillows and a pillow-case is likely to buy bed sheets in the next few weeks. Additionally, like our work, sequence mining does not take into account the strict order between the items. However, sequence mining just extracts the "interesting" patterns from a data set (for human consumption), and does not make recommendations using those patterns. Since we make recommendations, we not only need to aggregate these temporal patterns, but also take into account the fact that conditional probabilities in our context are slightly different, i.e., $Pr[x|\mathcal{T}^{\neg x}]$ as against $Pr[x|\mathcal{T}]$, especially since our items are consumed only once. To our knowledge, this is the first work that tries to use sequential patterns while making recommendations.

Previous work studying temporality or order in the context of recommendations has been confined to prediction problems where there is a strict order or path followed by the user, and one wants to predict the next step in that sequence [7, 9, 29]. A typical example is predicting the next Web page a user will access given a sequence of pages visited up to now (known as the *next-page prediction* problem [7]). On the other hand, we model the user's history as a set of items having occurred in the past instead of a strict sequence of items, and we predict the set of items most likely to follow in no particular order and not necessarily comprising the next step a user should make.

The impact of temporal effects on user preferences has recently attracted attention in the recommender community. Existing approaches include modeling temporal changes in user preferences [18] and time-based weighting schemes for similarity-based collaborative filtering [8, 16]. For instance, temporal information can be used to find people who have similar preferences to the target user but purchased items earlier and assign higher weight to similar users who tend to purchase novel items earlier than others [16].

Finally, in contrast to other probabilistic recommendation approaches (e.g., [5, 23]), we do not compute probabilities for ratings and we use finer-grained conditional probabilities. Due to the nature of our problem, state-based models, such as Markov models which have been widely applied to the next-page prediction problem [15, 25], are not appropriate.

Evaluating recommender systems and their algorithms is inherently difficult for several reasons [13]. First, different algorithms may be better or worse on different data sets. For example, many collaborative filtering algorithms have been designed specifically for data sets where there are many more users than items (e.g., the MovieLens data set has 65,000 users and 5,000 movies). Second, the goals for which an evaluation is performed may differ. Early evaluation work focused mostly on the accuracy of recommendation algorithms in predicting withheld ratings. Thirdly, the evaluation of recommender systems is an inherently subjective and ambiguous problem. In the following, we present our interpretation of the existing methods of evaluation of recommender systems, and how our evaluation differs.

Recently, researchers have speculated that there are properties other than accuracy that have a larger effect on user satisfaction and they have looked at measures including the

Item	No. Transcripts	Function
total	50	n
a total	10	$f(a)$
b total	7	$f(b)$
c total	6	$f(c)$
d total	4	$f(d)$
a then c	3	$g(a,c)$
c then a	2	$g(c,a)$
b then c	2	$g(b,c)$
c then b	0	$g(c,b)$
a then d	3	$g(a,d)$
d then a	0	$g(d,a)$
b then d	4	$g(b,d)$
d then b	0	$g(d,b)$
d then c	3	$g(d,c)$
c then d	0	$g(c,d)$
a then b	5	$g(a,b)$
b then a	0	$g(b,a)$

Table 2: Example Counts

degree to which the recommendations cover the entire set of items [22], the degree to which recommendations made are non-obvious [20], and the degree to which recommendations are diverse, i.e., not homogeneous [30]. For analyzing recommender systems with respect to the "non-obviousness" of the recommendation, two dimensions are considered [28]: novelty and serendipity. Novel recommendations help users find "surprisingly interesting" items. A serendipitous recommendation, on the other hand, helps the user find a "surprisingly interesting" item he might not have otherwise discovered. Thus, recommendations that are serendipitous are by definition also novel but the other way around is not always true. Finally, commercial systems may measure user satisfaction by the number of products purchased, while non-commercial systems may just ask users how satisfied they are [13].

In this paper, we evaluate our algorithms from different aspects that are most relevant to our recommendation problem, such as coverage, goodness and unexpectedness. While goodness is a measure of how "interesting" a recommendation is to a user, unexpectedness is a measure of how "surprising" a recommendation is to a user. Thus, a "good and unexpected" recommendation is the same as a "novel" recommendation. By evaluating goodness and unexpectedness separately, we operate at a finer granularity than existing approaches that evaluate novelty. Additionally, both unexpectedness and goodness are easy to measure in user surveys. In our recommendation context, diversity is not a critical aspect and hence we do not measure it in our experiments.

3. PRECEDENCE MINING

We start by presenting our notation and the precedence probabilities we need to mine. Since course recommendations is our main application, we couch our model in terms of courses taken. However, keep in mind that the model can be applied to movies, books, items, ... that can be watched, read, consumed, and so forth.

We consider a set $\mathcal{D}$ of distinct courses. We use lowercase letters (e.g., a, b, ...) to refer to courses in $\mathcal{D}$. For simplicity we model a *transcript* $\mathcal{T}$ as a sequence of courses, e.g., $a \rightarrow b \rightarrow c \rightarrow d$. (At times we may treat $\mathcal{T}$ as a set of courses abusing notation.) Note that in practice, students take courses concurrently in a quarter or semester, so a transcript is not a strict sequence. Our model and algorithms are easily generalized to sequences of sets of courses.

Top-k Recommendation Problem. Given a set $\mathbb{T}$ of transcripts over $\mathcal{D}$ for n users, the extra transcript $\mathcal{T}$ of a target user, and a desired number of recommendations k, our goal is to:

1. Assign a score $score(x)$ (between 0 and 1) to every course $x \in \mathcal{D}$ that reflects how likely it is the target student will be interested in taking x. If $x \in \mathcal{T}$, then $score(x) = 0$.
2. Using the score function, select the top k courses to recommend to the target user.

To compute scores, we propose to use the following statistics, where $x, y \in \mathcal{D}$:

- $f(x)$: the number of transcripts in $\mathbb{T}$ that contain x.
- $g(x,y)$: the number of transcripts in $\mathbb{T}$ in which x precedes course y.

As discussed in Section 1, we only mine precedence information, not sequencing details (e.g., a immediately follows b, or a follows b follows c, or a follows b two quarters later). The benefit of only focusing on precedence information is twofold: first, we only need to collect on the order of $|\mathcal{D}|^2$ statistics, as opposed to $|\mathcal{D}|!$ statistics that consider all possible course orderings. Second, we do not believe detailed sequence information is very useful for recommendations. In most cases, if course a is of interest to a student, it will be of interest regardless if she takes it the very next quarter, in two quarters, or later. If c is likely to follow a, and c is likely to follow b, then the order in which a and b were taken is not as important compared to the fact that a and b were taken in the past. Of course, our arguments for this second point are not rigorous (to say the least), but given that we can only do precedence mining in a reasonable time in our application, we focus exclusively on precedence mining.

In addition, note that our data set (and in general most recommender system data sets) are sparse, and if we were to collect statistics for $g'(x,y,z)$, which corresponds to three courses taken in sequence, we may not be able to get statistically significant quantities, simply because there may not be enough students in whose transcripts x precedes y and y precedes z. Thus, the two statistics listed above form a good choice for our purposes.

Example 3.1. *We use Table 2 as our running example in the paper. $\mathcal{D} = \{a,b,c,d\}$ is the set of courses that are offered to the students of a university and there are $n = 50$ students. Furthermore, we assume that $\mathcal{T} = \{a,b\}$ is the transcript of the user who needs recommendations. As seen in the table, the number of students who took course a is $f(a) = 10$. Course a occurs before course c in 3 transcripts, hence $g(a,c) = 3$. We can also observe that since a occurs in 10 transcripts but in $g(a,c) + g(c,a) = 3 + 2$ of them it occurs before or after c, then there are 5 transcripts where a occurs without c. Similarly, there is $6-(3+2) = 1$ transcript where c occurs without a.*

3.1 Precedence Mining Model

We next use the quantities $f(x)$ and $g(x,y)$ to compute probabilities that encode the precedence information that will be used by our recommendation algorithms. As motivated in the introduction (Section 1), we use the notation $x^{\neg y}$ to refer to transcripts where x occurs without a preceding y. Similarly, we use $x_{\neg y}$ to refer to transcripts where x occurs without y following it. (Note that a superscript refers to no *preceding* y, while a subscript refers to no *following* y.)

In what follows, we define the probabilities that we will use for computing scores, or that we will use in deriving some of our approximations. Let x and y be courses in $\mathcal{D}$. The probability of x appearing in a transcript is:

$$Pr[x] = \frac{f(x)}{n} \tag{1}$$

The probability of x appearing in a transcript, given that y is present in the transcript is:

$$Pr[x|y] = \frac{g(y,x) + g(x,y)}{f(y)} \tag{2}$$

The probability of y appearing in a transcript without a preceding course x is:

$$Pr[y^{\neg x}] = \frac{f(y) - g(x,y)}{n} \tag{3}$$

The probability of x appearing in a transcript without a succeeding course y is:

$$Pr[x_{\neg y}] = \frac{f(x) - g(x,y)}{n} \tag{4}$$

The probability of x appearing in a transcript, given that y is present in the transcript with no preceding x is:

$$Pr[x|y^{\neg x}] = \frac{g(y,x)}{f(y) - g(x,y)} \tag{5}$$

Note that this quantity above is the same as: $Pr[x_{\neg y}|y^{\neg x}]$, i.e., probability that x is present in the transcript without a following y, given that y is present without a preceding x.

The probability of y appearing in a transcript, given that x is present in the transcript without a succeeding y:

$$Pr[y|x_{\neg y}] = \frac{g(y,x)}{f(x) - g(x,y)} \tag{6}$$

The probability of y appearing in a transcript with no preceding x given that the transcript contains x is:

$$Pr[y^{\neg x}|x] = \frac{g(y,x)}{f(x)} \tag{7}$$

EXAMPLE 3.2. *As seen in Table 2, course a occurs in* 10 *transcripts, with c occurring before a in* 2*, and after a in* 3 *transcripts. Course c occurs in a total of* 6 *transcripts. The total number of transcripts is* 50*. Hence, the probability that any transcript picked at random contains course c (i.e., the probability that a student takes course c) is:* $Pr[c] = \frac{6}{50}$*. The probability that a transcript contains c, given that a is present in the transcript is:* $Pr[c|a] = \frac{2+3}{10}$*.*

The probability that a transcript contains course a without course c preceding a is: $Pr[a^{\neg c}] = \frac{10-2}{50}$*. The probability that a transcript contains course c without being followed by course a is:* $Pr[c_{\neg a}] = \frac{6-2}{50}$*. The probabilities* $Pr[a^{\neg c}]$ *and* $Pr[c_{\neg a}]$ *are found by taking transcripts, eliminating all courses that are neither a nor c and then counting transcripts that match the pattern.*

When deciding if course c is a good recommendation for a student who has taken a, we want to compute: $Pr[c|a^{\neg c}] = \frac{3}{10-2}$*.*

Since we are considering c as a recommendation, then it is clear that c does not precede a in the transcript, so we must rule out transcripts where c precedes a. Note that the quantity above is the same as: $Pr[c_{\neg a}|a^{\neg c}]$*, i.e., the probability that c is present in the transcript without being followed by a, given that a is present without c preceding a because items in a transcript occur only once.*

Our notation can also be used to describe a situation when a *set* of courses does not occur before or after a given course. For example, say $\mathcal{T} = \{a, b\}$. Then $Pr[x^{\neg \mathcal{T}}]$ is the probability that x occurs without either an a or a b preceding it. Similarly, $Pr[x_{\neg \mathcal{T}}]$ is the probability that x occurs without either an a or a b following it. Note that we cannot compute probabilities like these with our f and g statistics. For instance, if we compute $Pr[x^{\neg \mathcal{T}}]$ as $(f(x) - g(a,x) - g(b,x))/n$, we are subtracting out some transcripts twice (those that have both a and b preceding x). However, we can *approximate* values like $Pr[x^{\neg \mathcal{T}}]$. For instance, if we assume few transcripts have both a and b, the expression above may be reasonable. Furthermore, our ultimate goal is to compute scores for recommended courses, so if we compute all scores using the same approximation, the resulting ordering of courses may be similar to the ordering we would obtain with an exact computation. In Section 4.3, when computing set probabilities, we use approximations analogous to the one we illustrated here.

4. RECOMMENDATION ALGORITHMS

4.1 Popularity Algorithm

For reference, we describe a simple algorithm that does not use precedence information but can be easily described in our framework. This approach computes a recommendation score for a course x in $\mathcal{D}$ based on how often x occurs in the transcripts in $\mathbb{T}$. This is computed using f:

$$score(x) = f(x)/n \tag{8}$$

Presenting the most popular items in the system based on user interactions is a common strategy in many web sites. For example, commercial sites, such as Amazon, present the best-sellers, social bookmarking sites present the most popular tags, i.e., the tags that are most heavily used by users to annotate resources in a site, and so forth.

EXAMPLE 4.1. *Course c* ($f(c)/n = 0.12$) *is more popular than d* ($f(d)/n = 0.08$)*, and hence is recommended over d.*

4.2 Single Item Max-Confidence Algorithm

The popularity algorithm does not take into account the courses taken by the target user. Now we would like to take into account "causality", i.e., the principle that there is a necessary relationship between one event (*cause*) and another (*effect*), which is the direct consequence of the first.

Consider a student with a transcript $\mathcal{T}$ of taken courses. For each $y \in \mathcal{T}$, $Pr[x_{\neg y}|y^{\neg x}]$ reflects the likelihood that the student will take course x, ignoring the effect of the other courses in $\mathcal{T}$. One simple heuristic is then to take the maximum $Pr[x_{\neg y}|y^{\neg x}]$ over all $y \in \mathcal{T}$ as an indication of how good a recommendation x is. Of course, this approach ignores joint relationships. For example, a student that has taken Programming I is very likely to take Programming II, except if he has also taken Econ I (e.g., economics students tend to only take one programming course). Our heuristic would highly recommend Programming II in any case.

The Single Item Max-Confidence Algorithm assigns $score(x)$ to be the maximum $Pr[x_{\neg y}|y^{\neg x}]$ value, with one exception however: In computing the maximum, we only take into account courses y that have enough support. That is, we only use courses $y \in \mathcal{T}$ if y and x appear together in sufficient transcripts (implying a strong correlation between y and x). The motivation for this is that we want our computations to be done using statistically significant numbers. More specifically, we define $sup(y, x)$ as follows:

$$sup(y, x) = g(y, x) \tag{9}$$

If $sup(y, x)$ is less than a threshold θ that we set, then we do not use y in the computation of x's score.

SINGLE ITEM MAX-CONFIDENCE SCORE. Given a target user with transcript $\mathcal{T}$ and a support threshold θ, the recommendation score for a course x is:

$$score(x) = \begin{cases} 0 & \text{if } \nexists y \in \mathcal{T} \text{ with } sup(y, x) \geq \theta \\ max(\{conf(y, x) | y \in \mathcal{T}, sup(y, x) \geq \theta\}) & \text{else} \end{cases} \tag{10}$$

$$\text{where } \; conf(y, x) = Pr[x_{\neg y}|y^{\neg x}] = \frac{g(y, x)}{f(y) - g(x, y)} \tag{11}$$

In words, a course x may be recommended on the basis of *its strongest connection to some course* $y \in \mathcal{T}$ as long as this relationship is supported by enough evidence, i.e., it is recorded in many transcripts.

EXAMPLE 4.2. *For our example,* $conf(a, c)$ *is* $3/(10 - 2) = 3/8 = 0.38$. *Similarly,* $conf(b, c)$ *is* $2/7 = 0.29$. *Also,* $conf(a, d)$ *is* $3/10 = 0.3$, *while* $conf(b, d)$ *is* $3/7 = 0.43$. *Assuming that the support is sufficient (i.e., the threshold is 0, say), the course with the highest confidence is d, due to b.*

4.3 Joint Probabilities Algorithm

Our next algorithms take into account how the complete set of courses in a transcript impact the likelihood that a course x is taken later on. Intuitively, we would like to use

$$score(x) = P(x_{\neg \mathcal{T}}|\mathcal{T}^{\neg x}), \tag{12}$$

i.e., the likelihood that x is taken given taken courses $\mathcal{T}$. Unfortunately, as discussed earlier, it is not feasible to compute these quantities exactly, so we must use approximations. The various algorithms we present in this section differ in how they make the approximations.

Assume $\mathcal{T} = \{d_1, d_2, \ldots d_m\}$. For all schemes, we first replace $P(x_{\neg \mathcal{T}}|\mathcal{T}^{\neg x})$ by $Pr[x_{\neg \mathcal{T}}|d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}]$. (Note that we are implicitly performing an approximation by converting the list $\mathcal{T}$ to a set. Without this approximation, we must consider all possible lists $\mathcal{T}$ and compute conditional probabilities based on those lists.) Since it is still infeasible to use the second expression, we perform further simplifications, making independence assumptions.

JOINT P. APPROXIMATION 1. Given a transcript of taken courses $\mathcal{T} = \{d_1, d_2, \ldots d_m\}$, we score a not-taken course x with:

$score(x) = Pr[x_{\neg \mathcal{T}}|d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}] \approx$

$$ct \times Pr[x] \times \prod_{d_j \in \mathcal{T}} Pr[d_j^{\neg x}|x] \tag{13}$$

for some constant ct.

ARGUMENT. We compute $Pr[x_{\neg \mathcal{T}}|d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots d_m^{\neg x}]$ as:

$$\begin{aligned}
& Pr[x_{\neg \mathcal{T}}|d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}] \\
= \; & Pr[x|d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}] && (13.1) \\
= \; & \frac{Pr[x \wedge d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}]}{Pr[d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}]} && (13.2) \\
\approx \; & ct \times Pr[x \wedge d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}] && (13.3) \\
= \; & ct \times Pr[x] \times Pr[d_1^{\neg x}|x] \times Pr[d_2^{\neg x}|x \wedge d_1^{\neg x}] \times \\
& \ldots \times Pr[d_m^{\neg x}|x \wedge d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_{m-1}^{\neg x}] && (13.4) \\
\approx \; & ct \times Pr[x] \times \prod_{d_j \in \mathcal{T}} Pr[d_j^{\neg x}|x] && (13.5)
\end{aligned}$$

Eq. 13.1 holds because all of $\{d_1, \ldots, d_m\}$ have occurred already in the transcript, and therefore none of them can occur in the future. Eq. 13.2 holds due to Bayes rule in probabilities. We then make an assumption that the denominator is a constant not affected by x, since $Pr[d_1 \wedge \ldots \wedge d_m]$ should be nearly the same with or without $\neg x$, giving rise to Eq. 13.3. We then use the Bayes rule again to give Eq. 13.4. Subsequently, we make our second assumption, i.e. that $Pr[d_j^{\neg x}|x \wedge d_1^{\neg x} \wedge \ldots \wedge d_{j-1}^{\neg x}] \approx Pr[d_j^{\neg x}|x]$. This second approximation is because the d_j's are considered conditionally independent given x. □

Combining Eq. 1 and Eq. 7 for computing $Pr[x]$ and $Pr[d_j^{\neg x}|x]$, respectively, Eq. 13 can be written as:

$$Pr[x_{\neg \mathcal{T}}|d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}] \approx ct \times \frac{f(x)}{n} \times \prod_{d_j \in \mathcal{T}} \frac{g(d_j, x)}{f(x)} \tag{14}$$

EXAMPLE 4.3. *Thus, for our given example, our scores for c and d are as follows:*

$$\begin{aligned}
Pr[c_{\neg(\{a,b\})}|a^{\neg c} \wedge b^{\neg c}] \; & \approx \; ct \times \frac{f(c)}{n} \frac{g(a, c)}{f(c)} \frac{g(b, c)}{f(c)} \\
& = \; ct \times \frac{6}{50} \frac{3}{6} \frac{2}{6} = ct \times 2\%
\end{aligned}$$

$$\begin{aligned}
Pr[d_{\neg(\{a,b\})}|a^{\neg d} \wedge b^{\neg d}] \; & \approx \; ct \times \frac{f(d)}{n} \frac{g(a, d)}{f(d)} \frac{g(b, d)}{f(d)} \\
& = \; ct \times \frac{4}{50} \frac{3}{4} \frac{4}{4} = ct \times 6\%
\end{aligned}$$

JOINT P. APPROXIMATION 2. We can compute the joint probability $Pr[x_{\neg \mathcal{T}}|d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}]$ making a slightly different set of assumptions. In particular, we score a not-taken course x with:

$score(x) = Pr[x_{\neg \mathcal{T}}|d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}] \approx$

$$ct \times \min_{d_j \in \mathcal{T}} Pr[x_{\neg d_j}] \times \prod_{d_j \in \mathcal{T}} Pr[d_j^{\neg x}|x_{\neg d_j}] \tag{15}$$

for some constant ct.

ARGUMENT. Our computations are the following:

$$\begin{aligned}
& Pr[x_{\neg \mathcal{T}}|d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}] \\
= \; & \frac{Pr[x_{\neg \mathcal{T}} \wedge d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}]}{Pr[d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}]} && (15.1) \\
\approx \; & ct \times Pr[x_{\neg \mathcal{T}} \wedge d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}] && (15.2) \\
= \; & ct \times Pr[x_{\neg \mathcal{T}}] \times Pr[d_1^{\neg x}|x_{\neg \mathcal{T}}] \times Pr[d_2^{\neg x}|x_{\neg \mathcal{T}} \wedge d_1^{\neg x}] \\
& \times \ldots \times Pr[d_m^{\neg x}|x_{\neg \mathcal{T}} \wedge d_1^{\neg x} \wedge \ldots \wedge d_{m-1}^{\neg x}] && (15.3) \\
\approx \; & ct \times Pr[x_{\neg \mathcal{T}}] \times \prod_{d_j \in \mathcal{T}} Pr[d_j^{\neg x}|x_{\neg \mathcal{T}}] && (15.4) \\
\approx \; & ct \times \min_{d_j \in \mathcal{T}} Pr[x_{\neg d_j}] \times \prod_{d_j \in \mathcal{T}} Pr[d_j^{\neg x}|x_{\neg \mathcal{T}}] && (15.5)
\end{aligned}$$

$$\approx \quad ct \times \min_{d_j \in \mathcal{T}} Pr[x_{\neg d_j}] \times \prod_{d_j \in \mathcal{T}} Pr[d_j^{\neg x} | x_{\neg d_j}] \tag{15.6}$$

Note that we do not discard $\neg\mathcal{T}$ in the first step as in the previous model. The Bayes rule then gives us Eq. 15.1. We then make our first assumption, i.e., that the denominator is a constant not affected by x, since $Pr[d_1 \wedge \ldots \wedge d_m]$ should be nearly the same with or without $\neg x$, to give Eq. 15.2. We then use Bayes rule again giving Eq. 15.3. Our second assumption, i.e., $Pr[d_j^{\neg x} | x_{\neg\mathcal{T}} \wedge d_1^{\neg x} \wedge \ldots \wedge d_{j-1}^{\neg x}] \approx Pr[d_j^{\neg x} | x_{\neg\mathcal{T}}]$, based on conditional independence, gives us Eq. 15.4.

Note that until this point, all assumptions were similar to the previous model, except x is replaced by $x_{\neg\mathcal{T}}$. We now make our third assumption, that of replacing $Pr[x_{\neg\mathcal{T}}]$ with $\min_{d_j \in \mathcal{T}} Pr[x_{\neg d_j}]$ (which is actually an upper-bound) to give Eq. 15.5. Subsequently, we make the assumption that conditioning on $x_{\neg d_j}$ is sufficient since d_j's are independent given x to give Eq. 15.6. □

We can compute $\min_{d_j \in \mathcal{T}} Pr[x_{\neg d_j}]$ as follows:

$$\min_{d_j \in \mathcal{T}} Pr[x_{\neg d_j}] = \min_{d_j \in \mathcal{T}} \frac{f(x) - g(x, d_j)}{n} = \frac{f(x) - \max_{d_j \in \mathcal{T}} g(x, d_j)}{n}$$

Now, $Pr[d_j^{\neg x} | x_{\neg d_j}]$ is calculated in the following way (using Eq. 6):

$$Pr[d_j^{\neg x} | x_{\neg d_j}] = Pr[d_j | x_{\neg d_j}] = \frac{g(d_j, x)}{f(x) - g(x, d_j)},$$

Consequently, Eq. 15 can be written as:

$$Pr[x_{\neg\mathcal{T}} | d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}] \approx$$
$$ct \times \frac{f(x) - \max_{d_j \in \mathcal{T}} g(x, d_j)}{n} \times \prod_{d_j \in \mathcal{T}} \frac{g(d_j, x)}{f(x) - g(x, d_j)} \tag{16}$$

Example 4.4. *For our example, our scores are:*

$$\begin{aligned} Pr[c_{\neg(\{a,b\})} | a^{\neg c} \wedge b^{\neg c}] &\approx ct \times \frac{f(c) - \max\{g(c,a), g(c,b)\}}{n} \\ &\quad \times \frac{g(a,c)}{f(c) - g(c,a)} \times \frac{g(b,c)}{f(c) - g(c,b)} \\ &= ct \times \frac{6-2}{50} \frac{3}{4} \frac{2}{6} = ct \times 2\% \end{aligned}$$

$$\begin{aligned} Pr[d_{\neg(\{a,b\})} | a^{\neg d} \wedge b^{\neg d}] &\approx ct \times \frac{f(d) - \max\{g(d,a), g(d,b)\}}{n} \\ &\quad \frac{g(a,d)}{f(d) - g(d,a)} \times \frac{g(b,d)}{f(d) - g(d,b)} \\ &= ct \times \frac{4}{50} \frac{3}{4} \frac{4}{4} = ct \times 6\% \end{aligned}$$

A natural question to ask at this point is whether the approximations we have made are "reasonable" in practice. There are two ways to approach this question. One is to see if the recommendations made using Approximation 1 or 2 are better than the ones obtained by other schemes. We answer this question in Section 5, where we will see that the algorithms based on these approximations yield very good results in many cases.

A second way to evaluate the quality of the approximations is to directly compare the true values of $P(x_{\neg\mathcal{T}} | \mathcal{T}^{\neg x})$ or $Pr[x_{\neg\mathcal{T}} | d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}]$ with the values provided by our approximations, at least in some cases where it is feasible to compute the exact values. We have also performed this second type of evaluation, which can be found in Appendix A. In summary, these results show that although the approximations are relatively coarse, the relative rankings they provide are very close to the ones provided by the true probabilities. For example, say we fix courses $d_1, d_2, \ldots, d_m$ and we compute the value $Pr[x_{\neg\mathcal{T}} | d_1^{\neg x} \wedge d_2^{\neg x} \wedge \ldots \wedge d_m^{\neg x}]$ for all remaining courses by aggregating over student transcripts, and we produce a ranking of say the top 10 courses based on those true values. We repeat the process using the approximations. When we compare the exact ranking to the approximate ranking (we use three metrics, including Kendall-τ) we see that the differences are not very significant for either of the two approximations, or any of the algorithms described in the following.

In particular, on performing this evaluation with a selected set of CS courses, we find that nearly 9 of the top 10 courses (as inferred from the true probabilities) are present in the top 15 courses recommended by any of our approximations. In addition, the Kendall-τ score is less than 1/4 of other naive methods (indicating that our ordering of courses is close to the ones given by the true probability values). Thus, these results suggest that our approximations are reasonable when ranking courses.

4.4 Joint Probabilities Algorithm: Variants

In this subsection, we describe ways to enhance our two Joint Probabilities Approximation algorithms. To make our discussion more compact, we note that both equations Eq. 14 and Eq. 16 have the following format (given a course x and a transcript $\mathcal{T}$):

$$score(x) = ct \times global(x, \mathcal{T}) \times \prod_{d_j \in \mathcal{T}} local(x, d_j) \tag{17}$$

where the functions *global* and *local* are both ratios of linear combinations of statistics f and g.

4.4.1 Joint Probabilities Support Variant

Consider a course x that has appeared in 1000 transcripts, while y appeared in 10 transcripts. Assume the student has not taken neither x nor y. If there are 20 courses $d_1, \ldots d_{20} \in \mathcal{T}$ such that $\forall d_i, g(d_i, y) = 1$, while $g(d_i, x) = 70$, according to Approx. 1 in Eq. 14, y is recommended over x because:

$$10 \times (1/10)^{20} > 1000 \times (70/1000)^{20}.$$

However, note even though the *local* function as in Eq. 17 evaluates to 0.1 in the case of y and 0.07 in the case of x, we may still prefer recommending x because there is more "evidence", i.e., the pattern has been observed in 1000 transcripts as against 10 in the case of y. In other words, the fraction $\frac{g(d_i,x)}{f(x)} = \frac{70}{1000}$ is more meaningful than $\frac{g(d_i,y)}{f(y)} = \frac{1}{10}$. A similar example can be given for Joint P. Approx. 2.

Thus, we enforce an additional restriction on the set of courses that we can recommend, based on whether we have enough "evidence" to see if x is a good recommendation.

Joint P. Support Variant: Use either Approximation 1 or 2 to compute scores of not-taken courses. However, for any not-taken course x such that $f(x) < \theta$ (for some threshold θ), assign $score(x) = 0$.

4.4.2 Joint Probabilities Hybrid Variant

Consider a set of courses $d_1, d_2, \ldots, d_5$, all appearing in the transcript $\mathcal{T}$. Consider course x that we wish to recommend to a user. It may be the case (especially when the data is sparse) that x is strongly suggested by courses d_1, d_2, d_3, d_4 (i.e., $local(x, d_1)$, $local(x, d_2)$ etc. are high), but we may still not recommend x because of d_5. As a concrete example, d_1, d_2, d_3, d_4 and x could be core CS courses, while d_5 could be an extra-curricular dance course, in which case the ratio $local(x, d_5)$ would simply not be meaningful (especially if very few students from CS have taken the dance course). In this particular situation, $local(x, d_5) = 0$ or very small (for both Approximation schemes), and therefore $score(x) = 0$ (or small). Due to this reason, we may not be able to recommend any courses to the user, especially if the user has taken a course d_5 taken by very few students.

In such cases it is prudent to restrict our attention to the best I ratios in the product $\prod_{d_j \in \mathcal{T}} local(x, d_j)$ in Eq. 17, in the following manner:

JOINT P. HYBRID VARIANT WITH PARAMETER I: (Use either Approximation 1 or 2 throughout.) First, as in the Support variant above, we assign a score of 0 to not-taken courses where $f(x) < \theta$. To assign a score to a remaining course x we proceed as follows. We set top-$I(\mathcal{T})$ to be the top I courses from $\mathcal{T}$ ranked by $local(x, d_j)$ values. (If $|\mathcal{T}| < I$, we set top-$I(\mathcal{T}) = \mathcal{T}$.) Thus, top-$I(\mathcal{T})$ contains the I courses (or fewer) that are the best indicators that x will be taken. We then compute the score of x using only these top-$I(\mathcal{T})$ courses as follows:

$$score(x) = ct \times global(x, \mathcal{T}) \times \prod_{d_j \in \text{top-}I(\mathcal{T})} local(x, d_j) \qquad (18)$$

Also note that if we restrict I to be 1, this method is similar to the Single Item Max Confidence approach in Sec. 4.2, where we use the strongest piece of evidence (in this case, the "strongest" ratio) to predict courses in the remaining portion of the transcript. We can view the model that uses I pieces of evidence as a hybrid between the Joint Probabilities approximation models in the previous section (which uses all the courses in $\mathcal{T}$), and Single-Item Max Confidence (which uses just one course in $\mathcal{T}$), hence the name.

4.4.3 Joint Probabilities Hybrid Reranked Variant

Our algorithms in the previous sections have all focused on optimizing predictability and coverage, without explicitly handling goodness or unexpectedness, which are also important for recommendations (see Section 1).

In order to improve goodness and unexpectedness simultaneously, we need to be able to recommend courses that are in a sense "tailor-made" for the student: they are not popular courses, which is why they are "unexpected" recommendations, but they are relevant to the student, i.e., they form "good" recommendations. Thus, we want to recommend courses with a high score, and yet low popularity. We first select m courses that have high scores from the Joint P. Hybrid algorithm, and then order them on inverse popularity. That is, the least popular course among the top-m scoring ones is ranked as first, the second least popular is ranked as second and so on.

JOINT P. RERANKED VARIANT WITH PARAMETER m: Take the set $\mathcal{O}$ of courses recommended by the Joint P. Hybrid algorithm. If there are fewer than m recommended courses in $\mathcal{O}$, then we use all of them in the reranking phase. If not, we pick the best m courses recommended by the Joint P. Hybrid algorithm. For each course x in the remaining $|\mathcal{O}| - m$ courses, we assign $score(x) = score(x) \times \frac{1}{m}$.

Now we order this "picked" set (of size $\leq m$) in inverse order of f, i.e., if x and y are in the set, x is ranked higher than y if $f(x) < f(y)$. For the course with the lowest f in the top-m set, we assign $score = 1$, for the course with the second smallest f, we give a $score$ of $\frac{m-1}{m}$ and so on, until the last course which is given a $score$ of $\frac{1}{m}$.

Note that we could do this reranking with any of the previous models.As we shall see later on, the hybrid variant is a good choice because it performs very well on predictability (and hence the top-m courses returned by the algorithm are likely to be relevant courses for the student). Notice also that Reranked has the same coverage as the hybrid variant since we assign non-zero scores to the same set of courses.

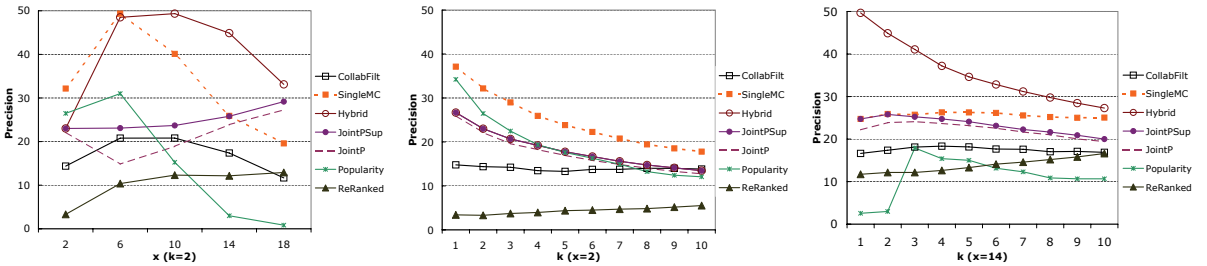
5. EVALUATION

In our experimental evaluation, we analyze the effectiveness of the recommendation algorithms proposed in this paper and we compare them to classical recommendation schemes, namely collaborative filtering and popularity. All the schemes have been implemented in CourseRank. We are using CourseRank's production implementation of collaborative filtering, which is a state-of-art nearest-neighbor approach tuned to the characteristics of the data in CourseRank. Our study is organized in two parts. For the first part, we use transcripts from CourseRank and evaluate recommendations for different parameters (Section 5.1). For the second part, we pick the best schemes and further analyze them via a user study with students (Section 5.2). Due to privacy regulations, public datasets containing customer data with timing information are not available, making it hard to perform this evaluation using other datasets.

5.1 Evaluation Based on Student Transcripts

Our dataset is the set of student transcripts from Stanford University. These transcripts are entered by the students, and may start with the very first course a student took in their academic program at university, or may begin at a point later on. Students are enrolled in various undergraduate programs, and they are in varying degrees of completion of their program. All courses entered are rated by the student, so our dataset is in that sense "the best" we could have. This dataset contains around 7,500 transcripts, which we mined for precedence information. For our experiments, we arbitrarily order the courses taken in a quarter. Consequently, the within quarter ordering is not significant when all transcripts are considered. In spite of this simplification, we find that our Recsplorer algorithms do really well.

In this set of experiments, we allow a recommendation algorithm to "see" x courses from a student's transcript, and we measure how well it does in predicting courses that follow in the transcript, which are of course hidden from the algorithm. Parameter k is the number of recommendations requested and is in the range [1, 10]. We evaluate predictability using *precision@k*, i.e., the fraction of the k recommendations that are present in the remaining "hidden" portion of the transcript [13]. We also want to study the cases where the algorithms "fail" in providing enough recommendations. This happens when for a given k, an algorithm is unable to make k or more recommendations, i.e., compute at least k



(a) all methods vs. x (b) all methods vs. k (c) all methods vs. k

Figure 2: Precision of all recommendation methods.

non-zero recommendation scores. For this purpose, we use a *coverage* metric that is the number of users for whom an algorithm can make at least k recommendations.

Since there may be users for whom an algorithm cannot recommend enough (i.e., $\geq k$) courses (as we will see later in Figure 3), we want to make sure that we use the same set of students for all the methods for each x value tested in order to make the methods comparable. Hence, to compare the predictability of different algorithms, for each x we restricted our attention to transcripts for which all methods can make at least 10 recommendations.

We evaluate: Single-Item Max-Confidence (*SingleMC*), Joint Probabilities Approximation (*JointP*), Joint Probabilities Support Variant (*JointPSup*), Joint Probabilities Hybrid Variant (*Hybrid*), Reranked Hybrid Variant (*ReRanked*), collaborative filtering (*CollabFilt*) and popularity-based (*Popularity*). For the implementation of *JointP*, *JointPSup*, *Hybrid* and *ReRanked*, we use Approximation 1, which is simpler to understand. We found that the behavior of Approximation 1 and Approximation 2 are near-identical for any of the joint probabilities variants above and for different parameters, thus they are not as critical as the overall precedence mining structure of the approximation.

5.1.1 Results

Precision (Predictability): We first present results for precision (predictability) and coverage across all methods (Figures 2 and 3). We set the support threshold θ to be equal to 30 (transcripts) for *SingleMC*, *JointPSup*, and *Hybrid*. We also set the number I of best ratios we use for *Hybrid* to be equal to 3. (We will see later in our discussion about Fig. 5 why these are good parameter choices for these methods.)

Figure 2(a) shows the average *precision*@2 of all methods for x ranging from 2 to 18 courses by increments of 4. For example, for $x = 14$, *Hybrid* does the best with 45% precision. *JointPSup* and *SingleMC* perform similar to each other at around 25%, with *JointP* following closely behind them. *CollabFilt* performs worse, at 17%, and *Popularity* is at 4%. As expected, *ReRanked* does not do well in predicting the courses a user may take (but may still do well in goodness and unexpectedness, as we will see later).

Roughly speaking, given only k=2 best guesses, *Hybrid* can predict at least one course out of two that a student will actually take. This level of precision is significant compared to a maximum precision of 20% for *CollabFilt*. In fact, *CollabFilt* does not do well compared to all of our methods (with the sole exception of *JointP* for $x = 6, 10$). This is because collaborative filtering can only mine recommendations across users similar to the target user whereas our precedence mining can exploit patterns across all users. In this way, precedence patterns can capture a wider variety of relationships in the data. In domains where such precedence patterns exist, such as electronic purchases or web browsing, we expect to have similar good precision.

For small x, *Popularity* does well because it recommends the popular "core" courses that are taken early on in the transcript. For large x, precision is poor since those same courses are likely to be already part of the x courses and hence the method recommends less likely courses.

For small $x \leq 6$, *SingleMC* does the best across all methods. This result seems to indicate that when we have very little to go by (i.e., when x is low), even a single piece of "evidence" is sufficient to make recommendations. For large $x > 6$, *Hybrid* is the best method, and for each x, *Hybrid* does better than *JointPSup*, because it combines only the best evidence (i.e., precedence patterns) to make a recommendation, and, in turn, the latter does better than *JointP* because it filters out weak or noisy evidence.

We thus have different behavior for small vs. large values of x. So we take a closer look at *precision*@k for $x = 2$ and $x = 14$ varying k from 1 to 10 as shown in Figures 2(b) and 2(c), respectively. In both plots, the best method (i.e., *SingleMC* for $x = 2$ and *Hybrid* for $x = 14$) is consistently better than any other method, including collaborative filtering, with increasing k. We observe that both these methods achieve high *precision*@k values for small k. This is good news because with fairly accurate predictions for small k, we do not need to compute many recommendations per user in order to cover the courses a user is likely to take.

In these figures (and Figure 5 later on), we observe that *precision*@k gradually decreases with k. This may be the effect of two phenomena. As k increases our recommendations may include courses with lower scores, which means that they are less likely to be "good" predictions. In addition, the number of hidden courses we are trying to guess may be smaller than k. So precision will naturally drop. For example, if for a given student, we are allowed to "see" x courses from the transcript with 3 remaining courses to guess and we make five recommendations, the best precision we can achieve is 3/5. We study the effect of varying the number of hidden courses in Figure 4.

Finally, in Figure 2(b), *SingleMC* performs better than any of the Joint probabilities methods for small x, because when we have little information, it is better to pick the strongest precedence pattern rather than several weak ones. In addition, recall that we have set I equal to 3 for *Hybrid*, so *Hybrid* ends up selecting all available ratios (which are at most two) and matches the behavior of *JointPSup* (that is why their curves coincide). In Figure 2(c), *Popularity* is the worst method because, as explained earlier, many pop-

ular courses are likely to already be in the input portion of the transcript. Once again, the relative ordering between *JointPSup*, *JointP* and *Hybrid* stays the same, across all k.

Coverage: Figure 3 shows *coverage*, i.e., the number of students for whom an algorithm is able to compute at least k (=10) recommendations, for each x. For example, for $x = 2$, *Popularity* provides 10 recommendations for all 7,500 users. Note that in this plot, *Popularity* can effectively be used as a baseline since it makes enough recommendations for all x (although its precision as seen earlier is low). For example, for $x = 6$ there are around 3,500 transcripts with at least 6 courses. *Hybrid*, *ReRanked* and *SingleMC* are able to provide 10 recommendations for almost all these transcripts. On the other hand, *CollabFilt* can provide 10 recommendations for only 1,800 transcripts. Overall, *CollabFilt* does not do well because it is hard to find enough similar students.

It is noteworthy that our best algorithms in terms of precision, *Hybrid* and *SingleMC*, also achieve high coverage, for all x. Overall, they can make better predictions for more users. This is especially important for small x, i.e., when we know little about a student. For instance, new students, who have been only enrolled one quarter (and need more help than senior students) can take advantage of these recommendations. Observe that even though *ReRanked* has low precision as seen earlier, it has good coverage.

Transcript Size Effect: Since we have widely varying sizes of transcripts in our data set (much like the number of ratings by a user in other recommender systems), we want to explore how our algorithms perform with transcripts of different size. For this experiment, we allow an algorithm to see x=2 courses from a student's transcript and we consider two sets of transcripts: one with $r \leq 5$ remaining courses in each transcript and one with $r \geq 20$. We do the same with $x = 18$ courses. Figure 4 shows *precision@k* on varying k.

For $r \leq 5$, precision is poor for all methods with a maximum around 15% for $x = 2$ (Figure 4(a)) and around 30% for $x = 18$ (Figure 4(b)). Hence, it seems that if there are few remaining courses in a transcript to guess, the more we know about the student the better predictions we make. This fact does not hold for *CollabFilt*, which does not get substantially better. The best method for small x is still *SingleMC*, and for large x is *Hybrid*.

For $r \geq 20$, Figures 4(c) and 4(d) show that the maximum precision rises significantly, reaching 90% (45%) for $x = 2$ ($x = 18$, respectively). Specifically, for $x = 2$ the difference in precision is spectacular compared to $r = 5$. We believe that this behavior is due to some highly popular courses which the students tend to take early on, and both *Popularity* and *SingleMC* capitalize on that fact to make recommendations for those students. However, these courses may not be present in the first $r = 5$ courses that the students take after the first $x = 2$ (which is why Figure 4(a) shows lower precision). (On the other hand, these same courses may already be present in the input set for $x = 18$, explaining why we observe lower maximum precision for Fig. 4(b) and 4(d) compared to 4(c).)

Next, we focus on *SingleMC* and *Hybrid* and study their behavior based on the algorithm-specific parameters, namely support and the number I of best ratios for *Hybrid*, and we explain how we fixed these parameters to the values used in our experiments above. (We do not discuss *JointPSup* since it is superseded by *Hybrid*.)

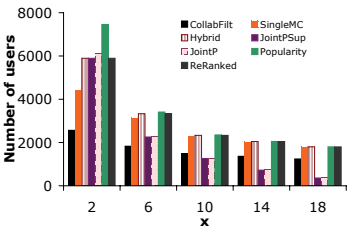


Figure 3: Coverage of all methods vs. x.

Tuning SingleMC and Hybrid methods: As we have seen earlier, *SingleMC* achieves the best precision for small x values. Figure 5(a) shows the variation of *precision@k* with support for various k values and x fixed at 6. As expected, precision drops with k increasing from 1 to 10 (the lines corresponding to different k are stacked in decreasing order of k vertically from top to bottom). *SingleMC* performs best at support equal to 30. Although not shown here, we have found that precision does not improve much if we increase support beyond 30 while the number of users for whom we can perform recommendations decreases. This motivates our choice of support threshold $\theta = 30$ for *SingleMC*.

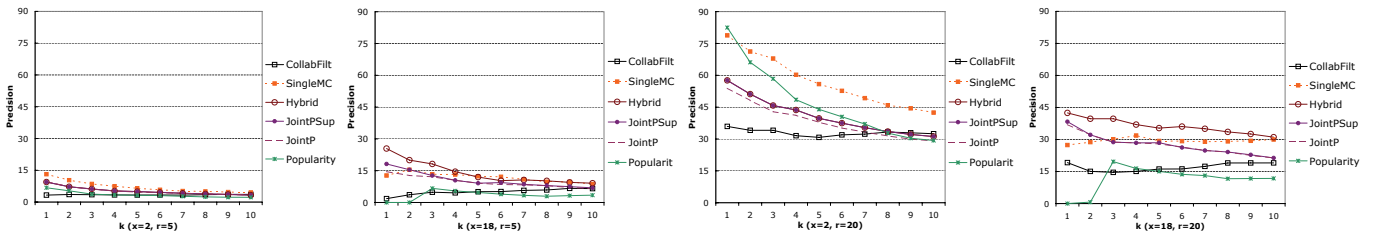
Figure 5(b) shows *precision@k* for *Hybrid* as a function of I and k. We set $x = 14$ and $\theta = 30$ (as seen earlier *Hybrid* does better for larger x.) We observe that combining $I = 3$ ratios performs the best. We have seen that increasing the number of I beyond 5 does not help, mainly because there may be not enough courses among the x taken that can recommend any course strongly. Figure 5(c) presents precision for different support and k (with $I = 3$ and $x = 14$). We find that the variation in precision is not significant between various support thresholds, and hence for comparing all methods in Figure 2, we picked the same support thresholds that we used for *SingleMC*, i.e., support = 30. We have also experimented with $I = 1$ and $I = 5$ and we did not observe much variation on changing the support.

Overall, $I = 3$ and support = 30 comprise good choices for *Hybrid* for large x, justifying our choice in earlier experiments. For small x, our experiments (not shown here for space constraints) show that $I = 1$ is more suitable (which in turn shows that *Hybrid* is similar to *SingleMC* when I is small). Still, even with $I = 1$, *Hybrid* is worse than *SingleMC* for small x.

Figure 5(d) shows precision as a function of x for selected k for both *SingleMC* and *Hybrid* for support=30 and $I = 3$. For small x, *SingleMC* does better. *Hybrid* performs better for just $k = 1, x = 6$ (55% as against 50%), but gets worse for $k \geq 2$. For larger x, *Hybrid* continues to perform well because it exploits the precedence information contained in the data whereas *SingleMC*'s precision deteriorates because even though it can work well on little information, *SingleMC* does not exploit all information when more is available.

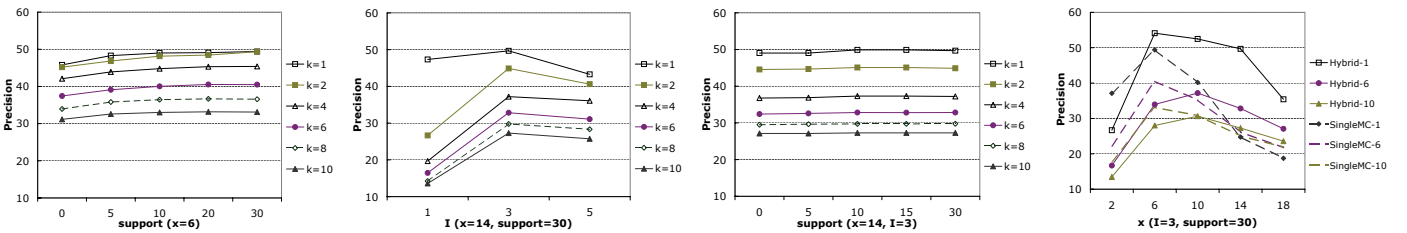
Efficiency: Given the availability of high performance, large memory computers, performance is not a limiting factor for the Recsplorer algorithms, at least in most applications. Thus, instead of presenting detailed performance numbers, we discuss the overall complexity of our algorithms.

Let $\mathcal{D}$ be the set of courses, $\mathcal{T}$ the input transcript and $\mathcal{U}$ the set of users. A recommendation process is divided into an offline phase and an online one. During the offline mode, we need to compute the quantities $g(x, y)$ for each pair of items x and y in $\mathcal{D}$. Essentially, we need to generate an item $\times$ item matrix g of size $|\mathcal{D}| \times |\mathcal{D}|$. If this matrix can fit in



(a) all methods ($x = 2, r = 5$) (b) all methods($x = 18, r = 5$) (c) all methods ($x = 2, r = 20$) (d) all methods ($x = 18, r = 20$)

Figure 4: Effect of x and r for various k.



(a) SingleMC vs. support (b) Hybrid method vs. I (c) Hybrid method vs. support (d) SingleMC and Hybrid vs. x

Figure 5: SingleMC and Hybrid methods.

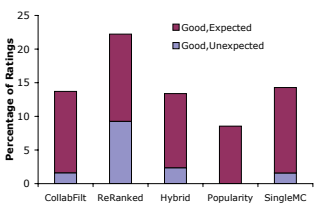


Figure 6: User Study.

main memory, then we can perform all updates in memory. This processing would take $|\mathcal{T}|^2$ time per transcript (we expect the size $|\mathcal{T}|$ of a transcript to be small), with a total of $|\mathcal{T}|^2 \times |\mathcal{U}|$ operations. If the g matrix cannot fit into memory, the offline processing (which is anyway not critical) can be extended in a straightforward way. The resulting g matrix is usually smaller or of similar size as the user $\times$ item ratings matrix used in user-based collaborative filtering, since the number of users is usually $\gg$ number of items.

During the online phase, we use g in order to compute recommendation scores. If g can fit in memory, then the Recsplorer algorithms compute scores of each non-taken course (there are $|\mathcal{D} - \mathcal{T}|$ such courses) by looking up at most $|\mathcal{T}|$ (typically few) entries. Thus the total lookup is $|\mathcal{T}| \times |\mathcal{D} - \mathcal{T}|$, which can be reduced even further if we use a support threshold. If g does not fit in memory, we can perform some additional pre-processing, such as sorting the g values in the offline phase itself, in order to keep the online computation time the same.

5.2 Evaluation Based on User Study

In the previous section, we showed that the Recsplorer algorithms perform well with respect to predictability and coverage Here, we study their performance with respect to goodness and unexpectedness. For this purpose, we constructed an online user study by picking 5 algorithms: the best two algorithms in the previous section, *Hybrid* and *SingleMC*, two traditional recommendation algorithms, *Popularity* and *CollabFilt*, and *ReRanked* (which fared poorly on predictability, but which we expect to do well for goodness and unexpectedness because it selects "tailor-made" courses).

Through email we selected a small subset of CourseRank users to help us improve the system by answering an online survey. For each participating student, we displayed the top 3 recommended courses by each of the 5 algorithms (in random order), and asked the student to answer two questions (*i*) How good is this recommendation? (on a scale from 1(poor)-7(great)) (*ii*) Is this recommendation unexpected? (Yes/No). If the same course is recommended by multiple algorithms, we asked the student about the course only once.

The number of participants in the user study was 48. We gathered a total of 603 responses to each of the two questions listed above. In Figure 6, we plot for each method the percentage of recommendations that were "good and unexpected" courses (lower part of the histogram) and the percentage of "good and expected" courses (upper part). A recommendation is considered good if it is rated 5 or above. As can be seen from the figure, *ReRanked* has the largest percentage of good and unexpected courses (around 10%), while *Hybrid* is second, and *SingleMC* and *CollabFilt* are similar. *Popularity* does not display any unexpected recommendations, since all popular courses are likely to be known to students. These results indicate that while the highly predictive Recsplorer algorithms (*SingleMC* and *Hybrid*) perform just as well as traditional schemes such as collaborative filtering and popularity on goodness and unexpectedness, if we use the reranking approach, we can achieve even better goodness and unexpectedness simultaneously.

Additionally, the height of each histogram indicates the total % of "good" recommendations made by the method. Once again, *ReRanked* performs the best, with around 23% good recommendations, while *CollabFilt, SingleMC* and *Hybrid* all perform similarly at 15%, with *SingleMC* performing slightly better than the other two. *Popularity*, on the other hand, performs fairly poorly, with around 8% "good" recommendations. Thus even if we restrict our attention to the goodness metric alone, the Recsplorer *ReRanked* algorithm

does better than traditional methods, while the other Recsplorer algorithms do just as well or better than traditional recommendation methods.

6. CONCLUSIONS AND FUTURE WORK

In this work, we developed a probabilistic framework for making recommendations based on precedence mining and a suite of recommendation algorithms that use the precedence mining information. These algorithms make different independence assumptions to combat the inherently exponential nature of the problem. We evaluated these algorithms on a data set of student transcripts and we found that they beat popularity-based recommendations and collaborative filtering, both in predictability and coverage, because they effectively exploit temporal patterns across all users. This means that more students can get more and better recommendations. The Single Item Max Confidence approach has the highest precision when we have little information about the student, i.e., a short existing transcript, while Joint Prob. Hybrid works best with more information at hand.

Often, there is a lot of value in presenting to users unexpected, interesting items (courses). To that end, we also compared our algorithms with traditional schemes via a user study to measure goodness and unexpectedness. We found that the Reranked Hybrid algorithm performs much better than traditional schemes both at generating good recommendations or good and unexpected recommendations.

Note that our results can be easily generalized to other scenarios where there is implicit or explicit precedence information, for example, purchases of books, DVDs or electronic equipment. For example, a user might choose to buy a basic computer. He may then want to make his computer more ergonomic, and would then go in for a special ergonomic keyboard, and an optical mouse. Subsequently, he might realize that he needs more RAM or disk space for his applications, so he might upgrade his RAM or hard disk. Similarly, in the purchase of books, a student may first choose to buy basic books on a particular topic, later on choosing to buy advanced books, or even more advanced books. Additionally, since we use simple "purchase" or "consumption" statistics, we can perform well even when there are no ratings available (which is the case in many commercial scenarios).

There are many avenues for future research. For example, in extracting precedence relationships, we could weigh more heavily pairs of courses that follow "close" in time. Incorporating ratings into our models could be another interesting research direction. We would also like to study temporal patterns for a given user, for instance, how a user's interests change over time. In addition, we would like to study how precedence statistics (and therefore "everyone's" interests) that we maintain age over time.

7. REFERENCES

[1] Amazon (2009): url: http://amazon.com/.

[2] A. Parameswaran et. al. Recsplorer: Recommendation algorithms based on precedence mining. Technical report, http://ilpubs.stanford.edu/, Stanford, 2010.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE '95*.

[4] D. Billsus and M. Pazzani. User modeling for adaptive news access. *UMUAI*, 10(2-3), 2000.

[5] J. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *14th UAI Conf.*, 1998.

[6] Y.-H. Chien and E. George. A bayesian model for collaborative filtering. In *Seventh Int'l Workshop Artificial Intelligence and Statistics*, 1999.

[7] M. Deshpande and G. Karypis. Selective markov models for predicting web page accesses. *ACM TOIT*, 4(2), 2004.

[8] Y. Ding and X. Li. Time weight collaborative filtering. In *CIKM*, pages 485–492, 2004.

[9] M. El-Sayed, C. Ruiz, and E. Rundensteiner. FS-Miner: Efficient and incremental mining of frequent sequence patterns in web logs. In *WIDM '04*.

[10] X. Fu, J. Budzik, and K. Hammond. Mining navigation history for recommendation. In *IUI*, 2000.

[11] L. Getoor and M. Sahami. Using probabilistic relational models for collaborative filtering. In *WEBKDD*, 1999.

[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, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *TOIS*, 22:5–53, 2004.

[14] T. Hofmann. Latent semantic models for collaborative filtering. *ACM TOIS*, 22(1), 2004.

[15] S. Jespersen, T. B. Pedersen, and J. Thorhauge. Evaluating the markov assumption for web usage mining. In *WIDM*, 2003.

[16] N. Kawamae, H. Sakano, and T. Yamada. Personalized recommendation based on the personal innovator degree. In *RecSys*, pages 329–332, 2009.

[17] P. Kazienko. Mining indirect association rules for web recommendation. *Int. J. Appl. Math. Comput. Sci.*, 19(1):165–186, 2009.

[18] Y. Koren. Collaborative filtering with temporal dynamics. In *KDD*, 2009.

[19] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3), 1997.

[20] S. McNee, I. Albert, D. Cosley, P. Gopalkrishnan, A. Rashid, J. Konstan, and J. Riedl. On the recommending of citations for research papers. In *CSCW*, 2002.

[21] S. Middleton, N. Shadbolt, and D. D. Roure. Ontological user profiling in recommender systems. *ACM Trans. Inf. Syst.*, 22(1):54–88, 2004.

[22] B. Mobasher, H. Dai, and M. Nakagawa. Effective personalization based on association rule discovery from web usage data. In *WIDM*, 2001.

[23] M. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27:313–331, 1997.

[24] J. Pei, J. Han, and W. Wang. Mining sequential patterns with constraints in large databases. In *CIKM*, 2002.

[25] J. Pitkow and P. Pirolli. Mining longest repeating subsequence to predict world wide web surfing. In *USENIX Symp. on Internet Tech. and Systems*, 1999.

[26] P. Resnick, N. Iakovou, M. Sushak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *CSCW*, 1994.

[27] J. Sandvig, B. Mobasher, and R. Burke. Robustness of collaborative recommendation based on association rule mining. In *RecSys*, 2007.

[28] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In 10^{th} *Int'l WWW Conf.*, 2001.

[29] G. Shani, R. Brafman, and D. Heckerman. An MDP-based recommender system. In *UAI*, 2002.

[30] C. Yu, L. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *EDBT*, 2009.

Top 10 out of 15										
k	Reference	4.1	4.2	4.3a	4.4.1a	4.4.2a	4.3b	4.4.1b	4.4.2b	Random
4	10	0	9	9	9	9	9	9	10	0
3	10	0	10	8	8	9	9	9	9	0
2	10	0	9	6	6	7	6	7	7	0
Kendall-τ Metric										
k	Reference	4.1	4.2	4.3a	4.4.1a	4.4.2a	4.3b	4.4.1b	4.4.2b	Random
4	0	2094	553	612	415	439	543	410	439	2380
3	0	2099	496	672	417	396	594	412	424	2380
2	0	2152	415	799	716	437	417	429	443	2380
Position-Weighted Score										
k	Reference	4.1	4.2	4.3a	4.4.1a	4.4.2a	4.3b	4.4.1b	4.4.2b	Random
4	40	3.23	21.73	20.6	23.82	23.6	20.97	23.95	23.88	0
3	40	3.1	23.07	19.0	23.02	23.77	19.9	23.09	23.17	0
2	40	2.3	24.47	17.38	22.58	23.58	17.85	22.8	23.13	0

Table 3: Ranking Comparison.

APPENDIX

A. RANKING COMPARISON

Our recommendations are based on approximations to Eq. 12, listed here again for convenience:

$$score(x) = P(x_{\neg\mathcal{T}}|\mathcal{T}^{\neg x}), \tag{19}$$

In this evaluation, we study how "close" our approximations are relative to the true probabilities of Eq. 19.

A.1 Methodology

For small datasets we can actually compute Eq. 19 exactly, so in this section we compare the rankings produced by Eq. 19 with the rankings our algorithms produce, in order to evaluate the quality of our approximations.

Say $\mathcal{T} = \{e_1, e_2, \ldots, e_k\}$. Let $x \notin \mathcal{T}$. Let us further assume that we have a set of transcripts, all of contain $\mathcal{T}$. In order to compute exact probability value $P(x_{\neg\mathcal{T}}|\mathcal{T}^{\neg x})$, we do the following.

- Compute $Pr[\mathcal{T}^{\neg x}]$ as follows: Count the number of transcripts that contain all the courses in $\mathcal{T}$, but without the course x in between (i.e., x may appear in the transcript but only after all the courses in $\mathcal{T}$ have happened). Let this number be f_2. Note that even transcripts where there are other courses y in between the ones in $\mathcal{T}$ are counted.
- Compute $Pr[x_{\neg\mathcal{T}} \wedge \mathcal{T}^{\neg x}]$ as follows: Count the number of transcripts that contain x following the courses in $\mathcal{T}$. (There may be other courses in between the ones in $\mathcal{T}$.) Let this number be f_1.
- Compute $Pr[x_{\neg\mathcal{T}}|\mathcal{T}^{\neg x}]$ as: f_1/f_2.

Subsequently, we compute the reference ranking using exact values of the probability as described above: For all $c \notin \mathcal{T}$, we compute $Pr[c_{\neg\mathcal{T}}|\mathcal{T}^{\neg c}]$, and rank these in decreasing order with the largest value first.

For each algorithm in Sec. 4, we assume that there is a transcript available with $\mathcal{T} = \{e_1, e_2, \ldots, e_k\}$. We then use the values returned per algorithm to produce a ranking per algorithm of all courses $c \notin \mathcal{T}$.

We then compare the ranking of each algorithm versus the reference ranking using three methods: (a) Kendall-τ score, which measures the number of pairs that have different orderings in the two rankings (b) top m of n, i.e., how many of the top m in the reference ranking is present in the top n of the rankings of the models (c) position weighted score, which sums up the differences in ranking weighted by a score proportional to the position in the reference ranking. Note that we do not compare the absolute values of the estimates with the reference probability numbers, instead, we prefer to compare rankings, since rankings are sufficient for recommendation.

4.1	Popularity
4.2	Single Item Max Conf.
4.3a	Joint Prob. Approx 1
4.4.1a	Joint Prob. Approx 1 Support Variant
4.4.2a	Joint Prob. Approx 1 Hybrid Variant
4.3b	Joint Prob. Approx 2
4.4.1b	Joint Prob. Approx 2 Support Variant
4.4.2b	Joint Prob. Approx 2 Hybrid Variant

Table 4: Key for Table 3.

A.2 Details and Results

For this experiment, we need complete transcripts (i.e., those that contain all the courses taken by each student until graduation), since incomplete ones could bias our probabilities towards the courses taken early on in the transcript.

If we used a large set of courses $\mathcal{T}$ to make our recommendations, then a fewer number of transcripts would qualify for computation of the exact value of $Pr[x_{\neg\mathcal{T}}|\mathcal{T}^{\neg x}]$, and our actual probability values may not be accurate. We therefore choose a small set of "core" courses $\mathcal{T}$, taken by students early on in the program.

We now describe the details of the specific setup of the experiment. We first chose a small set of 4 core courses taken by Computer Science students early on in their undergraduate program to be our target transcript $\mathcal{T}$. However, we also considered restricting this set to 3 and to 2 core courses, to see the effect of the number of courses on the ranking. We used a set of around 60 complete student transcripts to bootstrap our probability approximation algorithms, then compared the ranking produced by them against the reference ranking using the three methods described above. (For top m of n, we set $m = 10$ and $n = 15$.) The results are displayed in Table 3, which is separated into three parts vertically corresponding to top 10 of 15, Kendall-τ score, and Position-Weighted score.

We consider the ranking relative to the reference ranking for each of the Recsplorer algorithms. The section numbers in the table correspond to the algorithm described in the corresponding section. For example, 4.1 refers to the *Popularity Algorithm*. To distinguish between the joint probability approximation 1 and 2, we use characters a and b in

the table. For example, 4.4.2b refers to *Joint Probabilities Hybrid Variant* for joint probability approximation 2. The key for all these symbols is located in Table 4. Additionally, Reference stands for the reference ranking, while Random stands for the random recommendation (where each course has equal probability).

We set the support threshold to be 30 for each of these algorithms, while we set the number of ratios, I to be 3. (While we set these arbitrarily, we will find that the exact numbers do not matter, since all our methods perform well in this experiment.)

For example, if we consider the top 10 out of 15 metric, for $|\mathcal{T}| = 4$, (i.e., the first row in the table), the reference ranking has all 10 courses in the top 10 (as expected), while the rest of the Recsplorer algorithms have a score of 9 or higher. In fact, the only ranking that has a score of 10 apart from the reference ranking is 4.4.2b (i.e., Joint Prob. Approximation 2 Hybrid Variant). In addition, on examining the rest of the table, we find that the Recsplorer algorithms have only $\approx 500/2000 = 1/4$ of the Kendall-τ score of Popularity and Random, and they have as much as 7 times the Position weighted score of Popularity and Random. (Note, however, that all numbers go down once k is reduced to 2, since it is very hard to recommend courses when just two have been taken: However, even in this case, the Recsplorer algorithms do much better than Popularity and Random.)

These numbers show that the Recsplorer algorithms perform exceedingly well in all three of the ranking comparison methods used, and produce rankings that are "in sync" with the actual rankings if we were to compute the joint probabilities exactly. Thus, our approximations for $P(x_{\neg\mathcal{T}}|\mathcal{T}^{\neg x})$ in Sec. 4.3 are reasonably good.