

It Takes Variety to Make a World: Diversification in Recommender Systems

Cong Yu
Yahoo! Research New York
111 W 40th Street
New York, NY
congyu@yahoo-inc.com

Laks Lakshmanan
Dept. of Computer Science
Univ. of British Columbia
Vancouver, Canada
laks@cs.ubc.ca

Sihem Amer-Yahia
Yahoo! Research New York
111 W 40th Street
New York, NY
sihem@yahoo-inc.com

ABSTRACT

Recommendations in collaborative tagging sites such as del.icio.us and Yahoo! Movies, are becoming increasingly important, due to the proliferation of *general* queries on those sites and the ineffectiveness of the traditional search paradigm to address those queries. Regardless of the underlying recommendation strategy, item-based or user-based, one of the key concerns in producing recommendations, is *over-specialization*, which results in returning items that are too homogeneous. Traditional solutions rely on post-processing returned items to identify those which differ in their attribute values (e.g., genre and actors for movies). Such approaches are not always applicable when intrinsic attributes are not available (e.g., URLs in del.icio.us). In a recent paper [20], we introduced the notion of *explanation-based diversity* and formalized the diversification problem as a compromise between accuracy and diversity. In this paper, we develop efficient diversification algorithms built upon this notion. The algorithms explore compromises between accuracy and diversity. We demonstrate their efficiency and effectiveness in diversification on two real life data sets: del.icio.us and Yahoo! Movies.

1. INTRODUCTION

Web 2.0 has brought us a number of collaborative tagging sites such as Flickr and del.icio.us. The increasing popularity of those sites has made them the ideal destinations for a user to share contents (whether they are generated by the user herself as in Flickr, or by other means as in del.icio.us), express opinions (in the form of tagging and rating) on contents of interest to her, and build connections with other users (whether they are real-life friends or merely people with similar interests). Finding relevant content on those sites, however, has become increasingly difficult due to the enormous amount of content available. There are three main paradigms for locating content in those sites: *browsing*, *searching*, and *recommendation*. Recommendation, in particular, has been gaining growing importance (e.g., see Amazon and Netflix). A recent analysis [3] of 10 million Yahoo! Travel queries shows that more than 90% of the queries are “generic” in nature: the users are not looking for a few specific items (i.e., travel destinations) but for a large

group of items that fit a general description (e.g., “family trip”). Because of this lack of specificity in the query keywords, recommendation becomes the only effective way to differentiate among those items. Furthermore, unlike the first two paradigms, recommendations can be generated and presented to the user without requiring explicit actions from the user. It is therefore not surprising that many sites have begun to adopt recommendation as one of the core mechanisms with which they present content to the users.

Generating good recommendations is a non-trivial task. On one hand, users expect to receive content items that are related to their interests. On the other hand, users get bored quickly if all the recommended items are too similar to each other. For example, during the heat of the US 2008 Democratic Primary Election, the authors visited the topic “election” on del.icio.us and were shown, on the first page, URLs all about “Barack Obama”, and none about “Hilary Clinton”. Such a homogeneous set of “recommendations” could easily turn off the user and lower her interest in the site over time [2]. Like the item-based strategy that del.icio.us employs, most recommendation strategies focus on increasing the relevance of recommended items, thereby increasing the likelihood of returning homogeneous items.

The goal of *recommendation diversification* is to identify a list of items that are dissimilar with each other, but nonetheless relevant to the user’s interests. To our knowledge, almost all of the techniques proposed so far for recommendation diversification [13] are *attribute-based*. In other words, the diversity of the recommended list is defined by how much each item in the list differs from the others in terms of their attribute values. For example, in Y! Movies, recommended movies are post-processed based on the set of attributes that are intrinsic to each movie (including genre, director, etc.). Typically, a heuristic distance formula is defined on the set of attributes in order to allow the system to quantify the distance between a pair of items. In [21], diversity is measured based on a specific attribute of the item which is defined by the taxonomy in which the item is classified. This can also be considered as attribute-based since the classification of each item is intrinsic to that item.

1.1 Drawbacks of Using Attribute-Based Diversification

While suitable for traditional recommender systems where the items are well-described by their attributes (e.g., books in Amazon and movies in Netflix), attribute-based diversification has two main drawbacks.

First, there can be a *lack of attribute descriptions for items*. Items in social network sites often *lack a comprehensive set of attributes*, unlike their counterparts in traditional recommender systems. For example, URLs in del.icio.us are only described by the set of tags associated with them, so are images in Flickr and videos in YouTube. Tags are attached to the items by individual users and are fundamentally different from intrinsic descriptions of the items. Analyzing intrinsic properties of these items, on the other hand, is extremely difficult or computationally prohibitive. For example, to obtain intrinsic descriptions of each URL in del.icio.us, one is required to crawl and analyze the webpage. This is very costly, merely for the purpose of diversification of the recommendations and certainly beyond the purpose and capability of del.icio.us. For multimedia sites like Flickr and YouTube, while image processing and video analysis techniques do exist, the effort required to extract feature vectors for images and videos may be considered too high a price to pay just for diversifying recommendations.

Second, there can be *substantial computational overhead due to attribute retrieval for the purpose of diversification*. A diversification approach based on attributes often requires accessing item attributes in order to perform the diversification: a step that can become costly when the database is very large like in many of the collaborative tagging sites, especially when the attributes are not leveraged by the recommendation strategies (e.g., many collaborative filtering strategies).

1.2 Our Approach

In this paper, we propose to formalize a notion of diversity which relies on *explanations* [2, 14]. For content-based strategies, we define the explanation for a recommended item as a set of similar items that the user has highly rated in the past. The premise for explanation-based diversification in this case is that for two different recommended items i and j , the closer their explanations (i.e., the sets of items that are similar to the recommended items and that are liked by the user), the more homogeneous i and j . Similarly, for collaborative filtering strategies, an item explanation is defined as a set of users who rated that item highly. The intuition for diversification here is that the closer the explanations (i.e., the users who rate the items highly and who are similar to the user), the more homogeneous they are. We can therefore define a distance function for measuring the *diversity distance* between pairs of items based on explanations. It is important to note that explanation-based diversity is a notion which is independent of the presence of item attributes, which means it can be applied to scenarios where the attribute-based diversification does not apply. The effectiveness of explanation-based diversity was established in our recent study [20], where we showed that explanation-based diversification achieves similar levels of diversification as attribute-based diversification on a real data set, Yahoo! Movies.

Our focus in this paper is the exploration of efficient algorithms for explanation-based diversification. In summary, we make the following main contributions. First, we formally define the notion of item explanation for content and collaborative filtering-based recommendation strategies and define the notion of explanation-based diversity for a set of recommended items. It is worth noting that these notions were first formalized by us recently in [20]. Second, we develop efficient algorithms for computing recommendations and explanations on large data sets, an important step towards generating diversified recommendations. Third, we develop efficient diversification algorithms which explore a different balance

between relevance and diversity. Finally, we conduct a comprehensive set of experiments on real data sets for testing the scalability and effectiveness of our algorithms.

The rest of the paper is organized as follows. Section 2 reviews existing memory-based recommendation strategies and formalizes the problem of diversifying recommendations in collaborative tagging/rating sites. Section 3 contains our algorithms for efficient generation of recommendations and explanations for large-scale social tagging/rating sites. Section 4 presents efficient recommendation diversification algorithms. Our experiments are described in Section 5. The related work is reviewed in Section 6. Finally, Section 7 concludes the paper.

2. PRELIMINARIES AND PROBLEM STUDIED

In this section, we provide an overview of existing recommendation strategies and describe the problem studied in this paper. We aim to model collaborative rating sites where users express their endorsements of items¹ and establish ties with other users through explicit links (e.g., friendship network) or implicit ones (e.g., similar ratings of similar movies). We assume we are given a set of users \mathcal{U} and a set of items \mathcal{I} .

2.1 Recommendation Strategies Overview

Given a user, the goal of a recommendation strategy is to estimate the user's ratings of unrated items, based on which recommendations are generated. We explore the two most popular families of recommendation strategies and briefly review them below. These strategies are also referred to as *rating-based* since they predict ratings as opposed to *preference-based* which aim to predict the relative ordering of items. Rating-based strategies rely on finding items similar to the user's previously highly rated items (content-based), or on finding items liked by people who share the user's interests (user-based or collaborative filtering) [2].

2.1.1 Item Based Strategies

These are the oldest recommendation strategies. They aim to recommend items similar to the ones the end-user preferred in the past. The rating of an item $i \in \mathcal{I}$ by a current user $u \in \mathcal{U}$ is estimated as follows:

$$\text{relevance}(u, i) = \sum_{i' \in \mathcal{I}} \text{ItemSim}(i, i') \times \text{rating}(u, i')$$

Here, $\text{ItemSim}(i, i')$ returns a measure of similarity between two items i and i' , and $\text{rating}(u, i')$ indicates the rating of item i' by user u (it is 0 if u has not rated i'). Item based strategies are very effective when the given user has a long history of tagging/rating activities. However, it does have two drawbacks. First, items recommended by item based strategies are often very similar to what the user already knows [13] and therefore other interesting items that the user might like have little chance of being recommended. Second, item based strategy does not work well when a user first joins the system. To partially address those drawbacks, collaborative filtering strategies have been proposed.

¹For tagging sites like del.icio.us, we consider a tagging action as a positive rating without doing detailed tag sentiment analysis.

2.1.2 Collaborative Filtering Strategies

These strategies aim to recommend to an end-user items which are highly rated by users who share similar interests with or have declared relationships with her. The rating of an item i by the end-user u is estimated as follows:

$$\text{relevance}(u, i) = \sum_{u' \in \mathcal{U}} \text{UserSim}(u, u') \times \text{rating}(u', i)$$

Here, $\text{UserSim}(u, u')$ returns a measure of similarity or connectivity between two users u and u' (it is 0 if u and u' are not connected). Collaborative filtering strategies broaden the scope of items being recommended to the user and have become increasingly popular. Note that in both strategies, we use $\text{relevance}(u, i)$ to denote the estimated rating of i by u . *In the rest of the paper, the term relevance refers to this estimated rating.*

We note that there are also so-called fusion strategies which combine ideas from item based and collaborative filtering strategies. While we do not consider them in this paper, it should be straightforward to extend our methods for diversification for those strategies. Another set of recommendation strategies are so-called model based [2], where machine learning techniques are employed. How to provide explanation and diversification for those strategies is an interesting future topic of study, but beyond the scope of this paper.

2.2 Diversity Problem Definition

We first described the notion of explanation-based diversity and the problem of explanation-based diversification in our preliminary study [20]. We briefly describe those concepts here. To begin with, we denote by $\text{RecItems}(u)$, the set of candidate recommended items generated by one of the recommendation strategies described above. The size of this set is typically larger than the final desired number of recommendations.

An explanation for a recommended item depends on the underlying recommendation strategy used. If an item i is recommended to user u by a content-based strategy, then an *explanation* for recommendation i is defined as:

$$\text{Expl}(u, i) = \{i' \in \mathcal{I} \mid \text{ItemSim}(i, i') > 0 \ \& \ i' \in \text{Items}(u)\}$$

i.e., the set of items similar to items (i') that user u has rated in the past. The explanation may contain more information such as the similarity weight $\text{ItemSim}(i, i') \times \text{rating}(u, i')$. If an item i is recommended to user u by a collaborative filtering strategy, then an *explanation* for a recommendation i is:

$$\text{Expl}(u, i) = \{u' \in \mathcal{U} \mid \text{UserSim}(u, u') > 0 \ \& \ i \in \text{Items}(u')\}$$

i.e., the set of users similar to u who have rated item i . Similarly, we can augment each user u' with the similarity weight $\text{UserSim}(u, u') \times \text{rating}(u', i)$.

Note that in all cases, the explanation of a recommendation is either a set of items or a set of users. Based on this, we can define diversity of recommendations as follows. Let i, i' be a pair of items (recommended to user u) with associated explanation sets $\text{Expl}(u, i)$ and $\text{Expl}(u, i')$. Then the *diversity distance* between i, i' for user u can be defined as a similarity measure based on standard metrics such as Jaccard similarity coefficient or cosine similarity. E.g., the *Jaccard diversity distance* between recommendations i and i' is:

$$DD_u^J(i, i') = 1 - \frac{|\text{Expl}(u, i) \cap \text{Expl}(u, i')|}{|\text{Expl}(u, i) \cup \text{Expl}(u, i')|}$$

Note that this is defined as the complement of the standard Jaccard coefficient, since we want to use this as a distance measure. Intuitively, in the case of collaborative filtering, the distance between items depends on the ratio between the number of users who recommend both items and the total number of users who recommend these items.

When weights are incorporated, the *cosine diversity distance* between recommendations i and i' is defined by treating the explanations $\text{Expl}(u, i)$ and $\text{Expl}(u, i')$ as vectors in a multi-dimensional space and defining $DD_u^C(i, i')$ as 1 minus standard cosine similarity between the vectors. We refer the readers to the Vector Space Model [1] for more details. In the case of collaborative filtering, the weighted diversity distance depends on the ratio between the number of similar users who recommend both items and the total number of users who recommend these items. When we want to be neutral with respect to the type of diversity distance measure used and opt for the notation $DD_u(i, i')$. Depending on the context, it shall be interpreted as $DD_u^J(i, i')$ or as $DD_u^C(i, i')$.

For a set of items $S \subseteq \text{RecItems}(u)$, we define:

$$DD_u(S) = \text{avg}\{DD_u(i, i') \mid i, i' \in S\}$$

i.e., $DD_u(S)$ is the average diversity distance between pairs of items in the set S . In this paper, we consider the following problem:

Given a user u , find a subset $S \subseteq \text{RecItems}(u)$ such that $|S| = k$ and the choice of S strikes a good balance between relevance and diversity.

We deliberately leave the term balance undefined and avoid the simplistic treatment of combining relevance and diversity using a weighted sum. Our intention is to explore the relevance/diversity space on real data sets and design efficient diversification algorithms that can intuitively achieve a good balance between the two. In our experiments (Section 5), we use aggregate relevance and aggregate diversity distance of S to gauge the level of balance achieved by our algorithms.

3. EFFICIENT RECOMMENDATION GENERATION WITH EXPLANATION

In this section, we describe the adaptation of content-based and collaborative filtering strategies [2] for efficient recommendation generation in large scale social tagging/rating sites. Furthermore, we describe how to generate explanations efficiently. For simplicity, we mainly discuss collaborative filtering throughout the rest of the paper: the techniques described equally apply to item-based strategies.

For the sites we consider, which have millions of users, identifying networks of similar users for the purpose of collaborative filtering is a computationally extensive task which merits some attention. We dedicate the next section to it.

3.1 Preliminaries

In this section, we describe how to efficiently generate *implicit networks* in large scale collaborative rating sites, which is an essential

step for the efficient generation of recommendations and explanations. We also describe the *storage model* adopted in this study.

3.1.1 Implicit Network Generation

Collaborative filtering strategies (see Section 2.1) estimate the rating of an item for a given user based on the network of the user and how people in her network rate the item. There are two kinds of user networks, *explicit network* and *implicit network*. An example of the former is the del.icio.us friendship network, where users become friends by explicit declaration. However, unlike pure social networking sites (e.g., facebook), such explicit connections are rare in collaborative rating sites where the primary function is to help users organize contents. For example, a recent snapshot of del.icio.us shows that only about 10% of the users have at least one explicit friend and about 1% of the users have at least five explicit friends. This means that if the recommendations are based solely on friendship networks, 90% of the users will not be recommended any result [16].

Implicit networks, on the other hand, can be generated using various mechanisms, one of which is to leverage past behaviors of the users as identified by the items they rated before. Implicit networks provide a nice complement to explicit ones and can often significantly increase the coverage of the users [16]. For example, in del.icio.us, by creating a link between two users who have shared common URLs, we are able to establish an implicit similarity network where 40% of the users have at least one similar user and 25% of the users have at least five similar users. Combined with friendship networks, 45% of the users can now benefit from the recommender system. Another mechanism for generating implicit networks involves using profile information (e.g., age and income) about users. This information is not always available and as a result, shared-item is often the most common mechanism for implicit network generation.

When the number of users is extremely large, generating an implicit network is non-trivial. For example, each month, there are millions of unique users of del.icio.us who have tagged at least one URL. Even at a lower-end estimate of half a million users, a naive algorithm, which does a comparison between all pairs of users, needs 160 billion comparisons. At the rate of 10 micro-seconds per comparison, it will take the algorithm 18 days to finish, which is unrealistic for a web site that is fast evolving.

Most of the comparisons are wasted, however, because an average user shares common URLs with only a small number of users among all the users. In other words, the resulting user-user similarity matrix is often very sparse. Based on this observation, we propose *Algorithm Item-based Similarity Computation* (Algorithm 1) for the generation of implicit shared-item user networks. It achieves efficiency by organizing items based on how many users have tagged them and only does a comparison between two users if the comparison is likely to create a similarity link.

The algorithm starts by constructing the buckets for items (line 1-4), where items are put in the bucket corresponding to the number of users associated with them. We can eliminate a large number of items right away because items associated with only one user are no longer useful. They will not contribute to any link between any pair of users. During the similarity computation, the algorithm starts at the top-most bucket and iterates through all the items in each bucket. For each item, the algorithm iterates through its set of users. And for each user (u), the algorithm identifies all the

Algorithm 1 Item-based Similarity Computation

Require: $B[K]; \{B[K]$ is an array of item lists - each $B[i], i < K$ contains a list of items associated with i users and $B[K]$ contains a list of items associated with K or more users.)

```

1: for each item  $i$  in the system do
2:    $x =$  the number of users associated with  $i$ ;
3:    $B[x].add(i)$ ;
4: end for
5: for  $x$  from  $K$  to 2 do
6:   for each item  $i$  in  $B[x]$  do
7:      $U =$  the set of users associated with  $i$ ;
8:     for each user  $u$  in  $U$  do
9:        $S = \emptyset$ ;  $\{S$  maintains the set of candidate users to be compared with  $u$ , it also tracks occurrences of each candidate user}
10:      for each item  $i'$  associated with  $u$  do
11:         $S = S \cup$  the set of users associated with  $i'$ ;
12:        remove  $u$  from the list of users associated with  $i'$ ;
13:         $y =$  number of users associated with  $i'$ ;
14:         $B[y].remove(i')$ ;  $B[y-1].add(i')$ ;
15:      end for
16:      order users in  $S$  according to their occurrences;
17:      for each user  $u'$  in  $S$  do
18:         $sim = compare(u, u')$ ;  $\{S$  is ordered according to # shared items, we can prune this list of users easily when this number is no longer large enough.}
19:      output:  $(u, u', sim)$ ;
20:    end for
21:  end for
22: end for
23: end for

```

users (S) associated with the set of items that u is associated with—those are the candidate users to be compared with u . Since the number of items shared between u and each user in S is obtained in the process for free, similarity threshold-based pruning can be easily applied here. For example, for Jaccard similarity, there is one simple and yet effective pruning method that we adopt. The number of shared items divided by the number of items of u gives us an upper bound on the actual similarity. If this upper bound is smaller than the threshold, we know that we can simply throw that pair away. Furthermore, u can now be thrown away from the user lists of all the items, which allow the items to be moved down the buckets (lines 12-14).

3.1.2 Storage Model

With the generation of the implicit network, we now have all the pieces needed for recommendations. Here, we briefly describe how the system stores and manages its data. There are two main pieces of information: the *network* and the *actions*. The network keeps information about the similarity between two users (or items in the case of item based recommendation). It can be given directly from the underlying system (e.g., del.icio.us) or it can be created as we just described. The actions are the rating histories of all the users. Those are always given by the underlying system and are always in the order of when the action happened. Both can be easily modeled as database tables (see Table 1 for the schemas). Personal attributes of users and descriptive attributes of items are often maintained as well when they are available.

Network	Action
src: int	user: int
dest: int	item: int
similarity: float	rating: float

Table 1: Database Schemas.

While a native storage system is possible, we decided to be compatible with the underlying system and adopt a database (MySQL 5) as the back-end and only go to our own storage when necessary. As a result, most of the accesses that we will be describing in this section are implemented in the form of SQL queries.

3.2 Generating Recommendations

We recall the collaborative filtering recommendation formula in Section 2.1:

$$\text{relevance}(u, i) = \sum_{u' \in \mathcal{U}} \text{UserSim}(u, u') \times \text{rating}(u', i)$$

Given a user du , for whom a list of recommendations are to be generated, the goal is to generate a list of items with the top- k estimated ratings. In a straightforward way, this process can be realized through the following three SQL queries:

Q1: obtaining user network:

```
Udu = select C.dest, C.similarity
      from Network C
      where C.src = du and C.similarity > th1
```

Q2: obtaining candidate items:

```
I = select A.item, A.user, A.rating, Udu.sim
    from Action A, Udu
    where A.user in Udu.dest and A.rating > th2
```

Q3: generating recommendations:

```
R = select I.item, SUM(I.rating * I.sim) as score
    from I
    group by I.item
    order by score
    limit k
```

In the actual implementation, the three queries can be nested into a single SQL call to reduce the overhead of remote access. Thresholds th_1 , th_2 , and k are predefined thresholds that the system adopts.

3.3 Generating Explanations

In this section, we describe efficient approaches for explanation generation for collaborative filtering recommendation strategies. The approaches we describe here can equally apply to item based recommendation strategies with straightforward adaptation.

Recall that the explanation of a recommended item di , for a given user du , is defined as the set of users (called *contributors*) who contributed to the score of the item. For each contributor c , her contribution is: $\text{UserSim}(du, c) \times \text{Rating}(c, di)$. In the simple version, we treat each contributor equally and the explanation for (du, di) is the set of contributors. In the advanced version, we associate each contributor with their contributions and the explanation for (du, di) is the set of weighted contributors.

3.3.1 Post-Processing Approach

A naive approach to computing explanations is to obtain the list of recommended items as described in Section 3.2, and then issue the following query to retrieve the set of contributors and their contributions:

Q4: generating explanations:

```
E = select A.item, A.user as contributor,
      (A.rating * C.similarity) as contribution
    from Action A, Network C, R
    where C.src = du and A.item in R.item
          and A.user = C.dest and A.rating > 0.0
```

The system then collects the results and assembles all the contributors of a single item into a weighted list. This post-processing approach is not efficient since it does the aggregation of items twice, once in the recommendation generation, another time in the explanation generation, when it performs list aggregation. Another approach is to issue a single query for each item being recommended thus avoiding the second (list) aggregation. But the overhead of issuing numerous database queries quickly offsets all the gains obtained by avoiding the extra aggregation and therefore we don't consider this further.

3.3.2 Integrated approach

A more efficient approach avoids the double aggregations by maintaining a view over the `Action` table where all the actions belonging to the same user are stored together as a single list on the disk, and within each list, the actions are sorted on ratings. Specifically, consider generating candidate recommendations for user du . From the `Network` table, we obtain users who are similar to du and the similarities. From those user lists, we obtain ratings assigned by those users on various items in descending order of their score. We can now apply an algorithm such as the NRA Algorithm [8] to compute the candidate recommended items efficiently. More interestingly, since each user has a different similarity to du , we can use this similarity to adjust the rate at which we move down each list. As each candidate item i is being generated, we get for free, the set of users similar to du who rated i and their contributions.

4. DIVERSIFICATION ALGORITHMS

Given a set of candidate items, $\text{RecItems}(u)$, and a given threshold K , the optimal scenario of recommendation is finding a set of items, which has the highest diversity distance (See Section 2.2) and the highest aggregated relevance. In general, however, such an optimal top- K answer set does not exist: maximizing diversity does not always correlate with the most relevant answers being chosen. As a result, we need to explore a balance between relevance and diversity. In this section, we present four alternative algorithms for identifying a subset of K items out of the set of N ($> K$) candidate items with different levels of tradeoff between relevance and diversity. Two of those are optimal algorithms: *Algorithm MaxRel* maximizes the relevance of the K items returned while *Algorithm MaxDiv* maximizes the diversity distance of the K items. Since the primary purpose of a recommender system is to recommend relevant results, we propose to preserve the high scoring items *as much as possible*, leading to the design of the other two algorithms, *Algorithm Swap* and *Algorithm (Binary/Iterative) Greedy*. Both are heuristic algorithms where we try to maximize the diversity under relevance constraints.

Algorithm 2 Algorithm MaxRel

```
Require:  $u, K$ ;
1: SortedRecItems = Sort RecItems( $u$ ) on score;
2: RetList = topKItems(SortedRecItems,  $K$ );
3: return RetList
```

Intuitively, Algorithm MaxRel (Algorithm 2) simply extracts the top- K highest scoring items from the pool of items to be recom-

mended. While achieving the highest relevance, the diversity of the result set may suffer.

Algorithm 3 Algorithm MaxDiv

Require: u, K ;

- 1: $\text{RecItems} = \text{RecItems}(u)$;
- 2: $d = 0.0$; (d is the current maximum diversity)
- 3: $\text{RetList} = \emptyset$;
- 4: **for** each item set S of size K in RecItems **do**
- 5: **if** $\text{Diversity}(S) > d$ **then**
- 6: $d = \text{Diversity}(S)$;
- 7: $\text{RetList} = S$;
- 8: **end if**
- 9: **end for**
- 10: **return** RetList

Algorithm MaxDiv (Algorithm 3) on the other hand, iterates through all possible combinations of K items and identifies the one with the highest diversity to maximize the diversity of the result. There are two clear drawbacks of this algorithm. First, it maximizes the diversity without considering relevance at all and as a result can produce result sets that have very low relevance. Second, iterating through all possible item sets of size K out of N items is extremely expensive: $O(N^K)$ when $N \gg K$.

The two algorithms above represent two extremes in the two dimensional (relevance and diversity) space. We next design two heuristic algorithms that balance the relevance and diversity of the resulting item set.

4.1 Algorithm Swap

The basic idea behind Algorithm Swap (Algorithm 4) is to start with the K highest scoring items, and swap the item which contributes the least to the diversity of the entire set with the next highest scoring item among the remaining items. At each iteration, a candidate item with a lower relevance is swapped into the top- k set if and only if it increases the overall diversity of the resulting set. To prevent a sudden drop of the overall relevance of the resulting set, an optional pre-defined upper-bound UB (on how much drop in relevance is tolerated) can be used to stop the swapping when the lowest relevance of the remaining items is no longer high enough to justify the swap. The selection of the item to be swapped is accomplished by searching over all items in the current top- K set S and picking item i with the minimum diversity. More precisely, let $D_S^i = \sum_{j \in S, j \neq i} DD_u(i, j)$ (where DD_u is described in Section 2.2), we pick the item $i \in S$ with the minimum D_S^i as the candidate for swap.

Intuitively, Algorithm 4 pays the price of a relevance drop for the benefit of increasing diversity. Thus, the case where the top K items are retrieved (Algorithm 2) is a special case of Algorithm 4 where UB is set to zero. The initial swap candidate selection takes $O(K^2)$ and each subsequent selection (up to $N-K$ iterations, where N is the size of $\text{RecItems}(u)$) costs $O(K \lg K)$ to select the next candidate if a swap is needed². Therefore, Algorithm 4 has an overall worst case complexity of $O(NK \lg K)$.

4.2 Algorithm Greedy

²This is possible because we only compare the candidate item to the entire current set of results, which we empirically found to work well. Otherwise, the complexity is $O(K^2)$.

Algorithm 4 Algorithm Swap

Require: u, K, UB ; (UB is an optional score upper-bound used as a stopping condition);

- 1: $\text{SortedRecItems} = \text{Sort RecItems}(u)$ on score;
- 2: $\text{RetList} = \text{topKItems}(\text{SortedRecItems}, K)$;
- 3: $\text{pos} = K+1$;
- 4: **for** each item i in RetList **do**
- 5: $M.\text{add}(i)$; (M is a Heap maintaining each item i in RetList in increasing order of their D_{RetList}^i .)
- 6: **end for**
- 7: $i = M.\text{remove}()$;
- 8: **while** $i.\text{score} - \text{SortedRecItems}[\text{pos}].\text{score} \leq \text{UB}$ **do**
- 9: **if** $D_{\text{RetList}}^i < D_{\text{RetList}}^{\text{SortedRecItems}[\text{pos}]}$ **then**
- 10: $\text{RetList}.\text{remove}(i)$;
- 11: $\text{RetList}.\text{add}(\text{SortedRecItems}[\text{pos}])$;
- 12: **for** each item j in M **do**
- 13: update D_{RetList}^j ;
- 14: **end for**
- 15: $M.\text{add}(i)$;
- 16: $i = M.\text{remove}()$;
- 17: (i is a new candidate for swapping)
- 18: **end if**
- 19: **if** there is no more item left in SortedRecItems **then**
- 20: **break**;
- 21: **end if**
- 22: **end while**
- 23: **return** RetList

Another heuristic approach is to start with the most relevant item, greedily add the next most relevant item if and only if that item is far enough away (compared with a distance bound) from existing items in the set, and stop when there are K items. The drawback, however, is that the distance bound is often hard to obtain and can vary from user to user.

To address this problem, *Algorithm Greedy* (Algorithm 5) relies on iteratively refining two diversity thresholds: an upper-bound UB , initially set to 1, and a lower-bound, LB , initially set to 0. The algorithm first iterates through the set of candidate items in the order of their relevances and generates two lists: DivList , and SimList . The DivList contains items that are all maximally distant from each other, while the SimList contains items that have zero distance to some items in DivList . Because items are accessed in the order of their relevance, SimList essentially contains items that we no longer consider. If DivList already contains enough items, we pick K most relevant items from it can return. If not, we adjust UB and LB and reiterate through the candidate items. At each iteration, the KeepList tracks items that will definitely be in the result set while the DiscardList tracks items that should be eliminated from consideration. These two lists are merged with DivList and SimList , respectively, at the end of each iteration. The algorithm stops when K are found in DivList or the difference between UB and LB drops below a threshold.

At each pass, the worst case complexity is $O(NK)$, the number of passes is limited since the bounds are limited between 0 and 1 and the algorithm stops when the difference between LB and UB is smaller than 0.01 (up to 9 passes). We also note here that, while the complexity for Algorithm Greedy is lower than that of Algorithm Swap, in practice, Algorithm Greedy often takes longer time to run because of the multiple passes it needs to make³.

³i.e., Algorithm Greedy has a bigger constant.

	del.icio.us	Y! Movies
# distinct active users	413K ⁴	3.3M
# distinct items	3.7M	52K
# total actions	6.5M	21M
avg. # items per user	15.7	6.4
avg. # users per item	1.8	403

Table 2: Summary of Data Sets

Algorithm 5 Algorithm (Binary/Iterative) Greedy

```

Require:  $u, K$ ;
1: SortedRecItems = Sort RecItems( $u$ ) on score;
2: UB = 1, LB = 0;
3: DivList = SortedRecItems[0];
4: SimList =  $\emptyset$ ;
5: for  $i = 1$ ; SortedRecItems[ $i$ ]  $\neq$  NULL;  $i++$  do
6:   if  $D_{DivList}^{SortedRecItems[i]} \geq UB$  then
7:     DivList.add(SortedRecItems[ $i$ ]);
8:   end if
9:   if  $D_{DivList}^{SortedRecItems[i]} \leq LB$  then
10:    SimList.add(SortedRecItems[ $i$ ]);
11:   end if
12: end for
13: if  $|DivList| \geq K$  then
14:   return  $K$  highest scoring items in DivList;
15: end if
16: RemainList = SortedRecItems - SimList
17: if  $|RemainList| < K$  then
18:   return RemainList  $\cup$   $\{K - |RemainList|$  highest scoring items
   in SimList $\}$ ;
19: end if
20: while  $|UB - LB| > 0.01$  do
21:   Bound =  $(UB+LB)/2$ ;
22:   KeepList =  $\emptyset$ 
23:   DiscardList =  $\emptyset$ 
24:   for  $i = 1$ ; SortedRecItems[ $i$ ]  $\neq$  NULL;  $i++$  do
25:     item = SortedRecItems[ $i$ ];
26:     if DivList.has(item)  $\vee$  SimList.has(item) then
27:       continue; {This item is either in already or should be discard
       always.}
28:     end if
29:     if  $D_{DivList \cup KeepList}^{item} \leq Bound$  then
30:       KeepList.add(item);
       {To improve performance, we can break out the for loop here
       if  $|KeepList \cup DivList| > K$  and there are many remain-
       ing items.}
31:     else
32:       DiscardList.remove(item);
33:     end if
34:   end for
35:   if  $|KeepList \cup DivList| < K$  then
36:     DivList = DivList  $\cup$  KeepList;
37:     UB = Bound;
38:   else if  $|KeepList \cup DivList| > K$  then
39:     SimList = SimList  $\cup$  DiscardList;
40:     LB = Bound;
41:   else
42:     DivList = DivList  $\cup$  KeepList;
43:     break;
44:   end if
45: end while
46: if  $|DivList| \neq K$  then
47:   add  $K - |DivList|$  highest scoring discarded items to DivList;
48: end if
49: return DivList;

```

5. EXPERIMENTAL EVALUATION

We implemented our algorithms with JDK 5.0 on an Intel machine with dual-core 3.2GHz CPUs, 4GB Memory, and 500GB HDD, running Red Hat Enterprise Linux 5. The Java Virtual Memory size is set to 1GB. All performance numbers are obtained as the average of three runs. We ran our tests on two real life data sets: del.icio.us and Y! Movies. We start by briefly describing them.

del.icio.us (<http://del.icio.us/>) is a popular online social tagging site, where users can tag and share their bookmarks. A single action within del.icio.us is modeled as a 5-tuple: $\langle \text{user}, \text{URL}, \text{tags}, \text{time}, \text{private} \rangle$, where *tags* is a bag of words the user chooses to associate with the URL, *time* records the last time this action was modified, and *private* indicates whether the tagging action is to be kept private. We map actions in del.icio.us to our data model by appending each action with a rating of 1.0 and ignore the tags themselves. For the purpose of this study, we randomly extracted a subset of recent public tagging actions from the del.icio.us site over one month.

Y! Movies (<http://movies.yahoo.com/>) is an online movie rating and reviewing site, where users can rate movies on a scale of 0 to 100, and provide detailed movie reviews. A single action within Y! Movies is modeled as a 5-tuple: $\langle \text{user}, \text{movie}, \text{rating}, \text{review}, \text{time} \rangle$, which can be mapped to our data model in a straightforward way by converting the rating into the range of $[0.0, 1.0]$ and ignore the reviews. For the purpose of this study, we examined a snapshot of the site dated at 2005.

The summary statistics of both data sets are shown in Table 2⁴. The major distinction between them are the number of unique items within the system. This is not surprising since there are many more interesting and unique URLs in the world than there are movies.

5.1 Implicit Network Generation

We first evaluate the performance of *Algorithm Item-based Similarity Computation* (Algorithm 1), which is used to generate implicit user-user similarity networks that are based on shared items (e.g., URLs and movies) between two users. For del.icio.us, the similarity is computed as the Jaccard similarity:

$$sim(u_1, u_2) = \frac{|u_1.URLs \cap u_2.URLs|}{|u_1.URLs \cup u_2.URLs|}.$$

For Y! Movies, it is computed as the adjusted Jaccard similarity:

$$sim(u_1, u_2) = \frac{|u_1.movies \Theta u_2.movies|}{|u_1.movies \cup u_2.movies|},$$

where a movie rated by both u_1 and u_2 satisfies the Θ condition if and only if the ratings given by both are within 30 of each other (on a 0-100 scale).

Intuitively, the naive approach compares each user against each other user in the system. Our item-based algorithm takes advantage of the fact that the user-user similarity matrix is often very sparse,

⁴The number of distinct active users for del.icio.us is a random subset during the study period only and does not account for private tagging actions, the number for the entire site is much higher.

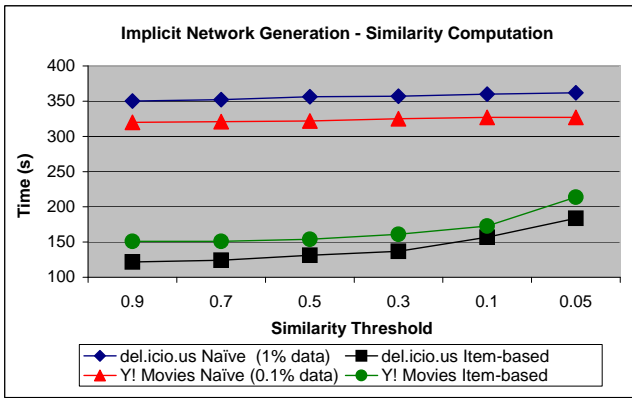


Figure 1: Performance comparison between naive and item-based similarity computation algorithms: the naive algorithm can not finish on the full data sets for either del.icio.us or Y! Movies, and we randomly sample 1% and 0.1% of the data, respectively.

	SI/th=0.9	SI/th=0.5	SI/th=0.1	Friends
del.icio.us				
# links	88K	262K	6.2M	131K
user cov.	3.1%	6.9%	39.5%	7.5%
Y! Movies				
# links	5.3M	8.3M	17.9M	N/A
user cov.	4.6%	5.9%	8.3%	N/A

Table 3: Summary of explicit (Friends) and implicit user networks (SI for shared interests) under various similarity thresholds (th) over the study period.

and a comparison is necessary only when the two users shared at least a certain number of items. As shown in Figure 1 and Table 3, the item-based algorithm significantly outperforms the naive algorithm. On both del.icio.us and Y! Movies, the item-based algorithm is able to finish processing the entire dataset in a few minutes even when the similarity threshold is set to 0.05 (see the discussion below for an important remark on Y! Movies). In comparison, the naive algorithm failed to finish processing the full dataset within 24 hours: the numbers in Figure 1 for the naive algorithm are based on running the algorithm on random subsets of the full datasets. In Table 3, # links are the number of user-user links in the network and user cov. are calculated as the percentage of distinct users with at least one friend over the total number of active users. For the rest of the experimental evaluation, we choose similarity threshold 0.1.

As expected, the naive algorithm is not affected by the similarity threshold since it always compares all possible pairs of users. The item-based algorithm, on the other hand, becomes more expensive as the similarity threshold goes down and more links are created between the users. However, as long as the resulting graph remains sparse (i.e., linear in the number of users), the algorithm is able to perform well. For the item-based algorithm, Y! Movies presents a unique challenge: there are about 1000 movies (2% of all movies) that are extremely popular that we call *heavy-hitter items*. Given any 5 heavy-hitter movies, the number of users that rated them all can be in the range of hundreds of thousands. In other words, the sub-network involving heavy-hitters is extremely dense. As a result, the pruning power of item-based algorithm no longer applies: pair-wise comparison of hundreds of thousands of users will require 10B comparisons and that’s just for 5 heavy-hitters! For now, we use the heuristic that if a user has rated a heavy-hitter movie, we assume she will have a base similarity to all other users who have rated heavy-hitter movies, and the similarity is proportional to the

	Low	Medium	High
del.icio.us friendship			
# users	121	110	118
avg. network size	1.0	1.5	12
del.icio.us shared-url			
# users	334	319	330
avg. network size	1.0	7.2	155
Y! Movies shared-movie			
# users	540	545	494
avg. network size	3.9	38.4	192.2

Table 4: Statistics of sample groups.

number of heavy-hitter movies she has, regardless of which heavy-hitter movies are rated. The statistics in Table 3 do not include those heuristic similarities, hence the low user coverage for Y! Movies. When we do consider those similarities, the coverage for the case of th=0.1 reaches 90%. How to improve the accuracy of similarity estimation in an efficient way for datasets like Y! Movies with such heavy-hitter characteristics is the subject of a future study.

5.2 Evaluation of Explanation Generation

Explanations do not come for free. Next, we analyze the additional cost associated with the generation of explanations during the recommendation process. Formally, the overhead of explanation generation is defined as *the extra processing time incurred to retrieve explanations compared to the processing required for only retrieving the recommendations*. The easy way to evaluate that would be to randomly sample the active users and record the average costs for producing the recommendation-only results, and for producing recommendations with explanations, for all users in the sample.

However, the cost for generating recommendations and retrieving explanations often varies considerably from user to user. In collaborative filtering recommendation, one highly indicative factor of those costs is the number of people in the user’s network. Given a user for whom the recommendations are to be generated, the larger her network is, the larger the number of potentially relevant items the recommendation system will have to go through, and therefore the higher the cost for generating recommendations.

Considering the effect of network size, we take the following more fine-grained and more informative sampling approach: First, we order the active users according to the number of users in their networks. Second, we evenly divide the list into five buckets, where the first bucket contains users with the highest number of users in their network and the fifth bucket contains users with the lowest number of users in their network. Third, we discard the second and fourth buckets and keep the first bucket (*high bucket*), third bucket (*medium bucket*), and fifth bucket (*low bucket*). Finally, for all three buckets that we keep, a 1% random sample is drawn and the resulting groups of users are called *high*, *medium*, and *low* groups, respectively. All subsequent experiments are carried out on all three sample groups. Table 4 illustrates the characteristics of all the sample user groups that we use in our experiments. Clearly, users in the high sample group for both shared item (shared URLs in del.icio.us and shared movies in Y! Movies) networks have a significantly higher number of users in their network.

As described in Section 3.3, recommendation explanations can be generated using either the post-processing approach or the integrated approach. In the post-processing approach, the recommendations for the given user are produced first and then for all the items being recommended, we query the database to retrieve the contributors to the recommendations. In the integrated approach,

	Low	Medium	High
del.icio.us friendship			
avg. rec./user	2.2	8.7	9.8
avg. explanation/rec.	1	1.2	6.6
del.icio.us shared-url			
avg. rec./user	4.89	9.41	9.93
avg. explanation/rec.	1	1.71	18.3
Y! Movies shared-movie			
avg. rec./user	5.61	9.72	9.95
avg. explanation/rec.	1.56	3.74	7.11

Table 5: Statistics of recommendations and explanations with number of recommendations capped at 10.

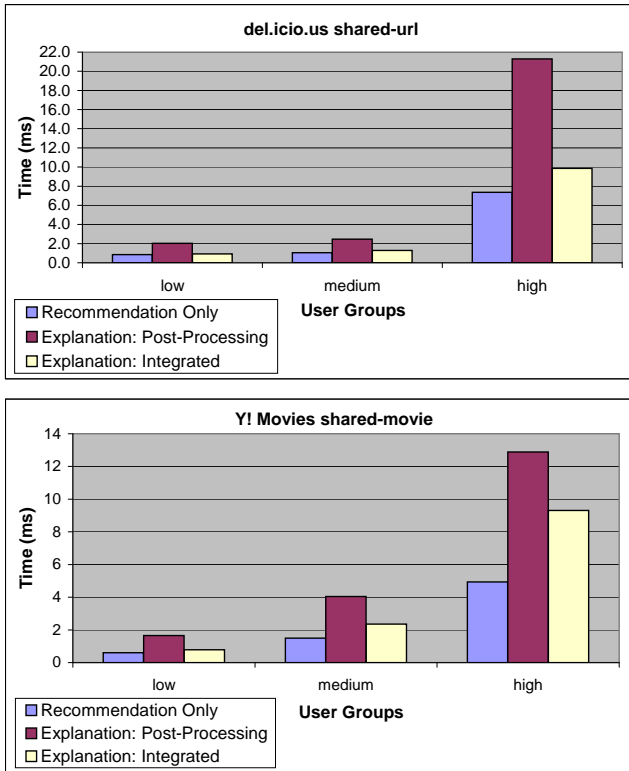


Figure 2: Average costs for generating recommendations and explanations.

we retrieve the set of users (potential contributors) in the network of the given user, and for each such user, the list of items she has rated in the descending order of the ratings. We then run our differential-speed NRA algorithm to generate the recommendation and explanations simultaneously.

Table 5 illustrates the recommendation results on all three different networks and for users in the three sample groups. As expected, the numbers of recommendations for users in the low group (i.e., users with few numbers of other users in their networks) are low compared with the other two groups. In particular, for del.icio.us recommendations using the friendship network, some users in the low group may not even have recommendations produced for them because their friends may not have tagged anything over the study period. For the other two groups, the number of recommendations approaches the pre-defined upper limit. Similarly, the number of explanations for each recommendation grows with the increasing network size.

As shown in Figure 2, the integrated approach for explanation generation significantly outperforms the post-processing approach: it reduces the overhead to less than 5% for users in the low and medium groups, and just a bit over 50% for users in the high group. The cost savings are mostly due to the fact that in the post-processing approach, the aggregation on items is effectively performed twice: once in the retrieval of the recommendation results (done inside the underlying database) and another time in the retrieval of explanations (done outside of the database). In the integrated approach, aggregation is done only once and not all items are fully analyzed. Often the user lists are already in memory during the process of previous recommendations, further speeding up the process.

It is somewhat surprising at the beginning that the average cost for producing a list of recommendations is higher for users in high groups of del.icio.us than those of Y! Movies, because Y! Movies users actually have more people in their network (192) than del.icio.us users (155) (Table 4). After some analysis, we realize that this is due to the fact that there are more items per user for del.icio.us users (15.7) than for Y! Movies users (6.4) (Table 2). In other words, to generate a recommendation for a del.icio.us user, the system will examine twice the number of items it examines when generating a recommendation for a Y! Movies user.

Finally, we note here that the results on the friendship network based recommendation for del.icio.us is similar and is therefore omitted for simplicity.

5.3 Evaluation of Diversification Algorithms

Next, we evaluate the effectiveness and efficiency of our explanation-based diversification algorithms described in Section 4 on medium and high group users. We ignore the low group since the size of the recommendation list is often smaller than the limit and therefore there is no need for diversification.

We demonstrated in our recent work [20] that explanation-based diversification (using both Algorithm Swap and Algorithm Greedy) can achieve similar diversification results as compared with attribute-based diversification (using the same two algorithms) on the Yahoo! Movies data set. Furthermore, when a collaborative filtering recommendation strategy is adopted, explanation-based diversification performs faster due to the fact that it does not have to retrieve attributes for the purpose of diversification. Here, we further perform two more experiments. The first is to evaluate the performance of each algorithm and understand the feasibility of adopting recommendation diversification for real systems. The second is to evaluate the quality of each algorithm in terms of the relevance and the diversity they achieve.

5.3.1 Performance

The performance numbers for high group users are shown in Figure 3⁵. *Algorithm MaxRel* is omitted: it takes no additional time since in most cases it simply returns the top-k result directly from the recommendations. *Algorithm MaxDiv* is much more costly than all three other algorithms, often runs in minutes, even when the number of candidate results are limited to the top 25 for a top-10 recommendation (if Algorithm MaxDiv is left to run on all candidate results, it does not finish in hours in most cases.) The two heuristic algorithms, *Algorithm Swap* and *Algorithm Greedy*, run in reasonable time, with the former a bit faster than the latter. This is

⁵Numbers for medium group user are similar.

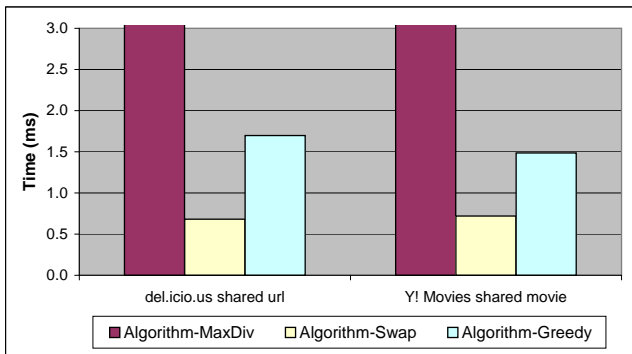


Figure 3: Average costs for recommendation diversification for users in high group.

high group	MaxRel	MaxDiv	Swap	Greedy
del.icio.us shared-url				
aggregate relevance	3.07	1.23	2.31	2.45
diversity	0.90	0.98	0.98	0.98
Y! Movies shared-movie				
aggregate relevance	9.05	5.46	8.77	8.74
diversity	0.81	0.93	0.91	0.95
medium group	MaxRel	MaxDiv	Swap	Greedy
del.icio.us shared-url				
aggregate relevance	0.32	0.16	0.20	0.24
diversity	0.90	1.00	0.97	0.96
Y! Movies shared-movie				
aggregate relevance	1.87	0.57	1.05	1.63
diversity	0.77	1.00	0.98	0.96

Table 6: Relevance and diversity of recommendation results from various diversification algorithms.

because the former runs through the candidate list once while the latter may have to go through the list multiple times.

5.3.2 Quality

The quality of each algorithm is shown in Table 6. Not surprisingly, Algorithm MaxRel achieves the best relevance in all cases, but relatively low diversity. We observe that the diversities are in general very high even when we are optimizing for relevance, and especially higher for recommendations lists for users in the high group than those in medium groups because the former group of users have more people in their networks. In general, Algorithm MaxDiv should be able to find a recommendation list with diversity at 1.0 (i.e., all items being recommended have a different set of explanations). It often does achieve that. However, due to its high computational complexity, we have to limit the number of candidate results for it to consider under 25, which reduces its capability to find the best recommendation list in terms of the diversity. As expected, the two heuristic algorithms, Algorithm Swap and Algorithm Greedy, increase the diversity at the expense of relevance, compared with results of Algorithm MaxRel. In some cases, the relevance drop can be significant. This is especially notable in Algorithm Swap, because it is very susceptible to characteristics of the top results in the recommendation set. E.g., suppose the top result happens to have a much higher relevance than every other item, but at the same time is very close to other items (diversity-wise). It is now a candidate for swapping and we are faced with a dilemma. Either we allow the swap to happen by removing the maximum relevance drop threshold imposed and getting a huge reduction in relevance, or we do not swap and get the same result as we would have obtained from Algorithm MaxRel. We have chosen the former approach because we would like to explore the characteristics of the algorithm more.

In summary, Algorithms Swap and Greedy are able to strike a good balance between relevance and diversity efficiently. In general, the former should be chosen when performance is more important and the latter should be chosen when losing the top scoring items is too much of a risk.

5.4 Discussion

It is worth pointing out that the standard precision/recall metrics have been known to not be able to measure the benefits of the non-traditional properties (e.g., diversity) of recommendation results [13]. As a result, the ultimate measure of the benefits of explanation and explanation-based diversification is to let the users provide us with feedback in the form of a user study or to carry out live testing online and examine users' click through data. There are several studies that have been done before to this effect and interested readers are referred to those studies [17, 21] to learn more about the benefits of explanation and diversification for recommendations. We are also in the process of initiating online bucket testing with del.icio.us.

6. RELATED WORK

The notion of diversity has been studied in many different contexts. We discuss those studies in the context of recommender systems, Web search and database queries.

6.1 Recommender Systems

Most recommender systems, with the exception of [21], focus on increasing the relevance of recommended items and neglect diversity. More recently, there has been a push toward going beyond improving relevance of recommendations [13, 10]. In [21], the authors introduce an order-independent intra-list similarity metric to assess the *topical diversity* of recommendation lists and a topic diversification approach for decreasing the intra-list similarity. Similarity is computed by mapping items to taxonomies to determine topics or using item features such as author and genre. The method is based on an exhaustive post-processing algorithm which operates on a top- N list to compute the top- K results ($N > K$).

The need for explaining recommendations is discussed in [14] and [17]. Particularly in [17], the authors analyze the importance of personalizing explanations to users, the source of recommendations, user mood, etc. The study concludes that explanations need to be tailored to the user and the context and that explanation sources matter. This work constitutes a good motivation for considering explanation-based diversification. Herlocker et. al. [9] studied explanation interfaces and found that, out of twenty-one explanation interfaces for a movie recommender system, participants were most likely to see a movie if they saw a histogram of how similar users had rated the item, with the "good" and "bad" ratings clustered separately. Using another dataset, Bilgic and Mooney [5] have shown that histogram-based explanations can be more persuasive than intended, causing items to be over-estimated.

6.2 Web Search

To the exception of [4] and [15], most Web search engines often enforce diversity over (unstructured) data results as a post-processing step [6, 19]. In [15], the authors develop a query reformulation method used to re-rank the top- N search results such that documents likely to be preferred by the user are presented higher. They

observe how large numbers of users modify their search queries in order to detect the kinds of results which tend to be missing from the top of search results from the user's perspective. Different query reformulations are selected based on pre-determined user interests. The effectiveness of the approach is evaluated for different user interests. The method developed in [4] relies on sampling Web search results in order to reduce homogeneity. It is based on associating results with taxonomies and invoking a basic sample-next(p) call that samples term postings with probability p. The authors also show how to construct sample-next(p) methods for Boolean operators from primitive methods. The approach described in [12] is based on clustering Web search results into groups of related topics. Clusters reflect different user interests.

6.3 Database Queries

Chen and Li [7] propose a notion of diversity over structured results which are post-processed and organized in a decision tree to help users navigate them. In [11], the authors define the Précis of a query as a generalization of traditional query results. For example, if the query is "Jim Gray", its précis would be not just tuples containing the terms, but also additional information such as publications, and colleagues. The précis is diverse enough to represent all information related to the query terms.

In [18], the authors study the problem of efficiently computing diverse query results in online shopping applications, where users specify queries through a form interface that allows a mix of structured and content-based selection conditions. They introduce a hierarchical notion of diversity in databases. For example, when querying a database of cars, diversity can be enforced on Make first, and then on Model. They develop efficient top-k processing algorithms. Our formal definition of diversity can be viewed as more general given that it allows a flexible combination of relevance scores and of diversity.

7. CONCLUSIONS

One of the popular means for content delivery used by collaborative tagging and rating sites is recommendation. Traditional recommender systems focus on bringing forth recommendations that maximize their estimated rating by the user. This can come at the expense of diversity. As pointed out in [13], recommendations lacking diversity can fail to engage the user, as the recommendations tend to be too homogeneous to be exciting. To mitigate this, we study the problem of generating diversified recommendations in a principled manner. Previous work has mainly relied on objective attributes of items to achieve diversity. We identify the limitations of such attribute-based approaches: attributes may not always be available and it is often difficult to balance the notion of diversity among multiple attributes. We propose a formal notion of explanation for recommendations, based on which we propose an approach for recommendation diversification using distance between pairs of items in terms of their explanations. We also develop efficient algorithms for generating recommendations, which achieve a good balance between relevance and diversity. With a detailed set of experiments, we show that our algorithms achieve very good diversity levels while imposing a small overhead on top of the traditional recommendation generation. Furthermore, we empirically demonstrate the levels of balance between relevance and diversity achieved by our algorithms.

8. REFERENCES

- [1] http://en.wikipedia.org/wiki/vector_space_model.
- [2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6), 2005.
- [3] S. Amer-Yahia, L. Lakshmanan, and C. Yu. SocialScope: Enabling information discovery on social content sites. In *CIDR*, 2009.
- [4] A. Anagnostopoulos, A. Z. Broder, and D. Carmel. Sampling Search-Engine Results. In *WWW*, 2006.
- [5] M. Bilgic and R. Mooney. Explaining recommendations: Satisfaction vs. promotion. Beyond Personalization Workshop. In *IUI*, 2005.
- [6] J. Carbonell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, 1998.
- [7] Z. Chen and T. Li. Addressing Diverse User Preferences in SQL-Query-Result Navigation. In *SIGMOD*, 2007.
- [8] R. Fagin and et. al. Optimal Aggregation Algorithms for Middleware. In *PODS*, 2001.
- [9] J. Herlocker, J. Konstan, and J. Riedl. Explaining collaborative filtering recommendations. In *CSCW*, 2000.
- [10] J. A. Konstan. Introduction to recommender systems. In *SIGIR*, 2007.
- [11] G. Koutrika, A. Simitsis, and Y. Ioannidis. Précis: The Essence of a Query Answer. In *ICDE*, 2006.
- [12] K. Kummamuru, R. Lotlikar, S. Roy, K. Singal, and R. Krishnapuram. A Hierarchical Monothetic Document Clustering Algorithm for Summarization and Browsing Search Results. In *WWW*, 2004.
- [13] S. McNee, J. Riedl, and J. A. Konstan. Being Accurate Is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems. In *CHI*, 2006.
- [14] P. Pu and L. Chen. Trust Building with Explanation Interfaces. In *IUI*, 2006.
- [15] F. Radlinski and S. T. Dumais. Improving Personalized Web Search using Result Diversification. In *SIGIR*, 2006.
- [16] J. Stoyanovich, S. Amer-Yahia, C. Marlow, and C. Yu. A Study of the Benefit of Leveraging Tagging Behavior to Model Users' Interests in del.icio.us. In *AAAI Spring Symposium on Social Information Processing*, 2008.
- [17] N. Tintarev and J. Masthoff. Effective Explanations of Recommendations: User-Centered Design. In *RecSys. ACM SIGCHI*, 2007.
- [18] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. Amer-Yahia. Efficient Online Computation of Diverse Query Results. In *ICDE*, 2008.
- [19] D. Xin, H. Cheng, X. Yan, and J. Han. Extracting Redundancy-Aware Top-k Patterns. In *SIGKDD 2006*, 2006.
- [20] C. Yu, L. Lakshmanan, and S. Amer-Yahia. Recommendation diversification using explanations. In *ICDE*, 2009.
- [21] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, 2005.