Minimal Interaction Search in Recommender Systems

Branislav Kveton
Adobe Research
321 Park Ave, San Jose, CA, United States
kveton@adobe.com

Shlomo Berkovsky
CSIRO
Vimiera Rd, Marsfield, NSW, Australia
shlomo.berkovsky@csiro.au

ABSTRACT

While numerous works study algorithms for predicting item ratings in recommender systems, the area of the user-recommender interaction remains largely under-explored. In this work, we look into user interaction with the recommendation list, aiming to devise a method that allows users to discover items of interest in a minimal number of interactions. We propose *generalized linear search* (GLS), a combination of linear and generalized searches that brings together the benefits of both approaches. We prove that GLS performs at least as well as generalized search and compare our method to several baselines and heuristics. Our evaluation shows that GLS is liked by the users and achieves the shortest interactions.

Author Keywords

Human-computer interaction; interactive search; generalized binary search; recommender systems; active learning

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

INTRODUCTION

Recommender systems are widely used in eCommerce sites and online social networks as a means to help users cope with the information overloading. The recommenders select, in a personalized manner, a set of items from a substantially larger set of candidate items and present the recommended items to users. The selection is based on the predicted ratings, reflecting the match of the items to the preferences and needs of the users. Normally, N top-scoring items are included in the recommendation list.

Although numerous works focused on the algorithms for improving the accuracy of item selection [20], little research looked into the intricacies of the presentation of recommendations [11]. The canonic way of presenting the recommendations is through a list, the items of which are ordered according to their predicted rating. The choice of the item to consume – a movie to watch, a product to buy, or a song to listen to – is done by the user upon examining the recommendation list. To ease navigation, the list is normally is split into small-size batches, such as 10 items per screen for a desktop interface or 3-4 items for a mobile app.

For a large N, the examination of the recommendation list can be tedious. Consider also the challenge of choosing the desired item out of a set of attractive recommendations, and the importance of the recommendation list presentation that allows easy discovery of items of interest becomes clear.

In this work, we focus on minimizing the length of user interaction with the recommender. We assume that the predicted item ratings are computed by the recommender and leave the details of rating computation beyond the scope of our work. Given the predicted ratings, we focus on N top-scoring items, assuming that an item of interest is included there. Note that the user may not know the exact item they are looking for and therefore cannot use the search functionality. However, the user may be able to describe the features of the item of interest (e.g., "something like that Irish black comedy I watched last week") and answer questions about the target item. We aim at developing a method for the recommendation list presentation that facilitates the discovery of the target item in least interactions (clicks, scrolls, etc.) between the user and the recommender.

The simplest approach to searching recommended items is what we refer to as *linear search* (LS): the items are ranked according to their predicted ratings and the users scroll through batches of items until they find the target item. A natural generalization of LS is *linear search with item categories* ($\mathtt{LS}_{\mathtt{cat}}$), which capitalizes on the knowledge of item categories. Here, the users are first shown the list of item categories, out of which they choose the category that best matches the target item. Then the users search for the target item in a smaller list of items in the selected category. This approach is adopted by several recommenders, e.g., the 'bands' interface deployed by Netflix. However, our simulations show that the discovery of the target item with LS and $\mathtt{LS}_{\mathtt{cat}}$ requires many user-recommender interactions.

But why does the item category selection occur only once? Can this be repeated in order to carry out a hierarchical search over dynamically computed item categories? This kind of search can be considered as a question-answering game [26]. At each round of the game, the recommended items are dynamically split into K item categories, which are presented to the user who selects the category of the target item. The game stops when only a single item – the target item – is left. This game is a form of *active learning* and is known as *generalized*

IUI 2015, March 29–April 1, 2015, Atlanta, GA, USA.

http://dx.doi.org/10.1145/2678025.2701367

Interaction 1 of GLS User is shown 8 categories (GS)
Action Thrillers
Foreign Regions
Ages 11-12
20th Century Period
Based on the Book
Classics
Drama
Something Else

Interaction 2 of GLS User is shown 8 categories (GS)
Romance Classics
Mobster
Thrillers
Based on Bestsellers
Crime Dramas
Drama
Classic Dramas
Something Else

Interaction 3 of GLS User is shown 8 categories (GS)
Contemporary Literature
Crime Thrillers
Mystery
Courtroom Dramas
Sci-Fi Horror
Crime Dramas
Based on Bestsellers
Something Else

Interaction 4 of GLS User is shown 8 movies (LS)
Misery (1990)
Primal Fear (1996)
General's Daughter (1999)
Silence of the Lambs (1991)
Dolores Claiborne (1994)
Pelican Brief (1993)
Boys from Brazil (1978)
Instinct (1999)

Figure 1. Searching for a movie like *Silence of the Lambs (1991)* with GLS. The choices of the user are highlighted in bold.

*search (*GS*)* [10, 12, 18, 5, 25]. In large-scale problems, GS is expected to find the target item in less user interactions than LS and LS_{cat}. The main limitation of GS is that the number of interactions can be high when the space of recommended items cannot be partitioned well into item categories.

In this work, we propose a combination of GS and LS, which brings together the benefits of both approaches. GS is a hierarchical approach and therefore it converges in $O(\log(N))$ interactions when the categories of recommended items are sufficiently diverse. However, in the worst-case, all N items may belong to the same category and GS may need $\Omega(N)$ interactions to find the target item. On the other hand, LS does not depend on the item categories and is efficient when some recommended items are more likely to be chosen than others. We combine the two approaches and suggest switching from GS to LS when the former is suboptimal. The pivotal question here refers to the *switching criterion*, i.e., when to switch from GS to LS. We argue that GS should be applied as long as the *expected length* of user interaction in GS is shorter than that in LS. We denote the proposed combined approach as *generalized linear search* (GLS).

We illustrate GLS with an example of searching for a movie like "Silence of the Lambs" (Figure 1). At the first interaction, GS suggests 8 movie categories and the user selects the best matching category, "Based on the Book". The search space gets reduced accordingly. At the second interaction, the expected cost of GS is smaller than that of LS. Therefore, GS suggest another 8 categories, the user selects the best matching category, "Thrillers", and the search space gets reduced again. The third interaction is still GS and the user selects again a movie category. At the fourth interaction, the expected cost of GS is larger than that of LS. Therefore, GLS switches to LS and the user is shown 8 movies that are consistent with the user's previous answers. Then the user selects "Primal Fear" as the desired movie. Overall, the length of this user interaction is 4.

In this work, we prove that GLS performs at least as well as GS, and compare it experimentally with GS and several heuristic search methods. We evaluate the expected length of user interaction with the recommender using two datasets and show that GLS achieves shorter interactions than GS. We also show that the expected length of GLS interaction is never longer than that of the heuristic methods. However, the performance of the heuristic methods requires an a-priori parameter tuning, whereas GLS is parameter-free. We also conduct an intra-group user study that compares the binary variants of GLS and GS. The study shows that GLS achieves shorter interactions and is easier to use. The subjects also express their direct preference towards GLS.

Overall, the proposed GLS approach combines the state-of-the-art in active learning with a widely used practical technique, and shows a significant improvement in using both. It should be highlighted that GLS combines two existing search methods, GS and LS, in an optimal way. The presented combination has several notable properties and contributions. First, GLS is guaranteed to perform no worse than each of its parts, GS and LS, and we prove this formally. Second, GLS requires no tunable parameters and can be easily applied in practice. Third, GLS improves significantly over both GS and LS, and we show this by both offline experiments and a live user study. In the experiments, we report improvements as large as 50%.

PRELIMINARIES

In this section, we discuss two policies for searching a list of recommended items, LS and LS_{cat}. We evaluate the policies on one problem and use this to motivate our work.

We formalize the problem of minimal interaction search as follows. The user has a *target item* e^* in mind, which belongs to a set of N *recommended items* $E = \{1, \dots, N\}$. The user cannot express what the item is, otherwise the user could find the item using conventional search techniques, like text search. The target item e^* is drawn i.i.d. from the distribution π over items E. The distribution π is generated by a recommender and is known, but the target item e^* is unknown. The goal is to present items E to the user such that e^* can be discovered in *least interactions*. More formally, let A be an interface for searching items E and $N_A(e^*)$ be the number of user interactions with A until e^* is discovered. Then our goal is to design an interface that minimizes $\mathbb{E}_{e \sim \pi}[N_A(e)]$.

Our abstraction of interactive search can model various problems. For instance, the search interface A can be a list of items, which are ordered according to their popularity, or an assistant on a mobile device that asks natural language questions. The interaction can be clicking on the screen of a device, scrolling down the list of items, or answering a natural-language question.

Linear Search

Linear search (LS) is a form of search, where items E are sorted in descending order according to their score. The sorted items are then scanned by the user until the target item e^* is discovered. This search interface is prevalent in existing recommender systems.

Algorithm 1 LS: Linear search.

Input:
Items $V \subseteq E$
Number of shown items L

Let $e_1, \ldots, e_{|V|}$ be an ordering of items V such that:
$\pi(e_1) \geq \ldots \geq \pi(e_{|V|})$
$\ell \leftarrow 0$
repeat
$A \leftarrow \{e_i \in V : \ell L < i \leq (\ell + 1)L\}$
Show items A to the user
$\ell \leftarrow \ell + 1$
until $(e^* \in A)$

Output: Target item $e^* \in A$

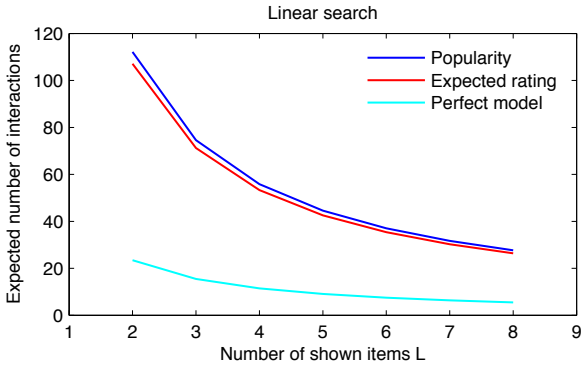

Figure 2. The length of interaction for three LS policies.

In this work, we assume that the items are divided into batches of size L. For the sake of concreteness, and without loss of generality, we assume that L is the number of items displayed on the screen of a device. Therefore, the length of interaction to find item e^* is the number of screens that the user has to scan until the target item e^* appears on the screen. The pseudocode of LS is in Algorithm 1. The score of item e is the probability of choosing the item, $\pi(e)$.

We present a motivational simulation, which compares three LS policies that differ in the computation of the item scores. The first policy scores items by their popularity, i.e., the number of times that the item was consumed. We refer to this policy as *popularity*. The second policy scores items by their expected rating, which is computed by averaging all the available ratings for the item. We refer to this policy as *expected rating*. In the last policy, the score of the item is the actual rating assigned by the user. The score of unrated items is 0. This policy is obviously unrealistic and cannot be implemented in practice, because the ratings of recommended items are typically unknown. However, the policy can be viewed as a lower bound on the length of interaction of any LS policy, under the assumption that the true preferences of the user are known. Therefore, we refer to this policy as *perfect model*.

We evaluate the expected length of user interaction with all three policies using the *MovieLens* dataset, which is described in detail in the evaluation section. The expected interaction length is computed as follows. Initially, we randomly choose a target user. Then, we randomly choose a target item e^* from the movies that the user liked, i.e., rated with 4 or 5 stars. All movies are scored according to the chosen policy: *popularity*, *expected rating*, or *perfect model*. Finally, we rank the movies according to their scores and simulate LS until the target item e^* is found. The length of interaction is averaged over all users and target items, and we vary the number of shown items L from 2 to 8. The results of the simulation are reported in Figure 2.

We observe two trends. First, the expected length of interaction in all three policies decreases with the number of displayed items L. This is expected; as L increases, the length of interaction required to find a target item cannot increase. Second, we observe that the predicted item score affects the expected length of interaction. In particular, the *popularity* policy consistently performs worse than the *expected rating* policy, which performs worse than the *perfect model* policy. Interestingly, *popularity* performs only slightly worse than *expected rating*, while *perfect model* significantly outperforms the other two policies. On average, it finds the target item in about 20% of interactions of the *popularity* policy.

Linear Search with Item Categories

Linear search with item categories ($\texttt{LS}_{\texttt{cat}}$) is a variant of LS, where the users are first asked to choose the category of the target item e^*, out of K item categories. After the category is chosen, all items in that category are searched using LS. Note that $\texttt{LS}_{\texttt{cat}}$ involves two types of user interaction. The first interaction is choosing one item category out of K. The remaining interactions are identical to LS – scanning the items until the target item e^* is discovered. This type of an interface is not unusual; e.g., Netflix 'bands' allow users to choose a movie genre and then scan through the movies in that genre.

$\texttt{LS}_{\texttt{cat}}$ is parameterized by the number of item categories K that are shown to the user. We represent each category as a set $q \subseteq E$ such that $e \in q$ if and only if item e belongs to category q. The problem of choosing K item categories is an optimization problem. Loosely speaking, the best K categories partition the set of items E into K disjoint sets of cardinality N/K each. In this case, the set E is reduced to N/K items regardless of the item category chosen by the user. In practice, such a partitioning is typically impossible because the categories are not sufficiently diverse.

How to partition E into K item categories? We implement a greedy strategy for choosing the categories that was proposed by Bhamidipati *et al.* [5]. For a given K, we greedily select $K-1$ item categories $q_1, \ldots, q_{K-1}$ whose cardinality is closest to N/K. In addition, we define a special category, $q_K \leftarrow \overline{(q_1 \cup \ldots \cup q_{K-1})}$, which includes all items that do not belong to any of the first $K-1$ categories. The key advantage of this strategy is that any item in E is guaranteed to belong to at least one category q_k. Therefore, the user can always choose a category that matches the target item e^*. After the category q_k is chosen, the set of candidate items is reduced to $q_k \cap E$ and we apply LS. The pseudocode of $\texttt{LS}_{\texttt{cat}}$ is shown in Algorithm 2.

Algorithm 2 $\texttt{LS}_{\texttt{cat}}$: Linear search with item categories.

Input:
Number of item categories K in a question
Number of shown items L

Choose $K-1$ categories $q_1, \ldots, q_{K-1}$ given E and π
$q_K \leftarrow \overline{(q_1 \cup \ldots \cup q_{K-1})}$
Ask the user to choose one category
The user chooses category q_k
$e^* \leftarrow \texttt{LS}(q_k \cap E, L)$

Output: Target item e^*

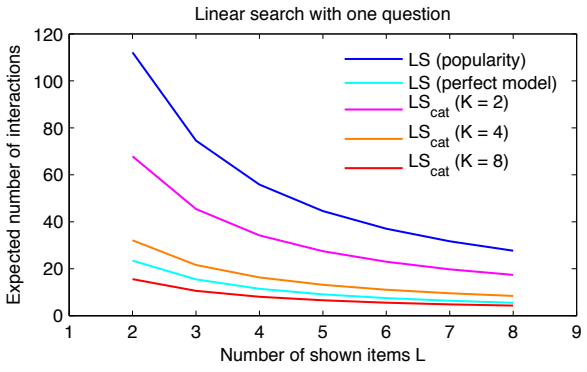

Figure 3. The expected length of $\texttt{LS}_{\texttt{cat}}$ interaction.

We evaluate $\texttt{LS}_{\texttt{cat}}$ using the same methodology as in the earlier evaluation of the LS policies. We simulate $K = 2$, $K = 4$, and $K = 8$ item categories being shown to the user, and compare $\texttt{LS}_{\texttt{cat}}$ with these values of K to the *popularity* and *perfect model* policies of LS. The results of the simulation are reported in Figure 3.

We observe that splitting items into K categories and then performing LS substantially decreases the length of user interaction. A split into $K = 2$ categories decreases the length of interaction by about 30%, consistently for any number of shown items L. Notably, for $K = 8$, the expected length of interaction by $\texttt{LS}_{\texttt{cat}}$ is on par with the *perfect model* LS policy. In other words, we show that a single piece of additional information provided by the user (choosing one item category out of K) reduces the expected length of interaction as much as knowing the exact ratings for the set of items that are recommended to the user.

MINIMAL INTERACTION SEARCH

In this section, we present the generalized search and our main contribution, *generalized linear search (GLS)*.

Generalized Search

Generalized binary search (GBS) [10, 12, 18] is a greedy algorithm for active learning, where the goal is to identify an item of interest e^* by asking a sequence of binary questions. A question-answering policy that minimizes the expected number of asked questions is NP-hard to compute, and GBS is its computationally efficient approximation. In particular, GBS always asks a question that partitions the *version space* V, the set of the items that are consistent with the past questions and answers, most evenly in expectation. A remarkable property of GBS is its theoretical guarantee to ask, in expectation, at most $\log(1/\pi_{\min})$ times more questions than the optimal policy, where $\pi_{\min} = \min_{e \in E} \pi(e)$ is the probability of choosing the least likely item.

Algorithm 3 GS: Generalized search.

Input:
Number of item categories K in a question

$V \leftarrow E$
repeat
Choose $K-1$ categories $q_1, \ldots, q_{K-1}$ given V and π
$q_K \leftarrow \overline{(q_1 \cup \ldots \cup q_{K-1})}$
Ask the user to choose one category
The user chooses category q_k
$V \leftarrow q_k \cap V$
until $(|V| = 1)$

Output: Target item $e^* \in V$

In this work, we consider a generalization of GBS to multi-way questions [5], which we refer to as *generalized search (GS)*. The search for the target item e^* proceeds as follows. The user is presented K item categories $q_1, \ldots, q_K$ and asked to choose one. After the user chooses a category q_k, the search space V is reduced to the items that belong to that category, $q_k \cap V$. Then the user is presented another K item categories. These categories may, and typically do, depend on the previously chosen categories. The user chooses again one category and this question-answering game continues until the cardinality of the version space V is one, and the target item e^* is found. Each round of this game is considered as an interaction. The pseudocode of GS is shown in Algorithm 3.

GS can search high-dimensional spaces efficiently when the item categories permit good partitioning of the version space. This is not always possible in practice. For instance, consider $N = 100$ items, where the probability of choosing the most likely item is $\pi(1) = 0.99$ and the item cannot be separated from the rest of the items by a single question. In this case, LS can identify the target item in a single interaction with at least 0.99 probability, because in 99% of cases the highest ranked item by $\pi(e)$ is the target item. On the other hand, GS needs to ask multiple questions to eliminate other items with at least 0.99 probability.

In previous section, we showed that a single split into K item categories in $\texttt{LS}_{\texttt{cat}}$ gains as much information about the target item e^* as knowing the exact item ratings. A natural question to ask is "what happens if the user is asked to choose a category of interest more than once". Will the expected length of interaction further decrease? To answer this question, we simulate a modified variant of GS with $K = 2$ item categories. Specifically, we fix the number of GS questions and switch to LS either after this number of questions or when the cardinality of the version space is one. The methodology for choosing users and items is identical to that in our earlier simulations.

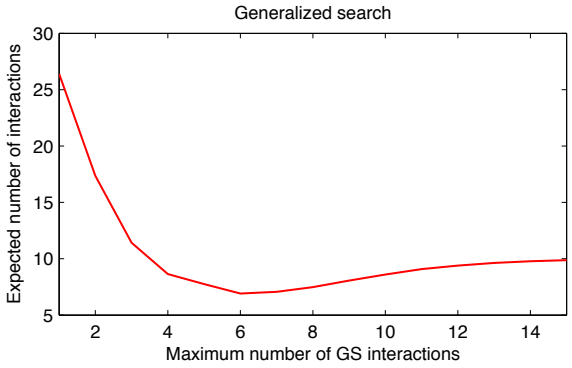

Figure 4. The expected length of interaction as a function of the number of GS steps.

The overall length of interaction is the sum of the number of interactions in GS and LS. The results of the simulation are reported in Figure 4.

Interestingly, the expected length of interaction does not always decrease with the number of asked GS questions. In fact, it starts increasing when GS asks more than 6 questions. For instance, the expected length of interaction at 9 questions is larger by one than at 6 questions. The reason for this behavior is that asking questions is beneficial only if the version space can be partitioned sufficiently well into K item categories. Loosely speaking, if this cannot be done, it is preferable to order items according to their popularity, from the most popular to the least popular, and ask the user to discover the target item by scanning the items.

Generalized Linear Search

This observation motivates our work on the hybridization of GS and LS that leverages the strengths of both approaches to minimize the length of user interaction. Specifically, we propose to perform GS up to the point where it is not effective, because the version space V cannot be partitioned well, and then switch to LS, which does not rely on the features of items. A pivotal question in this hybridization is "how to choose the sweet spot where GS should switch to LS". For instance, one suitable switching criterion may be when the number of items in the version space V is small. Another suitable criterion may be when the probability mass in V is small. In other words, there are many ways to combine the two methods and it is not obvious which one to apply.

In this work, we propose a new hybrid method that we call *generalized linear search* (GLS). GLS switches from GS to LS when the *expected length of interaction* to discover the target item using GS is greater than that of LS. We choose this switching oracle for several reasons. First, this criterion naturally captures the fact that GS outperforms LS in minimizing the expected length of interaction. Second, the oracle can be implemented computationally efficiently. Third, we can incorporate the oracle in the analysis of GLS and show that the expected length of interaction decreases. Finally, as we demonstrate in the experimental evaluation, this approach performs well in practice.

Algorithm 4 GLS: Generalized linear search.

Input:
Number of item categories K in a question
Number of shown items L

$V \leftarrow E$
repeat
Choose $K-1$ categories $q_1, \ldots, q_{K-1}$ given V and π
$q_K \leftarrow \overline{(q_1 \cup \ldots \cup q_{K-1})}$
Ask the user to choose one category
The user chooses category k
$V \leftarrow q_k \cap V$
until $(\texttt{costGS}(V, K) \geq \texttt{costLS}(V, L))$
$e^* \leftarrow \texttt{LS}(V, L)$

Output: Target item e^*

Algorithm 5 costLS: Expected cost of linear search.

Input:
Items $V \subseteq E$
Number of shown items L

Let $e_1, \ldots, e_{|V|}$ be an ordering of items V such that:
$\pi(e_1) \geq \ldots \geq \pi(e_{|V|})$
$c \leftarrow 0$
for all $i = 1, \ldots, |V|$ **do**
$c \leftarrow c + \lceil i/L \rceil \, \pi(e_i)$
end for

Output: Expected cost c

The pseudocode of GLS is in Algorithm 4. Note that GLS differs from GS in two aspects. First, GS stops when the expected cost of GS is higher than that of LS. Second, when GS stops, we switch to LS. The pseudocodes for computing the expected costs of LS and GS are in Algorithms 5 and 6, respectively.

GLS can be implemented efficiently. In particular, note that GS can be represented as a decision tree [12], where each node is a version space associated with K item categories. Similarly, GLS can be viewed a pruned version of this tree, where each leaf node is an instance of LS. The pruned tree is built as follows. First, we build the tree for GS and recursively compute the expected cost of GS in each node. Second, we compute the expected cost of LS in each node and prune the tree at the nodes where $\texttt{costGS}(V, K) \geq \texttt{costLS}(V, L)$. The expected cost of LS in each node can be computed in $O(N \log N)$ time, because it requires sorting at most N items.

Analysis

Let π be the distribution over items E and $\pi(e)$ be the probability that item e is chosen. Let $V \subseteq E$ be a version space and $\mathcal{V}$ be the set of all version spaces where GLS switches from GS to LS, $\texttt{costGS}(V, K) \geq \texttt{costLS}(V, L)$. Then $\mathcal{V}$ is a partitioning of E:

$$E = \bigcup_{V \in \mathcal{V}} V \qquad \forall V \in \mathcal{V}, U \in \mathcal{V} \setminus V : V \cap U = \emptyset$$

Algorithm 6 `costGS`: Expected cost of generalized search.

Input:
Items $V \subseteq E$
Number of item categories K in a question

$c \leftarrow 1$
if $(|V| > 1)$ **then**
Choose $K-1$ categories $q_1, \ldots, q_{K-1}$ given V and π
$q_K \leftarrow \overline{(q_1 \cup \ldots \cup q_{K-1})}$
for all $k = 1, \ldots, K$ **do**
$c \leftarrow c + \pi(q_k \cap V) \times \texttt{costGS}(q_k \cap V, K)$
end for
end if

Output: Expected cost c

because GLS is guaranteed to switch to LS, at the latest when the cardinality of the version space is one. Let $N(e)$ be the length of interaction to find item e by GLS and $N(V)$ be the length of interaction until GLS reaches V. Let $\pi(V) = \sum_{e \in V} \pi(e)$ be the probability mass of all items in V.

THEOREM 1. *The expected cost of* GLS *is smaller or equal to the expected cost of* GS.

PROOF. Note that the expected cost of GS is defined as

$$\mathbb{E}_{e \sim \pi}[N(e)] = \sum_{e \in E} \pi(e) N(e). \quad (1)$$

For any target item e^*, Algorithm 4 switches from GS to LS at one particular point, where $e^* \in V$ for some $V \in \mathcal{V}$. Based on this, the expected cost can be written as:

$$\sum_{V \in \mathcal{V}} \left[\pi(V) N(V) + \sum_{e \in V} \pi(e)(N(e) - N(V)) \right]. \quad (2)$$

The first term is the expected cost of GS before $V \in \mathcal{V}$ is reached. The second term is the expected cost of GS after V is reached. Equation 2 can be further rewritten as:

$$\sum_{V \in \mathcal{V}} \pi(V) \left[N(V) + \sum_{e \in V} \frac{\pi(e)}{\pi(V)} (N(e) - N(V)) \right], \quad (3)$$

where $\sum_{e \in V} \frac{\pi(e)}{\pi(V)}(N(e) - N(V))$ is the expected cost of GS applied in V. Let $N_{\text{LS}}(V)$ be the expected cost of LS from V. By our assumption:

$$N_{\text{LS}}(V) \leq \sum_{e \in V} \frac{\pi(e)}{\pi(V)} (N(e) - N(V)) \quad (4)$$

because $\texttt{costGS}(V, K) \geq \texttt{costLS}(V, L)$ for all $V \in \mathcal{V}$. Finally, we chain all inequalities and get:

$$\mathbb{E}_{e \sim \pi}[N(e)] \geq \sum_{V \in \mathcal{V}} \pi(V) \left[N(V) + N_{\text{LS}}(V) \right], \quad (5)$$

where the right-hand side is the expected cost of GLS. This concludes the proof. ■

Non-Unit Costs of Interaction

So far we assumed that each type of interaction bears the same cost. Our work can be easily generalized to the setting where the costs of GS and LS interactions differ. Without loss of generality, let the cost of GS interactions be one and the cost of LS interaction be α. Note that when $\alpha > 1$, LS interactions are more costly than GS interactions, and vice versa. Then GLS can be straightforwardly adapted to minimize the weighted length of interactions, by replacing c with αc in costLS (Algorithm 5). Similarly, it is straightforward to adapt the proof of Theorem 1, by replacing $N_{\text{LS}}(V)$ with $\alpha N_{\text{LS}}(V)$.

OFFLINE EVALUATION

We conduct offline evaluation of the proposed GLS method using two public recommender systems datasets and compare it to several baselines and heuristics.

Experimental Setting

We compare GLS to five baselines. The first two baselines are standalone LS and GS, which were presented in previous sections. The other three baselines are heuristic combinations of GS and LS, which we call GS-step, GS-num, and GS-ent. These baselines differ in how they switch from GS to LS. GS-step switches to LS after Δ iterations of GS. GS-num switches to LS when the version space contains less than Δ items. GS-ent switches to LS when the entropy of the probability mass of the items in the version space is smaller than Δ. Intuitively, the heuristics address the pivotal question of when to switch to LS: when the version space has been refined enough times in GS-step, when the version space is sufficiently small in GS-num, and when the probability mass is concentrated in just a few items in GS-ent. In all three cases, Δ is the external parameter for the heuristic method.

The performance of all methods is measured by the expected length of interaction. This quantity is computed as follows. First, we randomly choose a user, proportionally to the number of items that the user consumed in the dataset. Second, for a given user, we randomly choose a target item e^*, proportionally to the number of times that this item is consumed by the user. Finally, we compute the length of interaction to find e^*. The length of interaction is averaged over 10^6 randomly chosen pairs of user and item. Therefore, all error bars in the experiments are on the order of $1/\sqrt{10^6} = 10^{-3}$, and most differences in the expected lengths of interaction are statistically significant. We experiment with three different numbers of item categories that are shown: $K = 2$, $K = 4$, and $K = 8$. In the all experiments, the number of shown items in LS is $L = 8$. We observed similar trends for other values of L and do not report these due to space constraints.

The proposed methods are evaluated on two datasets: *MovieLens*[1] and *Lastfm*[2]. The *MovieLens* dataset contains the ratings of 6K users for 3.9K movies. We preprocess this dataset as follows. The items E are movies and $\pi(e)$ is the probability that movie e is watched. This probability is estimated as the number of times that e is rated with 4 or 5 stars divided by

[1]MovieLens dataset: http://www.grouplens.org/node/73
[2]Last.fm dataset: http://www.lastfm.com

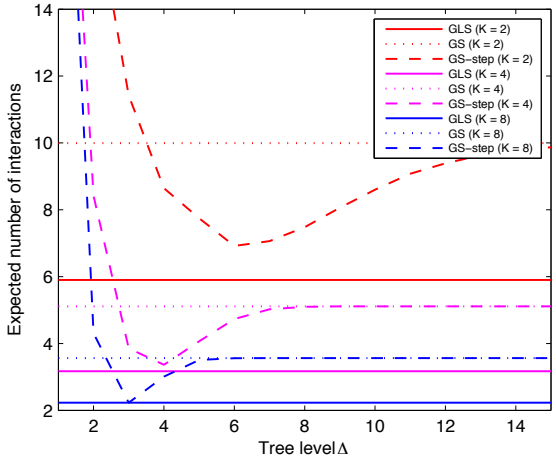

Figure 5. Comparison of `GLS` to `GS` and `GS-step` for *MovieLens*.

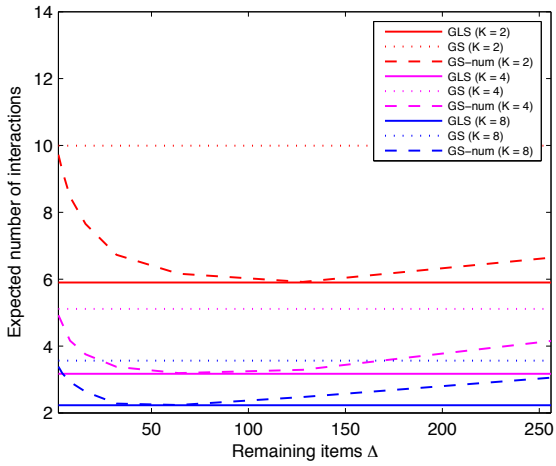

Figure 6. Comparison of `GLS` to `GS` and `GS-num` for *MovieLens*.

the total number of 4 and 5 star ratings. The item categories q are the genres of movies. *MovieLens* contains 18 movie genres, which is insufficient for our purpose. Therefore, we replaced the genres of each movie by its Netflix genres. There are 335 distinct Netflix genres in our dataset. We eliminate the movies that are described by less than 5 genres. These are unsuitable for `GS`-style methods because they are not described by a sufficient number of features. We end up with a dataset of 1.4K movies.

Lastfm is a music dataset of 2K users who listen to 18K artists. We preprocess this dataset as follows. The items E are artists and $\pi(e)$ is the probability that artist e is listened to. This probability is estimated as the number of times that artist e is listened to divided by the total number of the artist listening events. The item categories q are the tags assigned to the artists. We eliminate the artists that are described by less than 5 tags. Again, these are unsuitable for `GS` because they are not described by a sufficient number of features. We end up with a dataset of 6K artists.

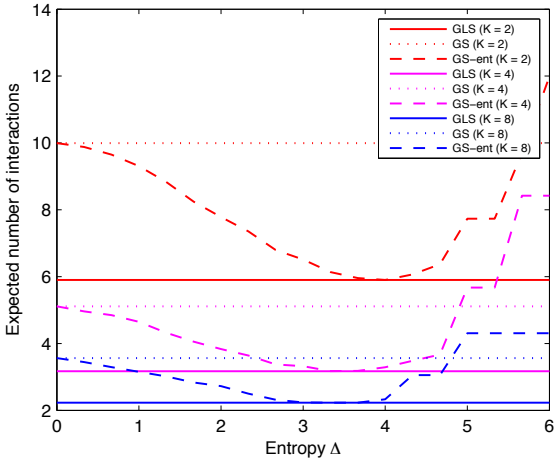

Figure 7. Comparison of `GLS` to `GS` and `GS-ent` for *MovieLens*.

Experimental Results

In Figure 5, we compare `GLS` to `GS` and heuristic `GS-step` on the *MovieLens* dataset, for three values of K. We observe two major trends. First, the length of interaction decreases with K, as larger values of K allow for a fine-grained partitioning of the version space. Second, for all values of K, `GLS` significantly outperforms `GS`. For example, for $K = 2$, the expected length of `GLS` interaction is 5.93, 40% shorter than that of `GS`, 9.85. For $K = 4$ and $K = 8$, the expected length of `GLS` interaction is, respectively, 36% and 34% shorter than that of `GS`. Third, `GLS` always performs better than or on par with `GS-step`. Fourth, we observe that the performance of `GS-step` depends on the parameter Δ. For instance, for $K = 2$, `GS-step`(6) performs the best, while `GS-step`(3) is already worse than `GS`. We observe the same trend for other values of K. For instance, `GS-step`(3) performs comparably to `GLS` for $K = 8$, but worse than `GS` for $K = 2$.

In Figure 6, we compare `GLS` to `GS` and `GS-num` on the *MovieLens* dataset. We observe the same trends as in Figure 5. First, the length of interaction is inversely correlated with K. Second, `GLS` always outperforms `GS`. Third, `GLS` always outperforms or is on par with `GS-step`. Fourth, the performance of `GS-num` depends on the value of Δ: for $K = 2$, the shortest `GS-num` interaction is obtained for $\Delta = 130$; while for $K = 4$ and $K = 8$, it is obtained for $\Delta = 70$ and $\Delta = 50$, respectively. Also note that for all values of K, at the optimal operating point `GS-num` performs on par with `GLS`, whereas for `GS-step` this was observed only for $K = 8$. This is attributed to the fact that the number of items in the version space is a finer indicator of the suboptimal performance of `GS` than the number of `GS` steps.

In Figure 7, `GLS` is compared to `GS` and heuristic `GS-ent` on the *MovieLens* dataset, for the same three values of K. We observe the same trends that were observed in Figures 5 and 6, although in this case the performance of `GS-ent` does not vary with K as widely as previously.

We conduct the same experiments on the *Lastfm* dataset, and our results are reported in Figures 8, 9, and 10. Our findings

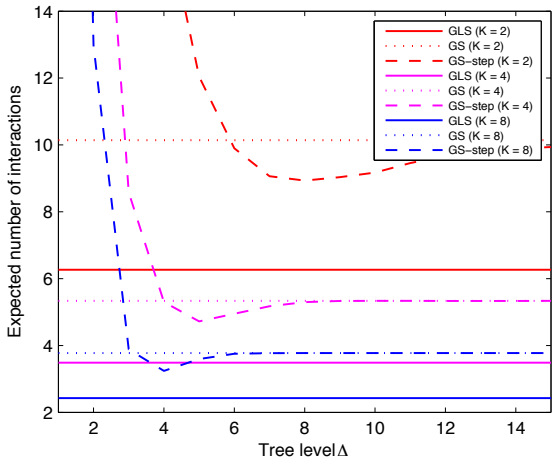

Figure 8. Comparison of `GLS` to `GS` and `GS-step` for *Lastfm*.

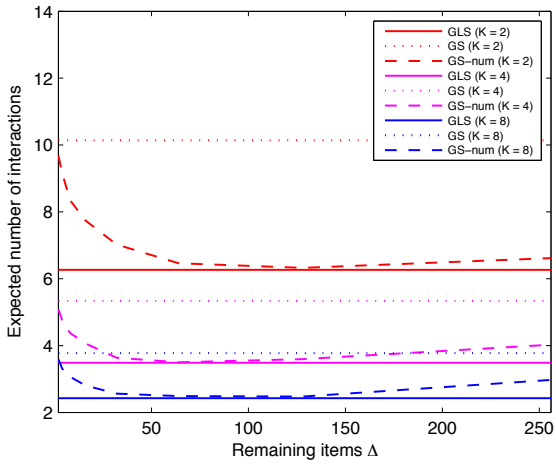

Figure 9. Comparison of `GLS` to `GS` and `GS-num` for *Lastfm*.

resemble those in the *MovieLens* dataset. The expected length of `GLS` interaction is always shorter than that of `GS`, and `GLS` is either superior or comparable to the three heuristic methods, whose performance depends on their parameter Δ. Also note that the optimal operating points of the heuristics are obtained for different values of Δ. For example, on the *MovieLens* dataset, `GS-step` performs the best for $\Delta = 3$, $\Delta = 4$, and $\Delta = 6$; when $K = 2$, $K = 4$, and $K = 8$, respectively. On the other hand, on the *Lastfm* dataset, `GS-step` performs the best for $\Delta = 4$, $\Delta = 5$, and $\Delta = 8$ for the same values of K. Similar differences across the datasets are observed also for the `GS-num` and `GS-ent` heuristics.

We conclude that `GLS` performs better than its individual components, `GS` and `LS`. This superiority is statistically significant and holds for the *MovieLens* and *Lastfm* datasets under evaluation, and for various values of K. A more intricate question deals with the comparison of `GLS` to the heuristic methods. We conclude that `GLS` steadily outperforms `GS-step`, and performs comparably with `GS-num` and `GS-ent`, but only when these two are parameterized optimally. However, the optimal

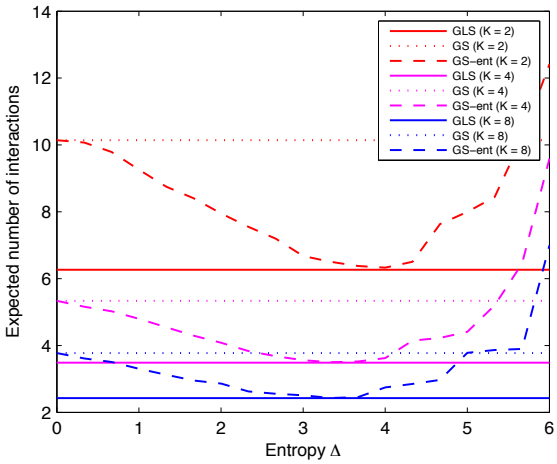

Figure 10. Comparison of `GLS` to `GS` and `GS-ent` for *Lastfm*.

operating points Δ of the heuristics depend on the applied heuristics, the dataset in hand, and the number of categories K. In practice, a suitable value of Δ has to be chosen a priori and this is not trivial. On the other hand, `GLS` is parameter free and never performs worse than `GS-step`, `GS-num`, and `GS-ent`. Hence, `GLS` should be preferred not only to `GS` and `LS`, but also to the three evaluated heuristic methods.

LIVE USER STUDY

We conduct a user study of `GLS` on Amazon's Mechanical Turk[3], a public crowd-sourcing service. For the sake of simplicity, we only compare the binary variants of `GLS` and `GS`, with $K = 2$ and $L = 10$. The study is intra-group: each subject experiences both searches and is asked to compare them.

Experimental Setting

We experiment with 253 subjects. Each subject interacts with `GLS` and `GS` in a random order, and is asked to compare the searches. The recommended items E are 200 most rated movies from the *MovieLens* dataset.

The experiment is organized as follows. At the beginning of the experiment, we randomly select 10 movies from E, present them to the subject, and ask the subject to choose a target movie that the subject is familiar with (top of Figure 11). Upon selecting the target movie, the subject interacts with the first search. This search is chosen at random to eliminate position bias. In the `GLS` search, the subject is asked to answer binary `GS` questions (middle of Figure 11) until `GLS` switches to `LS`. When this happens, the subject is shown a list of recommended movies, requested to examine it, and asked if the target item is in the list (bottom of Figure 11). Likewise, in the `GS` search, the subject answers binary `GS` questions until the search space is sufficiently small, and then is shown a list of movies and asked whether the target movie is in the list. Upon completing both searches, the subject is asked to answer four questions that compare the searches (Table 1).

[3] `http://mturk.com`

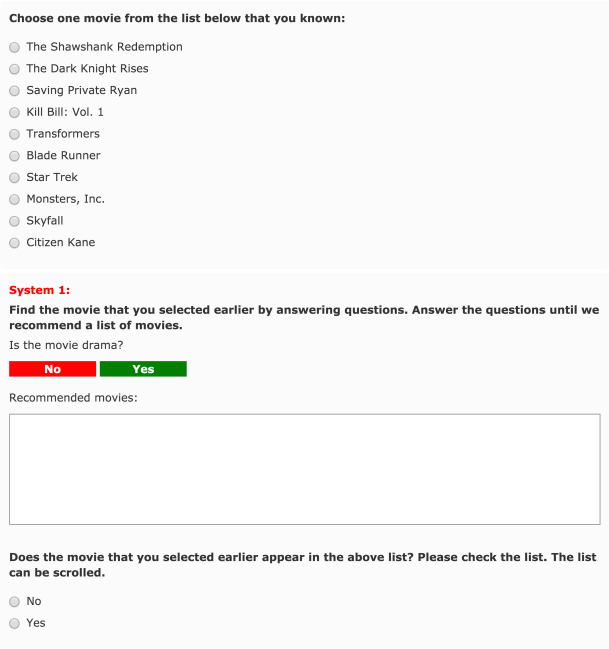

Figure 11. A portion of the user study.

For each search, we compute four metrics. The first is the *hit rate*, i.e., the fraction of searches in which the target movie appears in the final list, which communicates how good the search is in finding the target movie. The second metric is the *number of questions* that the subject was asked during the search. The third metric is the *duration of interaction*, i.e., the time (in seconds) between answering the first question and examining the final list. The fourth metric is the *length of the final list* of recommended movies.

Experimental Results

In Table 2, we report the values of the four metrics for each search as well as the distribution of the answers to the comparison questions from Table 1. We observe several trends. First and foremost, GLS is preferred over GS by more subjects, 61.3% vs 38.7%. What are the reasons for this preference?

Table 2 clearly shows that GLS requires significantly less interaction than GS, as reflected by both the number of asked questions and the duration of the search. This is noted by the subjects. Specifically, 46.7% of the subjects prefer GLS because it asks less questions (C1). Most of these subjects also find GLS easier to use (C3) and prefer it overall (C4).

Also, Table 2 shows that GLS produces much longer lists of movies than GS. Although the examination of these lists does not overload the subjects, as indicated by the duration of the search, 38.3% of the subjects prefer shorter lists that are produced by GS (C2), and most of these subjects also prefer GS overall. Nevertheless, the benefits of C1 and C3 outweigh the shortcomings of C2, as more subjects prefer GLS over GS.

We also observe that a large portion of the subjects has no clear preference towards the three criteria: 39.5% (C1), 43.9% (C2), and 55.7% (C3). This is not surprising, since both searches are similar and differ only in the last few steps,

Which system do you prefer with respect to the following:

(C1) Number of asked questions
- No preference
- I prefer System 1 because it asks less questions
- I prefer System 2 because it asks less questions

(C2) Length of the final list
- No preference
- I prefer System 1 because it produces a shorter list
- I prefer System 2 because it produces a shorter list

(C3) Ease of use
- No preference
- I prefer System 1 because it is easier to use
- I prefer System 2 because it is easier to use

(C4) Overall
- I prefer System 1 to System 2
- I prefer System 2 to System 1

Table 1. Search comparison questions.

	GS	GLS	Neither
Hit rate	0.32	0.53	
Number of questions	6.12	2.30	
Search duration	9.69	3.52	
Length of the final list	3.74	29.41	
(C1) Number of questions	13.8%	46.7%	39.5%
(C2) Length of the final list	38.3%	17.8%	43.9%
(C3) Ease of use	13.8%	30.5%	55.7%
(C4) Overall preference	38.7%	61.3%	

Table 2. Summary of the user study results.

when GLS terminates before GS and produces a longer list of movies. Finally, note that GLS achieves a substantially higher hit rate than GS. This is due to the fact that GLS asks less questions than GS and is less likely to eliminate the target movie due the incorrect answer of a subject to a question.

RELATED WORK

Recommendations produced by a recommender system are typically shown as a ranked list of items. The ranking of items in the list communicates their fit to the target user, such as the top-ranked items are supposedly the best recommendations [17]. Limiting the size of the list reduces the choice difficulty and helps the user to cope with information overloading [6].

Several issues around recommendation interfaces have been investigated in early recommendation works. Cosley *et al.* [9] studied how the display and the scale of item predictions affect the ratings provided by users. It was found that users were influenced by the displayed predictions, while the scale had no apparent effect. Ziegler *et al.* [28] argued that the recommendation list should incorporate a degree of diversity, to reflect the breadth of user interests [1, 2]. It was shown that, despite decreasing the accuracy of the recommendations, diversification increased the overall user satisfaction.

Often, the recommendation list can be grouped or categorized, to simplify user navigation and item exploration. Hu and Pu [15] evaluated the "organizational" recommendation interface that allowed users to group and filter items according to a set of domain features. A comparison between the organizational and list interfaces indicated that the former improved the perceived usefulness and diversity of the recommendations, increased user confidence, and was found to be more intuitive than the latter.

In addition to the LS and GS approaches that were elaborately discussed earlier, another related search is *faceted search* [14]. This is a search, where the user first chooses a facet, a group of semantically related attributes, and then a desired attribute in that facet. In our work, we have only one group of semantic attributes, either movie or music genres. That is, our approach can be viewed as a degenerate case of faceted search with only one group, and we do not perform experimental comparison work to faceted search.

The importance of an easy-to-use interface is paramount in critique-based recommendations [8]. Chen and Pu developed an interface that grouped items, compared the groups to the recommended items, and suggested critiques reflecting these comparisons [7]. The accuracy of the critiques and the usability of the interface were evaluated in a user study. The results showed that users preferred the suggested critiques and provided them more frequently, which reduced the duration of interactions and increased the perceived user confidence.

Another functionality availed by the recommender's interface is explanation and persuasion [4]. These are used to justify the recommendations, cultivate user trust, and convince users to follow the recommendations. Tintarev and Masthoff [22] developed several interfaces complementing the recommended items with explanations. Their evaluation confirmed that explanations boost user satisfaction, whereas the highest satisfaction was achieved when the explanations referred to item features that were important for users.

Although much research has been devoted to exploration interfaces in the general HCI context [3], this has received relatively little attention in recommender systems [21, 23]. In a recent work, Verbert *et al.* [24] developed TalkExplorer, an interactive academic conference visualization tool, building upon graph-based information exploration libraries, like PeerChoser [19] and SmallWorlds [13]. TalkExplorer allowed users to explore various relevance perspectives, e.g., tags assigned to recommended papers, users who bookmarked these papers, and other papers tagged by these users. The evaluation indicated that TalkExplorer was perceived more insightful that the standard list of recommended items.

The above interface components and visualization tools supply various information complementing the recommendations. However, none of them looks into optimizing user experience in recommender systems [16]. This issue is addressed by general user interaction design guidelines, and we observe successful practical solutions in industrial products, such as movie recommendation interface by Netflix[4] or contact recommendations by LinkedIn[5]. However, to the best of our knowledge, no evidence-based research that articulates the development of these interfaces, in particular, considering the recommendation scenarios, has been published.

[4]Netflix recommendations: Beyond the 5 stars, http://techblog. netflix. com/2012/04/netflixrecommendations-beyond-5-stars.html

CONCLUSIONS

The claim that a usable interface may improve user's subjective perception of a recommender is well established [20]. In spite of this, recommender systems research has mainly focused on algorithmic techniques for rating predictions, rather than on interface and user interaction matters.

In this work, we devised the GLS method that minimizes the length of user interaction with the recommender, to find a target item in a recommendation list. GLS combines two established search methods, GS and LS, and successfully leverages their advantages. We investigated the switching criterion from GS to LS and concluded that GS should be applied as long as its expected cost is lower than that of LS, and should be abandoned afterwards. We prove that GLS performs at least as well as GS.

We demonstrate in an offline and live evaluation that the proposed GLS method is superior not only to its individual components, GS and LS, but also to several heuristic methods. Having said that, GLS does not require any parameter tuning and can be deployed in a practical recommender system. We also conduct a user study that demonstrates steady user preference towards GLS over the baseline GS method.

Future questions for investigation refer to a thorough user evaluation of GLS. While it was found to lead to the shortest interactions and was preferred by users, it combines item categories with individual items in the same search process. Hence, it is critical to ascertain whether users find the GLS interactions intuitive and enjoyable [16]. Also, it is not obvious how GLS, and specifically the item categories in the GS stages, should be visualized [27]. On one hand, textual titles may not be as appealing as pictorial thumbnails. On the other hand, showing the thumbnails may not properly communicate the very notion of categories.

Apart from this, it is important to address the cost of answering questions. In our user study, we show that GLS recommends a much longer list of items than GS. This may put a significant burden on users [6], which may be comparable with answering a few GS questions, and practically factor out the advantages of GLS. Therefore, it is important to thoroughly study the cost, both in terms of time and cognitive effort, of answering different types of questions.

Finally, another aspect that asks for further research is the integration of GLS with a variety of interface and user-related topics, such as decision support, provision of critiques, content discovery, and explanation of recommendations [11]. It is, therefore, necessary to conduct user studies and evaluate the compound effect of GLS in these use cases.

[5]Navigating LinkedIn's New User Interface, http://randomactsofleadership.com/navigating-linkedins-new-user-interface/

REFERENCES

1. Ashkan, A., Kveton, B., Berkovsky, S., and Wen, Z. Diversified utility maximization for recommendations. In *RecSys Posters* (2014).
2. Ashkan, A., Kveton, B., Berkovsky, S., and Wen, Z. DUM: Diversity-weighted utility maximization for recommendations. *CoRR abs/1411.3650* (2014).
3. Baldonado, M. Q. W., and Winograd, T. Sensemaker: An information-exploration interface supporting the contextual evolution of a user's interests. In *CHI* (1997), 11–18.
4. Berkovsky, S., Freyne, J., and Oinas-Kukkonen, H. Influencing individually: Fusing personalization and persuasion. *ACM Trans. Interact. Intell. Syst. 2*, 2 (2012), 9.
5. Bhamidipati, S., Kveton, B., and Muthukrishnan, S. Minimal interaction search: Multi-way search with item categories. In *ITWP* (2013), 9–15.
6. Bollen, D. G. F. M., Knijnenburg, B. P., Willemsen, M. C., and Graus, M. P. Understanding choice overload in recommender systems. In *RecSys* (2010), 63–70.
7. Chen, L., and Pu, P. Experiments on the preference-based organization interface in recommender systems. *ACM Trans. Comput.-Hum. Interact. 17*, 1 (2010), 5.
8. Chen, L., and Pu, P. Critiquing-based recommenders: survey and emerging trends. *User Model. User-Adapt. Interact. 22*, 1-2 (2012), 125–150.
9. Cosley, D., Lam, S. K., Albert, I., Konstan, J. A., and Riedl, J. Is seeing believing?: how recommender system interfaces affect users' opinions. In *CHI* (2003), 585–592.
10. Dasgupta, S. Analysis of a greedy active learning strategy. In *Advances in Neural Information Processing Systems 17* (2005), 337–344.
11. Felfernig, A., Burke, R. D., and Pu, P. Preface to the special issue on user interfaces for recommender systems. *User Model. User-Adapt. Interact. 22*, 4-5 (2012), 313–316.
12. Golovin, D., and Krause, A. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artif. Intell. Research 42* (2011), 427–486.
13. Gretarsson, B., O'Donovan, J., Bostandjiev, S., Hall, C., and Höllerer, T. Smallworlds: Visualizing social recommendations. *Comput. Graph. Forum 29*, 3 (2010), 833–842.
14. Hearst, M. Design recommendations for hierarchical faceted search interfaces. In *SIGIR Workshop on Faceted Search* (2006).
15. Hu, R., and Pu, P. Enhancing recommendation diversity with organization interfaces. In *IUI* (2011), 347–350.
16. Knijnenburg, B. P., Willemsen, M. C., Gantner, Z., Soncu, H., and Newell, C. Explaining the user experience of recommender systems. *User Model. User-Adapt. Interact. 22*, 4 (2012), 441–504.
17. McNee, S. M., Riedl, J., and Konstan, J. A. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI* (2006), 1097–1101.
18. Nowak, R. The geometry of generalized binary search. *IEEE Transactions on Information Theory 57*, 12 (2011), 7893–7906.
19. O'Donovan, J., Smyth, B., Gretarsson, B., Bostandjiev, S., and Höllerer, T. Peerchooser: visual interactive recommendation. In *CHI* (2008), 1085–1088.
20. Ricci, F., Rokach, L., and Shapira, B. Introduction to recommender systems handbook. In *Recommender Systems Handbook*. Springer, 2011, 1–35.
21. Tintarev, N., Hu, R., and Pu, P. Recsys'12 workshop on interfaces for recommender systems. In *RecSys* (2012), 355–356.
22. Tintarev, N., and Masthoff, J. Evaluating the effectiveness of explanations for recommender systems - methodological issues and empirical studies on the impact of personalization. *User Model. User-Adapt. Interact. 22*, 4-5 (2012), 399–439.
23. Tintarev, N., O'Donovan, J., Brusilovsky, P., Felfernig, A., Semeraro, G., and Lops, P. Recsys'14 workshop on interfaces and human decision making for recommender systems. In *RecSys* (2014), 383–384.
24. Verbert, K., Parra, D., Brusilovsky, P., and Duval, E. Visualizing recommendations to support exploration, transparency and controllability. In *IUI* (2013), 351–362.
25. Wen, Z., Kveton, B., Eriksson, B., and Bhamidipati, S. Sequential Bayesian search. In *ICML* (2013), 226–234.
26. Winterboer, A., Tietze, M., Wolters, M., and Moore, J. The user model-based summarize and refine approach improves information presentation in spoken dialog systems. *Comput. Speech and Lang. 25*, 2 (2011), 175–191.
27. Zhang, J., Jones, N., and Pu, P. A visual interface for critiquing-based recommender systems. In *EC* (2008), 230–239.
28. Ziegler, C.-N., McNee, S. M., Konstan, J. A., and Lausen, G. Improving recommendation lists through topic diversification. In *WWW* (2005), 22–32.