COD: Iterative Utility Elicitation for Diversified Composite Recommendations

Khalid Alodhaibi
George Mason University
4400 University Drive 4A4
Fairfax, VA 22030
kalodhai@gmu.edu

Alexander Brodsky
George Mason University
4400 University Drive 4A4
Fairfax, VA 22030
brodsky@gmu.edu

George A. Mihaila
IBM T.J. Watson Res. Ctr.
19 Skyline Drive
Hawthorne, NY, USA
mihaila@us.ibm.com

Abstract

This paper studies and proposes methods for providing recommendations on composite bundles of products and services that are dynamically defined using database views extended with decision optimization based on mathematical programming. A framework is proposed for finding a diverse recommendation set when no prior knowledge on user preference is given. To support this framework, a method is developed for utility function elicitation, which is based on iteratively refining a set of axes in the n-dimensional utility space. The notion of a diverse recommendation set is refined and formalized by partitioning the recommendation space into layers that correspond to their distance to the maximal utility. In each layer, the method selects recommendations that maximize each dimension of the utility space. A preliminary experimental study is conducted, which shows that the proposed framework significantly outperforms a popular commercial system in terms of precision and recall.

1 Introduction

Recommender systems are increasingly used to help with selection of diverse products and services over the Internet. This paper focuses on recommending composite services and products and eliciting user preferences. Most of today’s recommender systems recommend only atomic products or services. Complex recommendation models involving composite alternatives, such as product configurations and service packages, are rarely addressed. In addition, the majority of recommender systems rely on a single ranking or utility score, whereas, in many applications, there are multiple criteria that need to be taken into account, such as price, quality and enjoyment.

Recently, multi-criteria ranking has been explored in recommendation set retrieval [2,15]. These methods choose a set of alternatives based on a distance measure calculated for each of the multiple criteria. Multi-criteria ranking can help provide a balance between diversity and optimality. However, most recommender systems limit recommendations to those that are relevant to users requests. Therefore, their recommendations are often similar to each other and do not provide enough diversity. Diversity is important because it helps users become aware of choices they may not have thought of.

With the recent surge in collaborative similarity-based recommenders, such as Amazon.com, a number of multi-criteria ranking methods have been proposed. Of significant importance to this research is work suggesting the importance of diversity sensitive recommendation sets. The work presented in [2,12] details several algorithms for selecting diverse recommendation alternatives based on the similarity of individual attributes. The work done by Linden, et al [9] also suggests a diverse ranking algorithm. Zhang and Hurley [23] used a similar approach with respect to calculating diversity; however, their similarity measure of recommendations was based on a set rather than individual recommendations. For example, a recommendation with low similarity to the target might make it to the final list because the similarity score of the set it belongs to, is above a threshold.

Furthermore, most existing recommender systems are designed for a single target domain and do not provide a general framework for the development of recommender systems. Finally, many recommender systems are intrusive and require explicit and significant feedback from the user [1]. Feedback will continue to be a primary factor in the recommender system concept; however, the next generation of recommender systems might want to extract information from users implicitly. An example might be how long the user spends reading a specific document to infer how much the user liked the document, consequently, giving it a higher rating.

There are several approaches for eliciting utility func-

978-0-7695-3869-3/10 $26.00 © 2010 IEEE
tions, most of which aim for semi-automated learning of a decision makers utility function. One approach is iterative learning and refinement of the users utility function using a value of information approach [21]. Another approach is by eliciting the utility function from a database of observed behavioral patterns [16,17,18,19]. A third approach is by eliciting the utility function from a database of already elicited utility functions [20]. While few recommender systems provide for estimating and refining the preferences of the user [10], works such as [12] have exemplified the need for such techniques. However, none of these works, to the best of our knowledge, work on recommendations for composite product and services, which makes the recommendation space very large (or infinite, for continuous selected quantities), and implicitly defined.

The CARD Framework [4] supports composite product and service definitions, and recommendations are based on dynamically learned utility function and decision optimization. Composite services in CARD are characterized by a set of sub-services, which, in turn, can be composite or atomic. CARD uses a decision-guidance query language DG-SQL to define recommendation views, which specify multiple utility metrics, and the weighted utility function. However, the CARD framework has a number of limitations. First, it assumes the knowledge of the estimated utility function, whereas often this may not be available, but needs to be extracted from the user. Second, in some cases, a recommendation space may have different utility functions for different cluster of recommendation, which the CARD framework does not address. For example, a person may be considering different categories of vacation packages, such as family, romantic or business travel, and would apply different utilities for these categories. Furthermore, the diversity method of CARD has not been mathematically formalized or tested.

In this paper, we adopt the CARD framework and resolve the limitations outlined above. More specifically, the contributions of this paper are as follows. First, we propose a framework for finding a diverse recommendation set, when no prior knowledge on user preference is given. The framework involves interaction with the user to (1) choose a recommendation cluster the user is interested in, (2) dynamic elicitation of the weighted utility function, and then (3) generating a diverse set of recommendation that contains an optimal recommendation in terms of the estimated utility function. We refer to our framework as Clustering-Optimization-Diversity (COD) framework.

Second, we develop a method for utility function elicitation. It is based on an iteratively refining a set of axes in the n-dimensional utility space, starting from the utility space standard axes. At every step, the user is asked to rank a set of recommendations, each being optimal for one of the current axes. Based on the user feedback, the method refines the set of axes which become closer to each other, until the user cannot differentiate among them.

Third, we formalize the notion of a diverse recommendation set by defining the notion of m-layered recommendations. These recommendations contain one that optimizes the learned weighted utility function. Then, the space of all non-dominated recommendations (the “skyline”) is split into m layers, so that the first layer contains all recommendations whose utility is at least the maximum utility minus $\frac{1}{m}$ of the utility function range; the second up to $\frac{2}{m}$ and so on. Within each layer, a recommendation is chosen that optimizes one dimension of the utility space.

Fourth, we conducted a preliminary experimental study on the efficacy of the proposed framework, comparing precision and recall of ranked recommendations of a popular commercial travel site (called herein System A) vs. the COD framework using the same underlying set of flights and accommodations. The study showed that COD significantly outperformed System A. Furthermore, COD showed an average recall of 26% at rank 5 compared to 16% for System A, and an average precision of 100% at rank 1 compared to 36% for System A. While the preliminary study did not directly assess the level of diversity provided by COD, we conjecture that recall studied is partly reflective of recommendations diversity.

Many recommender systems are intrusive where they require explicit feedback from the user and often at a significant level of user involvement. For example, before recommending any movies, MovieLens.org expects the user to rate a predefined number of movies (e.g., 20). This request comes with costs on the end-user [1]. However, COD does not require any explicit feedback prior to using the system. Initially, COD would recommend a set of alternatives to choose from or provide feedback on. In addition, COD uses a simple feedback extraction mechanism, where users are only asked to place recommendations in a stratum which is typically quick and easy, furthermore, the number of recommendations to rank becomes smaller after the previous iteration. According to our user study, only 3 out of 30 people complained about the explicit feedback required.

As will be discussed in more details in section 6, the utility function elicitation of COD allows for less intrusive learning. Initially, COD is capable of recommending alternatives without the need to extract feedback. In addition, COD could work with previously calculated utilities. This utility can be obtained by domain

---

1One recommendation dominates another if it is at least as good as the other in all respects.
experts or calculated using historical data. Finally, the COD framework deals with the recommendation space of composite products and services by using decision optimization when extracting recommendations that optimize specific axis in the utility space. Such problems are more complex when there is a large number of alternatives (possibly infinite), however, we noticed that our framework converges after 2 questions as explained in the case study section.

This paper is organized as follows: Section 2 briefly reviews CARD framework [4], on which COD is based. Section 3 gives a high level description of the COD framework. Section 4 describes the Utility axis selection of COD. Section 5 describes the Diversity Layering of the COD framework. Section 6 presents a case study for the purpose of validating the framework. Section 7 presents the related work. Section 8 is the conclusion and avenues for future work.

2 A Review of the CARD Framework

The COD framework is based on the Composite Alternative Recommendation Development (CARD). In this section, we provide a review of CARD with material extracted from [4]. CARD is a framework that supports composite product and service, Top-k decision optimization, and Dynamic preference learning. Subservices can be atomic or composite. For example, travel packages are composed of many services including ground and air transportation, accommodations, and activities. Each atomic and composite service is associated with metrics, such as cost, duration, and enjoyment ranking.

We start with the clustering step to determine what cluster the user is interested in. Our recommendation space is split into a number of clusters, where each cluster contains a number of packages (recommendations). Examples of clusters are: honeymooners, single, family, etc. Basically, the clusters are extracted from historical purchase data cross-referenced with user demographic data. However, this is beyond the scope of our research at this point. Currently, we just assume that these clusters exist and packages can therefore be targeted to specific user groups. We then return a package to represent the corresponding cluster. Each recommendation returned is the highest total utility function in its cluster.

Initially, we will give equal weight to metric attributes as it is too early to conclude what the user might value more with respect to metrics attributes (e.g. Saving, Enjoyment). However, domain knowledge could also be used to determine how to assign weights and selections of metrics attributes, consequently calculating the global utility function. We then ask the user to indicate her preference, and the chosen recommendation determines the
preferred cluster. Thus, future recommendation space is limited to the chosen cluster.

We then start an iterative process of learning utility vector of metrics attributes, e.g., Saving, Location attractiveness, Enjoyment. This step starts with presenting the user with a number of distinguishable recommendations in terms of utility vectors. Each recommendation returned will stretch the dimension it represents (e.g. Saving) and relax on the other dimensions (e.g. Enjoyment, Location attractiveness, etc.). The process continues iteratively updating the utility vector every time, based on the feedback of the user until an exit point is reached (e.g., indicating “no difference” between recommendations presented). Upon exit, the recommendation space will be constructed according to the utility vector learned.

Finally, the Recommender constructs a set of diverse recommendations by splitting recommendation space into layers. We provide the user with a given number of diverse alternatives to choose from. The process ends if the user either chooses a recommendation (and consequently the services to implement the selected alternative are invoked) or exits the recommendation process. The user also has the option of restarting any of the main steps at anytime: Cluster, Optimize, and Diversify.

4 Utility Axis Selection

Now that we know what cluster the user is in, we will limit our recommendation space to the chosen cluster, and then deploy an iterative method to learn the user’s preference with respect to \( n \) dimensional utility space (e.g., Enjoyment, Saving, Location attractiveness, etc.). Intuitively, at each step of the iterative process, we maintain a set of utility axes, which become “closer” to each other with every iteration until the user can no longer differentiate among them in terms of the preference. At that time the iterative process stops, and a final utility function is constructed. We first describe the overall process and then summarize it with an algorithm.

Recommendations space \( \mathbb{R} \), consists of composite products and services, each recommendation is represented by a tuple in a Service Metric View. A recommendation could be a vacation package the user can choose. Each recommendation is mapped to a utility vector \( \vec{u} \), from an \( n \) dimensional utility space \( U \), which is presented as \( \mathbb{R}^n_+ \), we denote this mapping by: \( \vec{U} : \mathbb{R} \rightarrow \mathbb{R}^n_+ \). Components of a utility vector \( \vec{u} = (u_1, u_2, \cdots, u_n) \), are associated with metrics such as Enjoyment, Saving, Location attractiveness, etc. Each metric has an associated domain \( D_i \), \( 1 \leq i \leq n \). For example \( D_{\text{Saving}} = \mathbb{R}_+ \), \( D_{\text{Enjoyment}} = \{0, 1, \cdots, 10\} \), \( D_{\text{Location}} = \{0, 1, \cdots, 10\} \). We assume the Location metric represents the attractiveness on a scale from 0 to 10 (this can be extracted from domain knowledge). Each domain \( D_i \) has a total ordering “better than” denoted \( \succeq D_i \). For example, for domain Saving, \( a_1 \succeq_{\text{Saving}} a_2 \iff a_1 \geq a_2 \).

We model the relative importance the user places in each dimension by means of a vector of weights \( \vec{w} = (w_1, w_2, \cdots, w_n) \), where \( |\vec{w}| = \sqrt{\sum_{i=1}^{n} w_i^2} = 1 \), which we call an axis. Each component \( w_i \) captures the weight of the \( i \)-th dimension. The total utility of a recommendation \( r_k \) w.r.t. axis \( \vec{u} \) is defined as \( U_{\vec{w}}(\vec{u}) = w_1 u_1 + w_2 u_2 + \cdots + w_n u_n = \vec{w} \cdot \vec{u} \).

In the beginning, we assume no prior knowledge of the users subjective weights along each dimension, and would like to learn it as follows. We start with \( n \) axes, that represent the original dimensions in the utility space i.e.,

\[
\begin{align*}
\vec{w}_1 &= (1, 0, \cdots, 0) \\
\vdots \\
\vec{w}_n &= (0, 0, \cdots, 1)
\end{align*}
\]

In every iteration, a current set of axes is modified as follows. For each axis in the current set, we select a recommendation that maximizes the total utility according to that axis. For example, if we have 3 axes, we would present the user with 3 different composite recommendations, each exhibiting the highest utility w.r.t. the corresponding axis. Figure 2 exemplifies recommendations proposed to the user.

Figure 2: Example of Utility Axis Selection

We then ask the user to partition these recommendations into up to \( k \) preference strata, where stratum 1 represents the best recommendations, stratum 2 the second best etc. Note that, each stratum may have 1 or more
recommendations in it, and that 2 or more recommendations in the same stratum indicates that the user doesn’t have a preference among them. Our goal from this step is to allow the user to inform the system of how to adjust the learned total utility function to better reflect her preference. The feedback extracted from the user, i.e., the preference strata, is used to move the current axis closer to the learned utility function.

We first replace each $\vec{w}_i$ as follows. Let $\vec{r}_i$ be a recommendation that maximizes $U_{\vec{w}_i}(\vec{u}_i)$, where $\vec{u}_i$ is a utility vector associated with $\vec{r}_i$. We then replace $\vec{w}_i$ with the axis $\vec{u}_i/\|\vec{u}_i\|$, where the notation $\| \cdot \|$ means the norm of the vector. Then, for every rank $k$, we calculate the normalized mean,

$$\mu_i = \mu(\{\vec{u}_i|\text{rank}(\vec{r}_i) = k\}) = \frac{\sum_{\text{rank}(\vec{r}_i) = k} \vec{u}_i}{\sum_{\text{rank}(\vec{r}_i) = k} 1} \quad (2)$$

We now build new axes $\vec{w}_1, \ldots, \vec{w}_k$, where $k$ is the number of strata, as follows:

- For stratum 1, $\vec{w}_1 := \vec{\mu}_1$
- For stratum 2, $\vec{w}_2 := \mu(\vec{w}_1, \vec{\mu}_2)$
- For stratum $i$, $\vec{w}_i := \mu(\vec{w}_1, \ldots, \vec{w}_{i-1}), \vec{\mu}_i$,
  where $3 \leq i \leq k$

Intuitively, after adjusting $\vec{w}_1, \ldots, \vec{w}_{i-1}$, we do not yet know the user’s preference among them, but do know that they are preferable over strata $i$, represented with $\vec{\mu}_i$. Therefore, we create new axis $\vec{w}_i$ as the vector mean of $\mu(\vec{w}_1, \ldots, \vec{w}_{i-1})$ and $\vec{\mu}_i$, intuitively moving it toward $\mu(\vec{w}_1, \ldots, \vec{w}_{i-1})$ “half way”. For example, assume the user is presented with 3 recommendations $\vec{r}_1$, $\vec{r}_2$, and $\vec{r}_3$, according to utility vectors $\vec{u}_1$, $\vec{u}_2$, and $\vec{u}_3$ respectively. The user placed recommendation $\vec{r}_1$ in stratum 1 and both recommendations $\vec{r}_2$ and $\vec{r}_3$ in stratum 2. First, for all recommendations, we replace $\vec{w}_i$ with the axis $\vec{u}_i/\|\vec{u}_i\|$. Second, we calculate the mean utility vector of $\vec{r}_2$ and $\vec{r}_3$ as $\vec{\mu}_2 = \mu(\vec{u}_2, \vec{u}_3)$. Third, we calculate $\vec{w}_2 := \mu(\vec{w}_1, \vec{\mu}_2)$ Finally, we use the resulting $\vec{w}_2$ to calculate the new recommendation $\vec{r}_2$ as shown in Figure 2. The iterative process continues each time with a new set of axes, until all proposed recommendations, optimal w.r.t. the current axes, are in a single stratum 1. This means recommendations presented are indifferent, i.e., the user can not differentiate among recommendations suggested. As a final step, we calculate the normalized mean one more time of the resulting axes to be used as the utility weight vector in the next step of our framework, described in section 5, that is diversity layering. Algorithm 1 captures the process of utility axis selection.

```
Algorithm 1 Algorithm of utility axis selection
1: for $i = 1$ to $n$ do
2:   $\vec{w}_i = \text{the vector with 1 on the } i\text{-th component and 0 everywhere else}$
3: end for
4: $p = n$
5: while $p > 1$ do
6:   for $p = 1$ to $p$ do
7:     $\vec{r}_i = \text{a recommendation which maximizes } U_{\vec{w}_i}$
8:     Recalculate $w_i$ using the weights of presented $\vec{r}_i$
9: end for
10: Ask the user to place each recommendation presented in a stratum where 1 is the best, 2 is next best, etc;
11: $\text{MaxRank} = \text{the max stratum label assigned by the user}$
12: for $k = 1$ to $\text{MaxRank}$ do
13:   Collect all recommendations labeled $k$
14:   $\vec{w}_k = \text{the mean of the weight vectors for recommendations labeled } k$
15: end for
16: for $k = 2$ to $\text{MaxRank}$ do
17:   $\vec{w}_k := \mu(\vec{w}_1, \ldots, \vec{w}_{k-1}), \vec{\mu}_k$
18: end for
19: $p = \text{MaxRank}$
20: end while
21: return $\vec{w}_1$
```

Figure 3: Recommendation view

5 Diversity Layering

Now that we know more about the user in terms of the utility axis, we construct the global utility function where weights given to each metric attribute reflect the attractiveness of recommendations to the user. However, giving recommendations by the utility learned may not provide sufficient diversity of recommendations. We would like to return a diverse set of recommendations with a range of options that are not too similar and which are ranked by the learned utility. For example, a person whose utility is mostly in favor of low price may decide to take a very attractive travel package even if the price is not minimal. In this section, we develop diversity layering method to provide diverse recommendations sorted by the utility. This is done by a recommendation view. The syntax template of a composite Recommendation View is depicted in Figure 3.
Each Recommendation View is associated with the corresponding Service Metric View (SMV), which appears in the from clause. Examples of SMV are Rental vehicle, Airline flights, Travel Accommodation, or a combined travel package. The where clause contains user constraints, e.g., the maximum budget and duration of travel. The select clause returns all the key and metric attributes of the service instance, along with utility that has been learned in the utility axis selection step. We introduce a new optimal clause layers which indicates how many layers to split the recommendation space into. The limit indicates the number of recommendations the user would like to be presented with. The limit value could be a configuration or user-defined parameter. Intuitively, to reach diversity we start with the optimal recommendation (in terms of the learned utility) and then dynamically partition the recommendation space into \( m \) layers. Recommendations in the first layer have the utility function close to the max utility up to \( \frac{1}{m}(U_{\text{max}} - U_{\text{min}}) \), i.e., their utility is in the interval \([U_{\text{max}} - \frac{1}{m}(U_{\text{max}} - U_{\text{min}}), U_{\text{max}}]\). Recommendations in the \( i \)-th layer have utility in the interval \([U_{\text{max}} - \frac{i}{m}(U_{\text{max}} - U_{\text{min}}), U_{\text{max}}]\). Within each layer we select \( n \) recommendations to maximize each dimension of the utility space in turn. Finally, we return the user a set of \( k \) recommendations (in terms of the learned utility) chosen from the \( m \) layers, after removing duplicates and sorting them by utility. We first illustrate the diversity layering method using an example, and then present a formal definition of diversity layering. Consider an example depicted in Figure 4, for three layers, i.e., \( m = 3 \) in the layers clause and given order by utility, \( u_1, u_2 \). There are two dimensions, \( u_1 \) and \( u_2 \) of the utility space (i.e. metrics relevant to selection), and \( U_w \) is the learned global utility. For example, \( u_1 \) can stand for (total-budget cost), i.e., Saving, and \( u_2 \) for the Location attractiveness factor of family travel. The two-dimensional polyhedral set in the figure depicts all possible utility vectors of recommendations.

We note that recommendations (e.g. \( r_0, r_{11}, r_{12}, r_{21}, r_{22}, r_{31}, r_{32} \)) residing on the “skyline” are the non-dominated choices of the recommendation space. For example, recommendation \( d \) in the figure is dominated by \( r_0 \), the utility vector of \( r_0 \) is higher than that of \( d \) in both dimensions. First, we eliminate all dominated recommendations, and thus are left with the “skyline”, which is denoted with the thick line. From the order by clause, the user indicated that \( u_1 \) is more important than \( u_2 \). The recommendation \( r_0 \) maximizes the global utility \( U \). Then the skyline is split into three layers. The first is corresponding to the area above the highest dashed line which corresponds to recommendations with utility \( U \geq U_{\text{max}} - \frac{1}{3}(U_{\text{max}} - U_{\text{min}}) \), and select recommendations \( r_{11}, r_{12} \) that maximize dimensions \( u_1 \) and \( u_2 \) respectively. Similarly, the second and third layers correspond to \( U \geq U_{\text{max}} - \frac{1}{3}(U_{\text{max}} - U_{\text{min}}) \) and \( U \geq U_{\text{min}} \) respectively. As a result, we extract recommendations \( r_{21}, r_{22}, r_{31}, r_{32} \). If the user requests four recommendations (i.e., limit = 4), then \( (r_0, r_{11}, r_{12}, r_{21}) \) will be returned in this order, and if limit = 6, \( (r_0, r_{11}, r_{12}, r_{21}, r_{22}, r_{31}, r_{32}) \) will be returned. Intuitively, maximizing each metric component in turns gives diversity, while restricting the global utility within its layer controls the distance from the optimal global utility. More formally, given two recommendations \( r_1, r_2 \) in \( \mathbb{R} \) and the corresponding utility vectors \( \vec{u} = (u_1, u_2, \ldots, u_n) \) and \( \vec{v} = (v_1, v_2, \ldots, v_n) \) respectively in \( \mathbb{R}^n \), we say that \( r_1 \) dominates \( r_2 \), denoted \( r_1 \succeq r_2 \), if \( u_i \geq v_i \) for all \( i \), \( 1 \leq i \leq n \). Intuitively, one recommendation dominates another if it is at least as good as the other in all respects. We denote by \( \mathbb{R} \) the set of all non-dominated recommendations i.e.,

\[
\hat{\mathbb{R}} = \{ r | (\exists r' \in \mathbb{R}) r' \succeq r \} \tag{3}
\]

As before, \( \hat{U} \) is the utility mapping, \( \hat{U} : \mathbb{R} \rightarrow \mathbb{R}^n_+ \). Below we use \( U_{\text{max}}, U_{\text{min}} \) defined as:

\[
U_{\text{max}} = \max U_w(\hat{U}(r)) \text{ s.t. } r \in \hat{\mathbb{R}} \tag{4}
\]

\[
U_{\text{min}} = \min U_w(\hat{U}(r)) \text{ s.t. } r \in \hat{\mathbb{R}} \tag{5}
\]

**Definition 1** An \( m \)-layered recommendation set is a set \( \{r_0, r_{11}, \ldots, r_{1n}, r_{21}, \ldots, r_{2n}, \ldots, r_{m1}, \ldots, r_{mn} \} \) of recommendations such that:

1. \( r_0 \in \hat{\mathbb{R}} \) for all \( 1 \leq i \leq n, 1 \leq j \leq m \) (i.e. only non-dominated recommendations are included).

![Figure 4: Diversity Layering Example](image-url)
2. \( U(r_0) \geq U(r_{11}) \geq \cdots \geq U(r_{1n}) \geq \cdots \geq U(r_{mn}) \) for all \( 1 \leq i \leq n, 1 \leq j \leq m \).

3. \( r_0 = \arg\max_{r \in \mathbb{R}} U_w(\overline{U}(r)) \) s.t. \( r \in \mathbb{R} \).

4. For every \( 1 \leq i \leq n, 1 \leq j \leq m \), \( r_{ij} = \arg\max_{r \in \mathbb{R}} U_{ij}(\overline{U}(r)) \) s.t. \( r \in \mathbb{R} \) \( \land \) \( U_w(\overline{U}(r)) \geq U_{\max} - \frac{1}{m}(U_{\max} - U_{\min}) \).

An \( m \)-layered \( k \)-recommendation is a sequence \( (r_0, r_1, \ldots, r_{k-1}) \) such that:

- \( r_0, r_1, \ldots, r_{k-1} \in \{r_0, r_{11}, \ldots, r_{mn}\} \)
- All \( r_0, r_1, \ldots, r_{k-1} \) are different
- \( r_0, r_1, \ldots, r_{k-1} \) are sorted in lexicographical order of \( U_w, u_1, \ldots, u_n \).

Finally, a \( k \)-recommendation is an \( m \)-layered \( k \)-recommendation, where \( m \) is selected to be the minimum of the number of layers that produce at least \( k \) recommendations. Note that a \( k \)-recommendation is returned when layers \( m \) clause is omitted.

### 6 Validation - User Case Study

In order to evaluate our proposed recommender system, we conducted a user study of 30 users. The objective of the study was to verify the following hypotheses:

1. Our system achieves a better recall and precision than a non-personalized travel recommender system.

2. The interactive elicitation of the utility axis imposes an acceptable overhead to the users.

3. The vertical diversity layering step increases the recall.

Hypothesis 1 is justified by the widespread adoption of recall and precision as a standard measure for validation in Information Retrieval(IR) [e.g., 12,23,24,25]. Recall is the percentage of correctly predicted “high” ratings among all the ratings known to be “high”[1], while precision is the percentage of truly “high” ratings among those that were predicted to be “high” by the recommender system[1]. The reason we chose Hypothesis 2 is to measure the burden caused by our interactive framework and determine if it is acceptable in view of the perceived benefits. Finally, we wanted to test Hypothesis 3 so that we can assess the usefulness of the vertical diversity layering. We loaded our systems database with real data about vacation packages extracted from a popular travel commercial website, which we will call System A.

We conducted the user study aiming to estimate the recall and precision of our system. Specifically, we submitted a request for a three week vacation in Los Angeles, California starting on May 1, 2009, including roundtrip airfare from Washington Dulles Airport. We then extracted all packages returned by System A, keeping just the cost and number of stars (enjoyment) of each package. Since we wanted to evaluate the quality of the top results returned by our system against System A, we limited the number of results shown to the user to five. We surveyed a total of 30 users, all working professionals.

For COD, in the first phase we learned the users utility function through a two step dialog: at each step, we present the user with two choices, one with a better enjoyment (number of stars), and the other with a smaller cost, as described in Section 4. Depending on the user’s answers, their inferred utility function can reflect either: a strong sensitivity to price (PP), a moderate preference for less expensive packages (PQ), a moderate preference to higher quality packages (QP), or a strong bias towards high quality packages (QQ). The distribution of answers was as follows: 6 users in the PP category, 18 users in the PQ category, 4 users in the QP category, and 2 users in the QQ category. In the second phase, we computed five recommendations using the diversity layering method described in Section 5 and presented them to the user in descending order of their utility (according to the personalized utility function estimated in the first phase). For System A, we just presented the top five recommendations in the order suggested by the website.

![Figure 5: Average Recall vs. Rank](image)

We then asked the users to rate each of the ten recommendations on a scale of 1 to 5, where 5 means “de-
nearly buy”, 4 means “likely to buy”, 3 means “neutral”, 2 means “unlikely to buy” and 1 means “definitely not buy”. We point out that users surveyed did not know which recommendation set came from which system.

In order to estimate recall at a given rank, we gathered all the packages rated 4 or 5 by any user, call that set “Buy” (this is the set of all recommendations which the users considered buying). Then, we counted how many of these recommendations were present in the top k results returned by our COD system and System A. Formally,

\[
\text{recall}(k) = \frac{|\{r \in \text{Buy}| \text{rank}(r) \leq k\}|}{|\text{Buy}|}
\]

For each of the two systems we then computed the average recall at each rank k by taking the average of recall(k) across all the 30 users. The results are summarized in Figure 5. As we can see, already at rank 2, our system returned 18% of the relevant packages compared to 6% for System A. Moreover, our system returned 26% of the relevant recommendations in the top 5 results. By contrast, System A returned only 16% of the relevant results in the top 5.

In order to estimate precision, we counted how many of the recommendations in the top k results were actually in the set Buy. Formally,

\[
\text{precision}(k) = \frac{|\{r \in \text{Buy}| \text{rank}(r) \leq k\}|}{k}
\]

For each of the two systems we computed the average precision at each rank k by taking the average of precision(k) across all the 30 users. The results are summarized in Figure 6. As we can see, at rank 1, all of the recommendations returned by our system in the top position were actually relevant, compared to 36% for System A. At rank 2, 75% of our recommendations were relevant compared to 18% for System A. In fact, at every rank our system considerably outperformed System A with respect to precision. In order to determine the statistical significance of our results, we assume a uniform distribution of ratings over the set of available packages, meaning that a package selected at random has an equal chance of receiving any of the 5 user ratings. Under this assumption, the probability of a randomly selected package to be rated “buy” (rating 4 or 5) is 2/5. Our system selected 52 packages rated Buy out of 150 trials (5 results for 30 users), which can occur by chance with a probability of 2.76%. Therefore, hypothesis 1 is confirmed with adequate statistical significance (p-value of 0.0276). We believe the better quality of our results comes from personalizing the utility function according to the learned user preferences. While System A returns the same set of results to every user, we make an attempt to learn more about what each particular user is interested in. Since this extra step is imposing an extra burden on the end user, we included at the end of our survey the following question: “Would you be willing to spend a few more minutes answering few questions so that the system can learn about you and, consequently, provide you more personalized recommendations, considering the amount of the transaction?”. The vast majority of the surveyed users (27 out of 30) answered “yes”, which confirms hypothesis 2.

Finally, for hypothesis 3, we examined the distribution of the “buy” ratings within the vertical layers in the ranked result lists. Here, we observed that all the “buy” ratings were restricted to the first position and the top-most layer (positions 2 and 3), and all the recommendations in the bottom layer were rated either “neutral” or “not buy”, therefore not improving the recall. We believe this is due to the rather low value we chose as a threshold for utility in this particular experiment. More work is needed to study whether a carefully calibrated threshold leads to improved recall beyond the top-most layer.

7 Related Work

Recommender systems have been extensively studied since the mid-1990s. Recent popular surveys (e.g., [1,5,10]) classify the current generation of recommendation methods into three main categories: content-based, collaborative, and hybrid recommendation approaches. Content-based systems (e.g., [12,13]) often employ classifier techniques that rely on information retrieval and information filtering to recommend an item to a user based upon a description of the item and a profile of the users.
interests. In contrast, Collaborative recommenders (e.g., [6,22]) use the preferences of “similar users, rather than the characteristics of an item, to make suggestions to the current user.

The hybrid approach (e.g., [8]) combines methods from collaborative and content-based approaches. More recently, utility-based, knowledge-based, and demographic systems have each suggested different techniques for providing recommendations. Systems employing one or more of these techniques have been proposed to recommend flights, movies, restaurants, and news articles (e.g., [1,5,7]). Another way to classify recommender systems is based on techniques, either heuristic-based (e.g., [22]) or model-based (e.g., [6]). The main difference between model-based techniques and heuristic-based techniques is how to calculate the utility (rating) predictions. In the heuristic-based approach, calculation of predicted utility (rating) is based on some ad hoc heuristic rules, whereas in the model-based approach, calculation is based on a model learned from the underlying data using statistical learning techniques [1]. Recommender systems employ different representation models, ranking methods, and learning techniques to recommend solutions in a variety of domains. For comprehensive analysis of recommender systems refer to [1,5,10,16]. Most relevant to our work are those systems supporting multi-attribute utility-based recommendation, multi-criteria ranking methods, utility function elicitation, and dynamic learning of user preferences.

Multi-attribute recommender systems characterize recommendation alternatives as associated attribute-value pairs. Case-based recommenders often evaluate recommendation alternatives according to their similarity to a target solution [2,11,14,15]. In contrast, utility-based recommenders make recommendations often based on a single utility score [5,10]. The COD recommender framework is used to construct utility-based recommenders; however, many of the COD components could be used to accommodate similarity-based recommendation.

Three different approaches for diversity were presented in [15], namely BoundedRandomSelection, GreedySelection, and BoundedGreedySelection where we can provide diversity by compromising similarity in a way it optimizes similarity-diversity trade-off. However, this work is based on measures such as Euclidean distance or hamming distance calculated for pairs of attribute values. Like the Diversity layering algorithm presented in this paper, the method proposed in [9] also makes recommendations by finding solutions that optimize one attribute of solution; however, the multi-criteria ranking method presented here optimizes each attribute while bounding the allowable degradation in overall utility. In this way, the recommendations made by Diversity layering offer the user a broad view of the solution space while maintaining an acceptable overall utility.

8 Conclusions and Future Work

In this paper we studied methods for providing recommendations on composite bundles of products and services, which are dynamically defined using database views extended with decision optimization using mathematical programming. We proposed a framework for finding a diverse recommendation set, when no prior knowledge on user preference is given, which includes (1) finding recommendation cluster, (2) user utility elicitation using decision optimization, and (3) partitioning the recommendation space into layers to extract a balanced set of both optimal and diverse recommendations. We also conducted a preliminary experimental study, which showed that the COD framework significantly outperforms a popular commercial system in terms of precision and recall.

Many research questions remain open. They include (1) identify the right balance between optimality of recommendations (in terms of the learned utility) and diversity, and refining algorithms to reflect that balance; (2) developing efficient algorithms for diversity layering queries, which take advantage of simultaneous optimization of multiple constraint problems; (3) expanding the diversity layering to incorporate users’ feedback on diverse recommendations. (4) Complete the native integration of the CARD/COD framework prototype system.

References


