Breaking out of the Box of Recommendations: From Items to Packages

Min Xie
Dept. of Computer Science,
Univ. of British Columbia
minxie@cs.ubc.ca

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

Peter T. Wood
Dept. of CS and Inf. Syst.,
Birkbeck, U. of London
ptw@dcs.bbk.ac.uk

ABSTRACT

Classical recommender systems provide users with a list of recommendations where each recommendation consists of a single item, e.g., a book or DVD. However, several applications can benefit from a system capable of recommending packages of items, in the form of sets. Sample applications include travel planning with a limited budget (price or time) and twitter users wanting to select worthwhile tweeters to follow given that they can deal with only a bounded number of tweets. In these contexts, there is a need for a system that can recommend top-k packages for the user to choose from.

Motivated by these applications, we consider *composite recommendations*, where each recommendation comprises a set of items. Each item has both a value (rating) and a cost associated with it, and the user specifies a maximum total cost (budget) for any recommended set of items. Our composite recommender system has access to one or more component recommender systems focusing on different domains, as well as to information sources which can provide the cost associated with each item. Because the problem of generating the top recommendation (package) is NP-complete, we devise several approximation algorithms for generating top-k packages as recommendations. We analyze their efficiency as well as approximation quality. Finally, using two real and two synthetic data sets, we subject our algorithms to thorough experimentation and empirical analysis. Our findings attest to the efficiency and quality of our approximation algorithms for top-k packages compared to exact algorithms.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - Information Filtering

General Terms

Algorithms, Theory

Keywords

Recommendation Algorithms, Optimization, Top-k Query Processing

RecSys2010, September 26–30, 2010, Barcelona, Spain.

1. INTRODUCTION

Recommender systems (RecSys) have become popular and have become an essential driver of many applications including web services [2]. However, classical RecSys provide recommendations consisting of single items, e.g., books or DVDs. Several applications can benefit from a system capable of recommending packages of items, in the form of sets. For example, in trip planning, a user is interested in suggestions for places to visit, or points of interest (POI). There may be a cost to visiting each place (time, price, etc.). Optionally, there may be a notion of compatibility among items in a set, modeled in the form of constraints: e.g., "no more than 3 museums in a package", "not more than two parks", "the total distance covered in visiting all POIs in a package should be $\leq$ 10 km." etc. The user may have a limited budget and is interested in suggestions of compatible sets of POIs such that the cost of each set is under budget and its value (as judged from ratings) is as high as possible. In these applications, there is a natural need for top-k recommendation packages for the user to choose from. Some so-called "third generation" travel planning web sites, such as NileGuide[1] and YourTour[2], are starting to provide certain of these features, although in a limited form.

As another application, in social networks like twitter, one of the important challenges is helping users with recommendations for tweeters to follow, based on the topics of their interest[3]. Tweeters are ranked based on how influential they are [20] and currently any new user is presented with a list of influential tweeters on each topic from which they manually choose tweeters they would like to follow[3]. To automate tweeter recommendation, a tweeter's influence score can be treated as their value and the frequency with which they tweet as their cost. Compatibility may correspond to the constraint that a given set of topics should be covered. Since a user can only deal with a bounded number of tweets in a day, given a user's topics of interest, it would be useful to select compatible sets of tweeters to follow such that their total influence score is maximized and the total cost is below a budget. Once again, it would be beneficial to give the user choice by presenting them with the top-k sets of recommended tweeters to follow. We also note that some newly founded startups like Followformation[4] are beginning to provide services on recommending to users the top-k influential tweeters in a specific domain.

[1] http://www.nileguide.com
[2] http://www.yourtour.com
[3] http://blog.twitter.com/2010/01/power-of-suggestions.html
[4] http://followformation.com

Motivated by these applications, we consider *composite recommendations*, where each recommendation comprises a set of items. Each item has both a value (rating or score) and a cost associated with it, and the user specifies a maximum total cost (budget) for any recommended set of items. Our composite recommender system consists of one or more recommender systems focusing on different domains. These component RecSys serve (i.e., recommend) top items in non-increasing order of their value (explicit or predicted ratings). In addition, our composite system also has access to information sources (which could be databases or web services) which provide the cost associated with each item.

In our setting, the problem of generating the top recommendation (package) is NP-complete as it models the Knapsack problem [10]. Because of this and the fact that we expect the component recommender systems to provide ratings for large numbers of items and access to these ratings can be relatively expensive,[5] we devise approximation algorithms for generating top-k packages as recommendations.

Other researchers have considered complex or composite recommendations. CARD [4] and FlexRecs [12] are comprehensive frameworks in which users can specify their recommendation preferences using relational query languages extended with additional features or operators. In contrast, we are concerned with developing efficient algorithms for combining recommendations from RecSys that provide only ratings for items. Closer to our work is [3] which is concerned with finding packages of entities, such as holiday packages, where the entities are associated in some way. However, their packages are of fixed size, whereas we allow packages of variable size. CourseRank [16, 17] is a system for providing course recommendations to students, based on the ratings given to courses by past students and subject to the constraints of degree requirements. While we do not capture all CourseRank constraints, in our framework we have item costs and user budgets—essential features of the application areas we consider for deployment of our system—which are not captured by CourseRank. Similarly, item costs and user budgets are not considered for team formation in [13].

In this paper, for space limitations, we restrict attention to the problem of recommending packages when there is just one component RecSys and no compatibility constraint is imposed. The problem remains intractable and still warrants approximation algorithms. We discuss in Sec. 5 how to extend our algorithms when multiple component RecSys and compatibility constraints are present.

The roadmap of the paper is as follows. After discussing related work in more detail (Sec. 2), we present the architecture of our system and give a precise definition of the problem we study (Sec. 3). We then describe the approximation algorithms we have developed for returning top-k composite recommendations (Sec. 4). We first present a 2-approximation algorithm that is instance optimal [6] with an optimality ratio of one. This means that any other 2-approximation algorithm, that can only access items in non-increasing order of their value, must access at least as many items as our algorithm. However, this algorithm makes repeated calls to a routine for solving exactly the problem of finding the top-rated package under budget, from those items that have been accessed from the component RecSys so far. Because this is an NP-complete problem, we then develop a greedy algorithm for returning top-k composite recommendations. This algorithm is also guaranteed to return a 2-approximation, but is no longer guaranteed to be instance optimal. It is interesting to note that the *average* value of packages returned by our approximation algorithms is higher than that returned by the exact algorithm. This is because an exact algorithm will add low-value items to a recommendation in order to maximize value. However, from a user's perspective the recommendations returned by the approximation algorithms may sometimes be preferable.

In Sec. 6 we subject our algorithms to thorough empirical analysis using two real data sets – TripAdvisor and MovieLens – and two synthetic data sets. We first investigate the quality of the recommendations produced by our approximation algorithms. Our findings confirm that our algorithms always produce recommendations that are 2-approximations, with many of them being close to optimal. We then compare the efficiency of our instance optimal and greedy approximation algorithms with that of an exact algorithm in terms of running time and number of items accessed. Our results indicate that our greedy algorithm is always significantly faster than the other two algorithms, while the greedy and instance optimal algorithms usually access substantially fewer items than the exact algorithm. Finally, we discuss future work and conclude the paper in Sec. 7. To the best of our knowledge, this is the first time instance optimality is established in the context of approximation algorithms.

2. RELATED WORK

Closest to our work is [3], where they are interested in finding top-k tuples of entities. Examples of entities include cities, hotels and airlines, while packages are tuples of entities. Instead of querying recommender systems, they query documents using keywords in order to determine entity scores. A package in their framework is of fixed size, e.g., one city, one hotel and one airline, with fixed associations among the entities essentially indicating all possible valid packages. Instead, we allow for packages (composite recommendations) of variable size, subject to a budget constraint. Associations between entities can be easily captured in our framework using the notion of compatibility of sets.

Other closely related work is [5] where a novel framework is proposed to automatically generate travel itineraries from online user-generated data like picture uploads and formulate the problem of recommending travel itineraries of high quality where the travel time is under a given time budget. However, in this work, the value of each POI is determined by the number of times it is mentioned by users, whereas in our work, item value is a personalized score which comes from an underlying recommender system and accessing these items is constrained to be in value-sorted order. Unlike [5], we optimize item accesses, establish instance optimality, and provide algorithms for generating top-k packages.

CARD [4] is a framework for providing top-k recommendations of composite products or services. Fine-grained control over specifying user requirements as well as how atomic costs are combined is provided by an SQL-like language extended with features for decision support. Each composite recommendation is of fixed size, making the problem simpler; thus CARD returns exact not approximate solutions.

Similarly, FlexRecs [12] is a sophisticated system for defining complex recommendations from relational data. Recommendation requirements are specified by relational algebra

[5]Especially when the ratings need to be predicted.

expressions enhanced with *extend*, *recommend* and *blend* operators. As with [3, 4], recommendations are of fixed size and thus solutions are exact.

In our setting of access to component RecSys (but with a restricted notion of binary boolean compatibility), the problem of finding the top-k *fixed-size* packages is simpler than that of finding packages of variable size; it can be solved efficiently using Rank Join [7].

CourseRank [17] is a project motivated by a course planning application for students, where constraints are of the form "take k_i from S_i," where k_i is a non-negative integer and S_i is a set of courses. Similar to our work, each course in this system is associated with a score which is calculated using an underlying recommendation engine. Given a number of constraints of the form above (and others), the system finds a minimal set of courses that satisfies the requirements and has the highest score.

Later work [16] extends CourseRank with prerequisite constraints, and proposes several approximation algorithms that return high-quality course recommendations which satisfy all the prerequisites. As in our work, such recommendations need not be of fixed size. However, [16, 17] do not consider the *cost* of items (cf. courses) which can be important for applications like trip planning and twitter.

The problem of team formation is studied in [13]. Here each person has a set of skills and pairs of people have a collaboration cost associated with them (lower cost indicates better collaboration). Given a task requiring a set of skills, the problem is to find a set of people whose skills cover those required and who have a low aggregated collaboration cost. The notion of compatibility in our framework can model their collaboration cost. Similar to CourseRank, the people (items) themselves are not rated. A further difference with our approach is that we wish to maximize the aggregate item (cf. people) ratings subject to item and compatibility costs, rather than minimize compatibility cost.

Although we do not include in our system complex constraints such as those in [13, 16, 17], for applications where complex constraints exist, we can leverage existing work to post-process each composite recommendation generated by our algorithms to ensure that the constraints are satisfied.

Finally, motivated by online shopping applications, [18] studies the problem of recommending "satellite items" related to a given "central item" subject to a cost budget. The resulting notion of packages is quite restricted compared to our framework, and item values are not taken into account.

3. ARCHITECTURE AND PROBLEM

3.1 System Architecture

In a traditional RecSys, users rate items based on their personal experience, and these ratings are used by the system to predict ratings for items not rated by an active user. The predicted ratings can be used to give the user a ranked recommendation (item) list.

As shown in Figure 1, our composite recommendation system is composed of one or more component RecSys and has access to external sources that provide the cost of a given item. An external source can be a local database or a web service. E.g., Amazon.com can be consulted for book prices. In terms of computation, we abstract each RecSys as a system which serves items in non-increasing order of their value (rating or score) upon request. In addition, the system includes a compatibility checker module, which checks whether a package satisfies compatibility constraints, if any. We assume the compatibility checker consults necessary information sources in order to verify compatibility.

The user interacts with the system by specifying a cost budget, an integer k, and optionally compatibility constraints on packages. The system finds the top-k packages of items with the highest total value such that each package has a total cost under budget and is compatible.

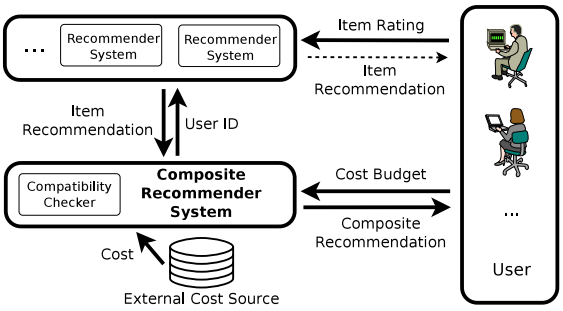

Figure 1: System Architecture

3.2 Problem Statement

Given a set N of items and U of users, an active user $u \in U$, and item $t \in N$, we denote by $v_u(t)$ the *value* of item t for user u. We denote the value as $v(t)$ when the active user is understood. A RecSys predicts $v(t)$ when it is not available, by using the active user's past behavior and possibly that of other similar users. For $t \in N$, we denote by $c(t)$ the *cost* of item t. Given a set of items $R \subset N$, we define $c(R) = \Sigma_{t \in R}\, c(t)$ and $v(R) = \Sigma_{t \in R}\, v(t)$. Given a cost budget B, a set of items $P \subset N$ is called *feasible* if $c(P) \leq B$. *In this paper, for space limitations, we focus on the following problem (with extensions discussed in Sec. 5):*

Definition 1 (Top-k Composite Recommendations). *Given an instance I of a composite recommendation system consisting of one component RecSys and an external information source, a cost budget B and an integer k, find the top-k packages $P_1, ..., P_k$ such that each P_i is feasible and among all feasible packages $P_1, ..., P_k$ have the k highest total values, i.e., $v(P) \leq v(P_i)$ for all feasible packages $P \notin \{P_1, ..., P_k\}$.*

When $k = 1$, the top-k composite recommendation problem (CompRec) can be viewed as a variation of the classical 0/1 knapsack problem [10] with the restriction that items can be accessed only in non-increasing order of their value. Without loss of generality, we assume all items have cost smaller than the cost budget B.

Note that ratings of items from the component RecSys are retrieved using sorted access, while the cost of a given item is obtained via random access. Let c_s and c_r be the costs associated with these accesses. Then the total *access cost* of processing n items is $n \times (c_s + c_r)$. Notice that c_s and c_r can be large compared to the cost of in-memory operations: for both accesses information needs to be transmitted through the Internet, and for the sorted access, $v(t)$ may need to be computed. So, well known algorithms for knapsack which need to access *all* items [10] may not be realistic. Thus, an efficient algorithm for top-k CompRec should minimize the total access cost, i.e., it should minimize the number of items accessed and yet ensure the top-k packages are obtained.

It can be shown that if we have no background knowledge about the cost distribution of items, in the worst case, we must access all items to find top-k packages. In order to facilitate the pruning of item accesses, we thus assume some background information about item costs is precomputed and maintained at the composite RecSys. The background cost information, which we denote generically by $\mathcal{BG}$, can be a histogram collected from the external cost source or something as simple as a minimum item cost c_{min}. This information can be materialized in our system and be refreshed regularly by re-querying the cost source.

Our composite recommendation problem can be considered as a special case of a resource-limited knapsack problem where in addition to quality guarantee, the number of items to be accessed should also be minimized. So standard algorithms for knapsack, e.g., exact algorithms [10] and approximation algorithms [8, 19] may not be efficient as they always need to access the entire dataset. The only known variation of knapsack which deals with resource limitation is the Online Knapsack Problem [15]. However, for this problem, no access constraints are considered, only competitiveness in terms of quality is studied. And furthermore, no information about items can be inferred, which makes the problem significantly harder and difficult to approximate.

4. COMPOSITE RECOMMENDATIONS

In this section, we develop several approximation algorithms for top-1 CompRec, after which we extend them to handle top-k CompRec.

4.1 Instance Optimal Algorithms

As identified in Section 3.2, top-1 CompRec is a variation of the 0/1 knapsack problem where the underlying items can be accessed only in non-increasing order of their value (rating). Because of the huge potential size of the sets of items and the high cost of retrieving item information from the source, it is crucial for an algorithm to find high-quality solutions *while minimizing the number of items accessed*. Furthermore, as the 0/1 knapsack problem is NP-Complete [10], we need to develop efficient approximation algorithms.

4.1.1 Top-1 Composite Recommendation

Given an instance I of top-1 CompRec, let $\mathcal{BG}$ denote the known background cost information and $S = \{t_1, ..., t_n\}$ be the set of items which have been accessed or seen so far.

Let $\bar{v}$ be the value of the first accessed item, because items are accessed in the non-increasing order of their value, $n \cdot \bar{v}$ is a trivial upperbound on the value that can be achieved by any knapsack solution for S.

For each $i \in \{1, ..., n\}$ and $v \in \{1, ..., n \cdot \bar{v}\}$, let $SS_{i,v}$ denote a subset of $\{t_1, ..., t_i\}$ whose total value is exactly v and whose total cost is minimized. Let $C(i, v)$ be the cost of $SS_{i,v}$ ($C(i, v) = \infty$ if the corresponding $SS_{i,v}$ doesn't exist), then it is well known from previous work [19, 10] that a pseudo-polynomial algorithm can be utilized to get the optimal knapsack solution for S by first calculating all $C(i, v)$ using the following recursive function and then choosing the maximum value achievable by any subset $SS_{n,v}$ of which the total cost is bounded by budget B, i.e., $max\{v|C(n, v) \leq B\}$.

$$C(i+1, v) = \quad (1)$$
$$\begin{cases} \min\{C(i, v), c(t_{i+1}) + C(i, v - v(t_{i+1}))\} & \text{if } v(t_{i+1}) \leq v \\ C(i, v) & \text{otherwise} \end{cases}$$

Let the background cost information be $\mathcal{BG} = c_{min}$, which is the minimum cost of all items, let $v_{min} = \min_{t \in S} v(t)$ be the minimum value of all accessed items, and let OPT be the true optimal solution to the underlying top-1 CompRec Instance I. We can get an upperbound V^* on the value $v(OPT)$ of the optimal solution using the following algorithm MaxValBound. (Proofs of all lemmas and theorems can be found in [1].)

Algorithm 1: MaxValBound(S, C, B, $\mathcal{BG}$)

```
1 V* = ⌊B / c_min⌋ × v_min
2 for v ∈ {1, ..., n · v̄} do
3     if C(n, v) < B
4         V*=max{V*, v + ⌊(B − C(n,v)) / c_min⌋ * v_min}
5 return V*
```

LEMMA 1. *Given S, C, B, $\mathcal{BG}$, the value V^* returned by MaxValBound is an upperbound on $v(OPT)$. And V^* is tight in that there exists a possible unseen item configuration for which V^* is achievable by using a subset of accessed items and feasible unseen items*[6].

Given the upper bound V^* on the optimal solution, we next propose a 2-approximation algorithm for top-1 CompRec which is guaranteed to be instance optimal (see below). The algorithm, InsOpt-CR, is shown as Algorithm 2. One item is retrieved from the source at each iteration of the algorithm (lines 3–4). After accessing this new item, we can use the pseudo-polynomial algorithm to find an optimal solution R^o *over the accessed itemset* S (line 5). We calculate the upper bound value V^* of the optimal solution using MaxValBound. If $v(R^o) \geq \frac{1}{2} \times V^*$, the algorithm terminates; if not, it continues to access the next item (lines 7–8). The following example shows how InsOpt-CR works.

Algorithm 2: InsOpt-CR(N, B, $\mathcal{BG}$)

```
1 S ← An empty buffer
2 while TRUE do
3     t ← N.getNext()
4     S.Insert(t)
5     (R^o,C) ← OptimalKP(S, B)
6     V* = MaxValBound(S,C,B,BG)
7     if v(R^o) ≥ 1/2 × V*
8         return R^o
```

EXAMPLE 1. *Let $I = \{t_1, t_2, \ldots, t_n\}$ be a top-1 CompRec instance, where $v(t_1) = v(t_2) = 101$, $c(t_1) = c(t_2) = 100$, for $i = 3, \ldots, 101$, $v(t_i) = c(t_i) = 1$, and for $i = 102, \ldots, n$, $v(t_i) = 1$ and $c(t_i) = 0.5$. Let $B = 199$. Clearly, $\mathcal{BG} = c_{min} = 0.5$. After accessing the first 101 items, $S = \{t_1, \ldots, t_{101}\}$, $R^o = \{t_1\} \cup \{t_3, \ldots, t_{101}\}$, $v(R^o) = 200$. Because $c_{min} = 0.5$ and $v_{min} = 1$, we can calculate $V^* = 398$ and InsOpt-CR will stop since $v(R^o) \geq \frac{1}{2} \times V^*$.*

Given a top-1 CompRec instance I with optimal solution OPT, because $V^* \geq v(OPT)$, if $v(R^o) \geq \frac{1}{2} \times V^*$, then $v(R^o) \geq \frac{1}{2} \times OPT$, so OptIns-CR returns a correct 2-approximation of OPT.

[6]An unseen item t is feasible iff. $v(t) \leq v_{min}$ and $c(t) \geq c_{min}$

To analyze the optimality of our proposed algorithm, we utilize the notion of *instance optimality* proposed in [6].

DEFINITION 2. **Instance Optimality:** *Let $\mathcal{A}$ be a class of algorithms, and let $\mathcal{I}$ be a class of problem instances. Given a non-negative cost measure $cost(\mathcal{A}, \mathcal{I})$ of running algorithm $\mathcal{A}$ over $\mathcal{I}$, an algorithm $A \in \mathcal{A}$ is* instance optimal *over $\mathcal{A}$ and $\mathcal{I}$ if for every $A' \in \mathcal{A}$ and every $I \in \mathcal{I}$ we have $cost(A, I) \leq c \cdot cost(A', I) + c'$, for constants c and c'. Constant c is called the* optimality ratio.

To prove the instance optimality of InsOpt-CR, we first show the following.

LEMMA 2. *Given any top-1 CompRec instance I and any 2-approximation algorithm A with background cost information $\mathcal{BG}$ and the same access constraints as InsOpt-CR, A must read at least as many items as InsOpt-CR.*

THEOREM 1. *Let $\mathcal{I}$ be the class of all top-1 CompRec instances, and $\mathcal{A}$ be the class of all possible 2-approximation algorithms that are constrained to access items in non-increasing order of their value. Given the same background cost information $\mathcal{BG}$, InsOpt-CR is instance optimal over $\mathcal{A}$ and $\mathcal{I}$ with an optimality ratio of one.*

4.1.2 Top-k Composite Recommendations

In addition to the best composite recommendation, it is often useful to provide the user with the top-k composite recommendations, where k is a small constant. In this section, we extend the algorithm proposed in Section 4.1.1 to one that returns the top-k composite recommendations. Similar to the top-1 case, due to the hardness of the underlying problem, we seek an efficient approximation algorithm which can give us high quality recommendations.

Given an instance I of top-k CompRec, assume $\mathcal{R}^I$ is the set of all *feasible composite recommendations*, i.e., $\mathcal{R}^I = \{R \mid R \subseteq N \wedge c(R) \leq B\}$). Following Fagin et al. [6] and Kimelfeld et al. [11], we define an α-approximation of the top-k composite recommendations to be any set $\mathcal{R}_k$ of $\min(k, |\mathcal{R}^I|)$ composite recommendations, such that, for all $R \in \mathcal{R}_k$ and $R' \in \mathcal{R}^I \backslash \mathcal{R}_k$, $v(R) \geq \frac{1}{\alpha} \times v(R')$.

To produce top-k composite recommendations, we will apply Lawler's procedure to InsOpt-CR. Lawler's procedure [14] is a general technique for enumerating optimal top-k answers to an optimization problem, which relies on an efficient algorithm to find the optimal solution to the problem.

Let InsOpt-CR-Topk be the InsOpt-CR algorithm modified using Lawler's procedure. All we need to change is that instead of returning the 2-approximation solution found in Algorithm 2 (line 8), we enumerate at this point all possible 2-approximation solutions using Lawler's procedure. If the number of 2-approximation solutions is at least k, then we can report the top-k packages found; otherwise, we continue accessing the next item.

In InsOpt-CR, the enumeration of all possible 2-approximation solutions is straightforward. Since we know the upper bound V^*, we can simply utilize Lawler's procedure to enumerate candidate packages which are under cost budget and have aggregated value of at least half of V^*.

LEMMA 3. *Given any instance I of top-k CompRec and any 2-approximation algorithm A with the same background cost information $\mathcal{BG}$ and access constraints as InsOpt-CR-Topk, A must read as many items as InsOpt-CR-Topk.*

THEOREM 2. *Let $\mathcal{I}$ be the class of all top-k CompRec instances, and $\mathcal{A}$ be the class of all possible 2-approximation algorithms that are constrained to access items in the non-increasing order of their value. Given the same background cost information $\mathcal{BG}$, InsOpt-CR-Topk is instance optimal over $\mathcal{A}$ and $\mathcal{I}$ with an optimality ratio of one.*

4.2 Greedy Algorithms

Although the instance optimal algorithms presented above guarantee to return top-k packages that are 2-approximations of the optimal packages, they rely on an exact algorithm for the knapsack problem which may lead to high computational cost. To remedy this, we propose more efficient algorithms next. Instead of using an exact algorithm to get the best package for the currently accessed set of items S, we use a simple greedy heuristic to form a high quality package R^G from S and then test whether R^G is globally a high quality package.

Compared with InsOpt-CR, our greedy solution Greedy-CR for top-1 CompRec needs to replace OptimalKP in InsOpt-CR with GreedyKP, which uses greedy heuristics [10] to find a high quality itemset in polynomial time [7], and to change R^o to the greedy solution R^G. Furthermore, instead of using tight upperbound calculated by MaxValBound, we need to use an untight heuristic upperbound which is calculated by the following algorithm MaxHeuristicValBound.

Algorithm 3: MaxHeuristicValBound(S, B, $\mathcal{BG}$)

```
1 τ ← v_min / c_min
2 Sort S = {t_1, ..., t_n} by value/cost ratio
3 m = max{m | v(t_m)/c(t_m) ≥ τ ∧ c(R_m) ≤ B}
4 R_m = {t_1, ..., t_m}
5 if m == n
6     V* = v(R_m) + τ * (B − c(R_m))
7 else
8     V* = v(R_m) + max{τ, v_{m+1}/c_{m+1}} * (B − c(R_m))
9 return V*
```

It follows from known results about knapsack that, similar to InsOpt-CR, Greedy-CR will always generate a correct 2-approximation to the optimal solution.

However, unlike InsOpt-CR, Greedy-CR is not instance optimal among all 2-approximation algorithms with the same constraints, as the following example shows.

EXAMPLE 2. *Let $I = \{t_1, t_2, \ldots, t_n\}$ be a top-1 CompRec instance, where $v(t_1) = v(t_2) = 101$, $c(t_1) = c(t_2) = 100$, for $i = 3, \ldots, 101$, $v(t_i) = c(t_i) = 1$, and for $i = 102, \ldots, n$, $v(t_i) = 1$ and $c(t_i) = 0.5$. Let $B = 199$, $\mathcal{BG} = c_{min} = 0.5$ and approximation ratio $\alpha = 2$. From Example 1, we know that after accessing the first 101 items, $S = \{t_1, \ldots, t_{101}\}$, $v(R^o) = 200$, $V^* = 398$ and InsOpt-CR will stop. However, at this moment $R^G = \{t_1\}$, and $v(R^G) = 101 < \frac{1}{2} \times V^*$. So Greedy-CR will continue accessing new items and it can be easily verified that Greedy-CR needs to access another 98 items before it stops.*

[7]Note that any approximation algorithm for knapsack [10] can be plugged in here while correctness of the resulting algorithm and the instance optimality result won't change.

We note that, in practice, cases such as the above may occur rarely. In fact, in our experimental results (Sec. 6) we observed that, on a range of datasets, Greedy-CR exhibited a very low running time while achieving similar access costs and overall result quality when compared to InsOpt-CR.

Similar to Section 4.1.2, we can easily extend Greedy-CR to Greedy-CR-Topk by using Lawler's procedure [14] to enumerate all possible high quality packages after one such package is identified. However, unlike InsOpt-CR-Topk which guarantees instance optimality, here we simply use Lawler's procedure to enumerate all candidate packages using the greedy algorithm instead of the exact algorithm. Similar to [11], we show in the following theorem that for top-k CompRec, if an α-approximation algorithm is utilized in Lawler's procedure instead of the exact algorithm which finds the optimal solution, we get an α-approximation to the top-k composite recommendations.

THEOREM 3. *Given an instance I of top-k CompRec, any α-approximation algorithm A for top-1 CompRec can be utilized with Lawler's procedure to generate a set $\mathcal{R}_k$ of composite recommendations which is an α-approximation to the optimal set of top-k composite recommendations.*

So the quality of the packages generated by the resulting enumeration process can be guaranteed. In this enumeration process, given a candidate package, we use the greedy algorithm to get the next candidate package for each sub-search space in Lawler's procedure, and if all of them are not guaranteed to be 2-approximations, the enumeration will stop.

However, similar to Greedy-CR, it is obvious that Greedy-CR-Topk is not instance optimal. We note that, in practice, the difference between the results generated by InsOpt-CR-Topk and Greedy-CR-Topk (in terms of the aggregate values of packages generated) may be very small.

5. DISCUSSION

As mentioned in Sec. 3, our framework includes the notion of a package satisfying compatibility constraints. E.g., for trip planning, the user may require the returned package to contain no more than 3 museums.

To capture these constraints in our algorithms, we can define a Boolean *compatibility function* $\mathcal{C}$ over the packages under consideration. Given a package P, $\mathcal{C}(P) = true$ iff all constraints on items in p are satisfied. We can add a call to $\mathcal{C}$ in InsOpt-CR-Topk and Greedy-CR-Topk after each candidate package has been found. If the package fails the compatibility check, we just discard it and search for the next candidate package. In terms of access cost, it can be easily verified that the modified InsOpt-CR-Topk algorithm is still instance optimal.

It is worth noting that the Boolean compatibility function defined here allows for greater generality than the constraints studied in previous work such as [3, 17]. However, depending on the application needs, for scenarios where only one specific type of constraint is considered, e.g., having one item from each of 3 predefined categories, more efficient algorithms like Rank Join [7] can be leveraged.

Furthermore, although in the previous algorithms we assume there is only one component recommender system, it is straightforward to combine recommendation lists from several component recommender systems by creating on-the-fly a "virtual recommendation list", e.g., select at each iteration the item which has the maximum value/rating across all recommender systems. The details will appear in the full version of the paper where efficient algorithms are given for special cases of compatibility constraints as well as compatibility constraints based on continuous functions.

6. EXPERIMENTS

In this section, we study the performance of our proposed algorithms based on both real and synthetic datasets.

6.1 Experimental Setup and Data Sets

The goal of our experiments were: (i) evaluate the relative quality of Inst-Opt-CR and Greedy-CR compared to the optimal algorithm, in terms of both the total and average values of the top-k packages returned, and (ii) evaluate the relative efficiency of the algorithms with respect to the number of items accessed and the actual run time. All experiments were done on a Xeon 2.5GHz Quad Core Windows Server 2003 machine with 16GB RAM and a 128GB SCSI hard disk. All code is in Java using JDK/JRE 1.6.

We use four datasets in our experiments. The first dataset is from MovieLens[8]. We use the 10 million rating MovieLens dataset which contains 1 million ratings for 3900 movies by 6040 users. In our experiments, we used the running time of movies, obtained from IMDB[9], as cost and we assume users are interested in packages of movies where the total running time is under a given budget.

TripAdvisor[10] is a well-known website where users can share and explore travel information. For our experiments, we crawled user rating information from places of interest (POIs) in the 10 most popular cities in the US. We exclude POIs which have one or no reviews, and the dataset contains 23658 ratings for 1393 POIs by 14562 users, so it is very sparse.[11] We associate with each POI in the dataset, a cost which is based on log(number of reviews) and scaled to the range of 1 to 50. The intuition of this cost function is that the more popular a POI is (in terms of number of reviews), the more likely it is to be crowded or the more likely it is for the tickets to be expensive. In practice, we may also use some existing work like [5] to mine from online user-generated itineraries other cost measures, e.g., average time users spent at each POI, average cost of visiting each POI, etc.

For the MovieLens and TripAdvisor datasets, we use a simple memory-based collaborative filtering algorithm [2][12] to generate predicted ratings for each user. The ratings are scaled and rounded to integers ranging from 1 to 50.

For the MovieLens dataset, we randomly selected 20 users from the 23594 user pool, and the budget for each user was fixed at 500 minutes[13]. For the TripAdvisor dataset, because of the sparsity of the underlying user rating matrix, we selected the 10 most active users as our sample for testing the algorithms, and set the user cost budget to 50.

[8] `http://www.movielens.org`

[9] `http://www.imdb.com`

[10] `http://www.tripadvisor.com`

[11] Pruning more aggressively rendered it too small.

[12] Our algorithms don't depend on a specific recommendation algorithm; in practice, our framework assumes ratings come from existing recommender systems.

[13] For all datasets, we tested our algorithms under various cost budgets with very similar results, so other budgets are omitted for lack of space.

		1st Package		2nd Package		3rd Package		4th Package		5st Package	
		SUM	AVG	SUM	AVG	SUM	AVG	SUM	AVG	SUM	AVG
MovieLens	Optimal	427	46.7	426	46.6	425	46.7	424	46.7	423	46.6
	InsOpt-CR-Topk	386	47.5	385	47.4	385	47.3	384	47.2	383	47.2
	Greedy-CR-Topk	384	47	381	47	380	46.8	379	46.7	379	46.7
TripAdvisor	Optimal	300	50	300	50	300	50	300	50	300	50
	InsOpt-CR-Topk	185	50	175	50	165	50	160	50	155	50
	Greedy-CR-Topk	220	50	210	50	210	50	205	50	205	50
Uncorrelated Data	Optimal	1092	36.4	1091	36.4	1090	36.3	1090	36.3	1089	36.5
	InsOpt-CR-Topk	929	43.6	926	43.6	925	43.6	925	43.6	924	43.5
	Greedy-CR-Topk	945	42.9	939	42.8	938	42.8	936	42.7	931	42.8
Correlated Data	Optimal	122	5.3	122	5.2	122	5.2	122	5.1	122	5.2
	InsOpt-CR-Topk	110	6.7	110	6.7	110	6.7	110	6.6	110	6.5
	Greedy-CR-Topk	110	6.6	110	6.6	109	7.6	109	6.5	109	7.15

Table 1: Quality Comparison for Different Composite Recommendation Algorithms

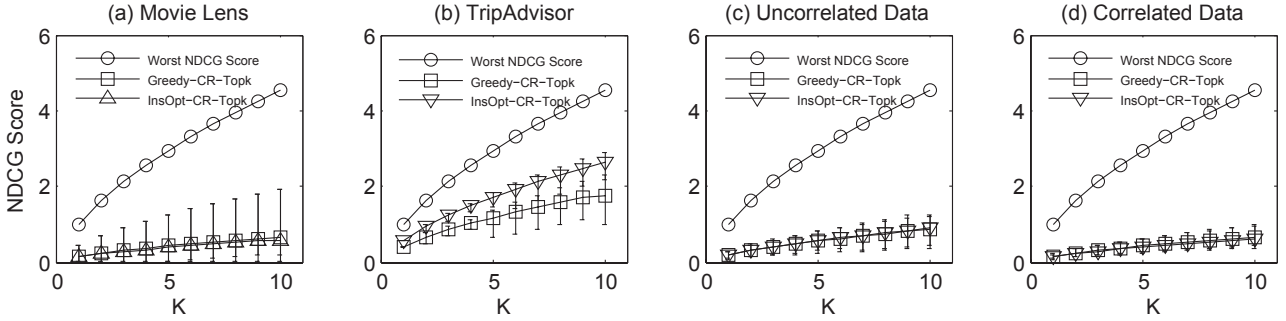

Figure 2: NDCG Score for Top-k Packages

We also tested our algorithms on synthetic correlated and uncorrelated datasets. For both datasets, item ratings are randomly chosen from 1 to 50. For the uncorrelated dataset, item costs are also randomly chosen from 1 to 50, but for the correlated dataset, the cost of item t is randomly chosen from $\min\{1, v(t)-5\}$ to $v(t)+5$. In both datasets, the total number of items is 1000, and the cost budget is set to 50. For all datasets, we assume the background cost information $\mathcal{BG}$ is simply the global minimum item cost.

6.2 Quality of Recommended Packages

For each dataset, Table 1 shows the quality of the top-5 composite recommendations returned by the optimal and approximation algorithms. We use as measures of quality the aggregated value of each package (SUM column) and the average item value of each package (AVG column).

It can be verified from Table 1 that our approximation algorithms do indeed return top-k composite packages whose value is guaranteed to be a 2-approximation of the optimal. Furthermore, from the average item value column, it is clear that our proposed approximation algorithms often recommend packages with high average value, whereas the optimal algorithm often tries to fill the package with small cost and small value items. So by sacrificing some of these lower quality items, the proposed approximation algorithms may manage to find high quality packages much more efficiently.

To better study the overall quality of returned packages, we also adopt a modified Normalized Discounted Cumulative Gain (NDCG) [9] to measure the quality of the top-k composite packages returned by the approximation algorithms against the optimal algorithm. Let $R^o = \{P_1^o, \ldots, P_k^o\}$ be the top-k packages returned by the optimal algorithm, and $R^a = \{P_1^a, \ldots, P_k^a\}$ be the top-k packages returned by the approximation algorithm. The modified NDCG score is a weighted sum of aggregated package value difference at each position of the returned top-k list, and is defined as:

$$NDCG(R^o, R^a) = \sum_{i=1}^{k} \frac{\log(1 + \frac{v(P_i^o) - v(P_i^a)}{v(P_i^o)})}{\log(1+i)}$$

The ideal value for the modified NDCG score is 0, where the top-k packages returned have exactly the same value as the optimal top-k packages. The worst possible value for the modified NDCG score is $\sum_{i=1}^{k} \frac{\log 2}{\log(1+i)}$, where each package returned has an aggregated value of 0. In Figure 2, we show for the 4 datasets the NDCG score of the top-k packages (k ranging over 1 to 10) returned by the instance optimal algorithm and the greedy algorithm. It is clear that, while having a substantial run time advantage, the greedy algorithm can achieve a very similar overall top-k package quality compared to the instance optimal algorithm. We also note that both approximation algorithms have a very small NDCG score.

6.3 Efficiency Study

The running times of our algorithms on the 4 datasets are shown in Figure 3 (a)-(d), while access costs are shown in Figure 3 (e)-(h). For MovieLens, TripAdvisor and the uncorrelated dataset, it can be seen that on average the greedy algorithm Greedy-CR-Topk has excellent performance in terms of *both* running time and access cost. The instance optimal algorithm InsOpt-CR-Topk also has low access cost, but its running time grows very quickly with k since it needs to solve exactly many instances of knapsack, restricted to the accessed items.

As can be seen in Figure 3 (h), the only dataset where both the greedy and instance optimal algorithms have a high access cost is the correlated dataset (but notice that the greedy algorithm still has good running time). The reason for this

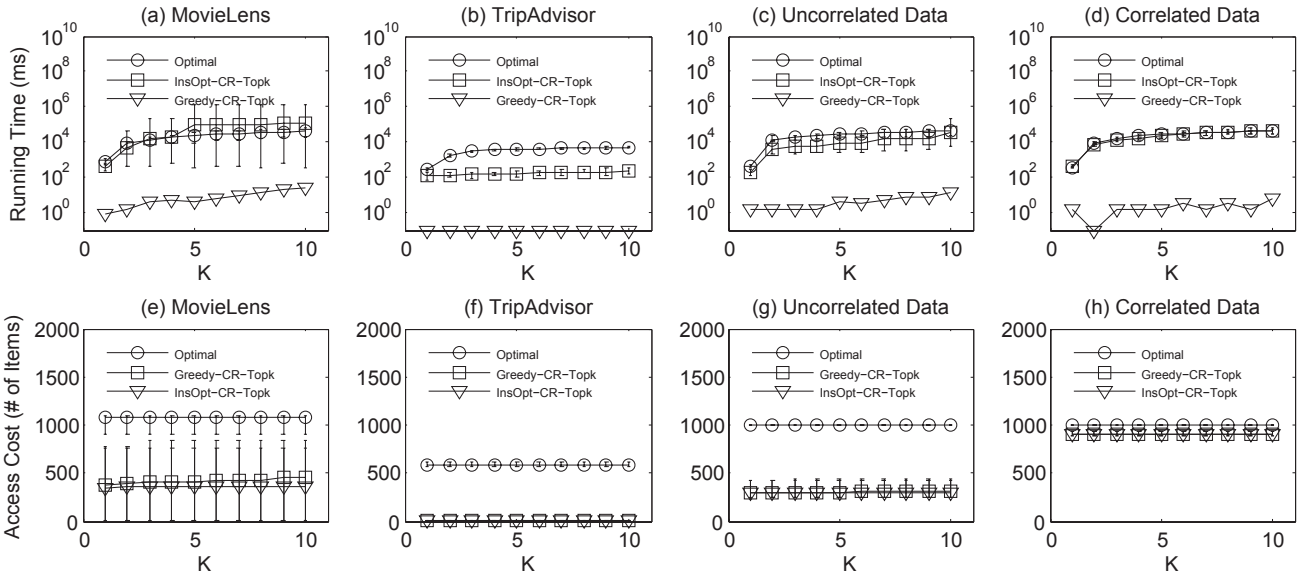

Figure 3: (a)–(d) Running Time for Different Datasets; (e)–(h) Access Cost for Different Datasets

is that, for the correlated dataset, the global minimum cost corresponds only to items which also have the least value. Thus the information it provides on the unseen items is very coarse. In practice, one solution to this might be to obtain more precise background cost information.

7. CONCLUSION

Motivated by applications in trip planning and in finding the most effective tweeters to follow, we studied the problem of recommending packages consisting of sets of items. Our composite recommender system has one or more component RecSys, which serve item recommendations in non-increasing order of their value. We proposed the problem of generating top-k package recommendations that are compatible and are under a cost budget, where a cost is incurred by visiting each recommended item and the budget and compatibility constraints are user specified. We focused on the case where there are no compatibility constraints and there is only one component RecSys. The problem is NP-complete since it is a variant of the Knapsack problem with the restriction that items need to be accessed in value-sorted order. So we developed two 2-approximation algorithms that are designed to minimize the number of items accessed. The first of these, InsOpt-CR-Topk, is instance optimal in a strong sense: every 2-approximation algorithm for the problem must access at least as many items as this algorithm. The second of these, Greedy-CR-Topk, is not guaranteed to be instance optimal, but is much faster. We experimentally evaluated the performance of the algorithms and showed that in terms of the quality of the top-k packages returned both algorithms are close to each other and deliver high quality packages; in terms of the number of items accessed Greedy-CR-Topk is very close to InsOpt-CR-Topk, but in terms of running time, Greedy-CR-Topk is much faster.

8. ACKNOWLEDGMENTS

Part of this work was done during Peter Wood's visit to UBC in January, 2010. This research was supported by a grant from NSERC (Canada).

9. REFERENCES

[1] http://www.cs.ubc.ca/~minxie/TR-2010-04.pdf.
[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE TKDE*, 17(6):734–749, 2005.
[3] A. Angel, S. Chaudhuri, G. Das, and N. Koudas. Ranking objects based on relationships and fixed associations. In *EDBT*, pages 910–921, 2009.
[4] A. Brodsky, S. M. Henshaw, and J. Whittle. CARD: a decision-guidance framework and application for recommending composite alternatives. In *RecSys*, pages 171–178, 2008.
[5] M. De Choudhury, M. Feldman, S. Amer-Yahia, N. Golbandi, R. Lempel, and C. Yu. Automatic construction of travel itineraries using social breadcrumbs. In *ACM Hypertext*, 2010.
[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.
[7] J. Finger and N. Polyzotis. Robust and efficient algorithms for rank join evaluation. In *ACM SIGMOD*, pages 415–428, 2009.
[8] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975.
[9] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.
[10] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.
[11] B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *PODS*, pages 173–182, 2006.
[12] G. Koutrika, B. Bercovitz, and H. Garcia-Molina. FlexRecs: expressing and combining flexible recommendations. In *SIGMOD*, pages 745–758, 2009.
[13] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, pages 467–476, 2009.
[14] E. L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Man. Sci.*, 18(7):401–405, 1972.
[15] A. Marchetti-Spaccamela and C. Vercellis. Stochastic on-line knapsack problems. *Math. Program.*, 68:73–104, 1995.
[16] A. Parameswaran and H. Garcia-Molina. Recommendations with prerequisites. In *ACM Recommender Systems*, pages 353–356, 2009.
[17] A. Parameswaran, P. Venetis, and H. Garcia-Molina. Recommendation systems with complex constraints: A CourseRank perspective. Technical report, 2009.
[18] S. B. Roy, S. Amer-Yahia, A. Chawla, G. Das, and C. Yu. Constructing and exploring composite items. In *ACM SIGMOD*, 2010.
[19] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.
[20] J. Weng, E.-P. Lim, J. Jiang, and Q. He. TwitterRank: finding topic-sensitive influential twitterers. In *WSDM*, pages 261–270, 2010.