On the complexity of package recommendation problems

Citation for published version:
Deng, T, Fan, W & Geerts, F 2012, On the complexity of package recommendation problems. in *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012.* ACM, pp. 261-272.
https://doi.org/10.1145/2213556.2213592

Digital Object Identifier (DOI):
10.1145/2213556.2213592

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012

On the Complexity of Package Recommendation Problems

Ting Deng
School of Computer Science
and Engineering
Beihang University
Beijing, China
dengting@act.buaa.edu.cn

Wenfei Fan
Lab. for Foundations of
Computer Science
School of Informatics
University of Edinburgh, UK
wenfei@inf.ed.ac.uk

Floris Geerts
Dept. of Mathematics &
Computer Science
University of Antwerp
Antwerp, Belgium
floris.geerts@ua.ac.be

ABSTRACT

Recommendation systems aim to recommend items that are likely to be of interest to users. This paper investigates several issues fundamental to such systems.

(1) We model recommendation systems for packages of items. We use queries to specify multi-criteria for item selections and express compatibility constraints on items in a package, and use functions to compute the cost and usefulness of items to a user.

(2) We study recommendations of points of interest, to suggest top-k packages. We also investigate recommendations of top-k items, as a special case. In addition, when sensible suggestions cannot be found, we propose query relaxation recommendations to help users revise their selection criteria, or adjustment recommendations to guide vendors to modify their item collections.

(3) We identify several problems, to decide whether a set of packages makes a top-k recommendation, whether a rating bound is maximum for selecting top-k packages, whether we can relax the selection query to find packages that users want, and whether we can update a bounded number of items such that the users' requirements can be satisfied. We also study function problems for computing top-k packages, and counting problems to find how many packages meet the user's criteria.

(4) We establish the upper and lower bounds of these problems, *all matching*, for combined and data complexity. These results reveal the impact of variable sizes of packages, the presence of compatibility constraints, as well as a variety of query languages for specifying selection criteria and compatibility constraints, on the analyses of these problems.

Categories and Subject Descriptors: H.3.5 [**Information Systems**]: Online Information Services – *Web-based services*; F.2.0 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity – *General*

Keywords: Recommendation problems, Complexity, Query relaxation.

PODS'12, May 21–23, 2012, Scottsdale, Arizona, USA.

1. INTRODUCTION

Recommendation systems, *a.k.a.* recommender systems, recommendation engines or platforms, aim to identify and suggest information items or social elements that are likely to be of interest to users. Traditional recommendation systems are to select top-k items from a collection of items, *e.g.,* books, music, news, Web sites and research papers [3], which satisfy certain criteria identified for a user, and are ranked by ratings with a utility function. More recently recommendation systems are often used to find top-k packages, *i.e., sets of items*, such as travel plans [34], teams of players [22] and various course combinations [19, 26, 27]. The items in a package are required not only to meet multi-criteria for selecting individual items, but also to satisfy compatibility constraints defined on all the items in a package taken together, such as team formation [22] and course prerequisites [26]. Packages may have variable sizes subject to a cost budget, and are ranked by overall ratings of their items [34].

Recommendation systems are increasingly becoming an integral part of Web services [34], Web search [4], social networks [4], education software [27] and commerce services [3]. A number of systems have been developed for recommending items or packages, known as *points of interest (POI)* [34] (see [3, 4] for surveys). These systems use relational queries to specify selection criteria and compatibility constraints [2, 7, 19, 27, 34]. There has also been work on the complexity of computing POI recommendations [22, 26, 27, 34].

However, to understand central issues associated with recommendation systems, there is much more to be done. (1) The previous complexity results were developed for individual applications with specific selection criteria and compatibility constraints. They may not carry over to other settings. This highlights the need for studying recommendation problems in a *uniform model.* (2) In most cases only lower bounds were given (NP-hard by *e.g.,* [26, 27]). Worse still, among the few upper bounds claimed, some are not quite correct. It is necessary to set the record straight by establishing *matching* upper and lower bounds. (3) No previous work has studied where high complexity arises. Is it from variable sizes of packages, compatibility constraints or from complex selection criteria? The need for understanding this is evident when developing practical recommendation systems. (4) In practice one often gets no sensible recommendations. When this happens, a system should be able to come up with recommendations for the users to *revise selection criteria*, or for vendors to *adjust their item collections*. However, no matter how important these issues are, no previous work has studied recommendations beyond POI.

Example 1.1: Consider a recommendation system for travel plans, which maintains two relations specified by:

flight(f#, From, To, DT, DD, AT, AD, Pr),
POI(name, city, type, ticket, time).

Here a flight tuple specifies flight f# from From to To that departs at time DT on date DD and arrives at time AT on date AD, with airfare Pr. A POI tuple specifies a place name to visit in the city, its ticket price, type (*e.g.,* museum, theater), and the amount of time needed for the visit.

(1) Recommendations of items. A user wants to find top-3 flights from EDI to NYC with at most one stop, departing on 1/1/2012, with lowest possible airfare and duration time. This can be stated as item recommendation: (a) flights are items; (b) the selection criteria are expressed as a union $Q_1 \cup Q_2$ of conjunctive queries, where Q_1 and Q_2 select direct and one-stop flights from EDI to NYC leaving on 1/1/2012, respectively; and (c) the items selected are ranked by a utility function $f()$: given an item s, $f(s)$ is a real number computed from the airfare Pr and the duration Dur of s such that the higher the Pr and Dur are, the lower the rating of s is. Here Dur can be derived from DT, DD, AT and AD, and $f()$ may associate different weights with Pr and Dur.

(2) Recommendations of packages. One is planing a 5-day holiday, by taking a direct flight from EDI to NYC departing on 1/1/2012 and visiting as many places in NYC as possible. In addition, she does not want to have more than 2 museums in a package, which is a compatibility constraint [34]. Moreover, she wants the plans to have the lowest overall price. This is an example of package recommendations: (a) the selection criteria are expressed as the following conjunctive query (CQ) Q, which finds pairs of flights and POI as items:

$$Q(\mathsf{f\#}, \mathsf{Pr}, \mathsf{name}, \mathsf{type}, \mathsf{ticket}, \mathsf{time}) = \exists\, \mathsf{DT}, \mathsf{AT}, \mathsf{AD}, x_{\mathsf{To}} \; \big(\mathsf{flight}(\mathsf{f\#}, \text{EDI}, x_{\mathsf{To}}, \mathsf{DT}, 1/1/2012, \mathsf{AT}, \mathsf{AD}, \mathsf{Pr}) \wedge \mathsf{POI}(\mathsf{name}, x_{\mathsf{To}}, \mathsf{type}, \mathsf{ticket}, \mathsf{time}) \wedge x_{\mathsf{To}} = \text{NYC}\big);$$

(b) a package N consists of some items that have the same f# (and hence Pr); (c) the rating of N, denoted by $\mathsf{val}(N)$, is a real number such that the higher the sum of the Pr and ticket prices of the items in N is, the lower $\mathsf{val}(N)$ is; (d) the compatibility constraint can be expressed as a CQ query Q_c such that $Q_c(N) = \emptyset$ iff the requirement is satisfied (see Section 2); and (e) the cost of N, denoted by $\mathsf{cost}(N)$, is the total time taken on visiting all POI in N. Note that the number of items in N is *not* fixed: N may contain as many POI as possible, as long as $\mathsf{cost}(N)$ does not exceed the total time allocated for sightseeing in 5 days. Putting these together, the travel planning is to find top-k such packages ranked by $\mathsf{val}(N)$, for a constant k chosen by the user.

(3) Computational complexity. To develop a recommendation system, one naturally wants to know the complexity for computing top-k packages or top-k items. The complexity may depend on what query language we use to specify selection criteria and compatibility constraints. For instance, in the package recommendation example given above, the criteria and constraints are expressed as CQ queries. Suppose that the user can bear with indirect flights with an unlimited number of stops. Then we need to express the selection criteria in, *e.g.,* DATALOG, which is more costly to evaluate than the CQ queries. What is the complexity of package recommendations when criteria and constraints are expressed in various languages? Will the complexity be lower if compatibility constraints are absent? Will it make our lives easier if we fix the size of each package? To the best of our knowledge, these questions have not been answered.

(4) Query relaxation recommendations. One may not get any direct flight from EDI to NYC. Nevertheless, if we relax the CQ Q given above by, *e.g.,* allowing To to be a city within 15 miles of NYC, then direct flights are available, *e.g.,* from EDI to EWR. This suggests that we help the user revise her selection criteria by recommending query relaxations.

(5) Adjustment recommendations. The collection of POI in the system may consist of museums only, which users may not want to see too many, as indicated by the compatibility constraint Q_c above. This motivates us to study adjustment recommendations, by recommending the vendor of the system to include, *e.g.,* theaters, in their POI collection. □

These highlight the need for a full treatment of recommendation problems, to study them in a generic model, establish their matching upper and lower bounds, and identify where the complexity arises. Moreover, analogous to POI recommendations, query relaxation recommendations and adjustment recommendations should logically be part of a practical system, and hence, deserve to be investigated.

A model for package recommendations. Following [2, 7, 19, 26, 27, 34] we consider a database D that includes items in a recommendation system. We specify (a) multi-criteria for selecting items as a relational query Q; (b) compatibility constraints on the items in a package N as another query Q_c such that $Q_c(N, D) = \emptyset$ iff N satisfies the constraints; (c) a rating function $\mathsf{val}()$ from packages to real numbers $\mathbb{R}$ such that $\mathsf{val}(N)$ assesses the usefulness of a package N to a user; and (d) a cost budget C and a function $\mathsf{cost}()$ from packages to $\mathbb{R}$ such that a package N is a "valid" choice iff $\mathsf{cost}(N) \leq C$. Given a constant k, package recommendation is to find top-k packages based on $\mathsf{val}()$ such that each package consists of items selected by Q and satisfies the constraints Q_c. As shown in Example 1.1, packages may have *variable sizes*: we want to maximize $\mathsf{val}(N)$ as long as $\mathsf{cost}(N)$ does not exceed the budget C.

Traditional item recommendations are a special case. We use a utility function $f()$ that gives a rating in $\mathbb{R}$ to each tuple in $Q(D)$. For a given k, it is to find top-k items that meet the criteria specified by Q, ranked by the function $f()$.

This yields a model for top-k package querying that subsumes previous models studied for, *e.g.,* travel and course recommendations. We study recommendation problems in a generic setting when selection criteria and compatibility constraints are expressed as queries, and when $\mathsf{cost}(), \mathsf{val}()$ and $f()$ are only assumed to be computable in PTIME.

Recommendation problems. We identify several problems for POI recommendations. (a) Decision problems: Is a set of packages a top-k recommendation? Is a constant B the largest bound such that there exists a top-k recommendation in which each package is rated above B? (b) Function problem: find a top-k recommendation if there exists one. (c) Counting problem: how many valid packages are there that have ratings above a bound B?

Beyond POI recommendations, we propose to study the following features that future recommendater systems could support. (a) Query relaxation recommendations: Can we find a "minimum" relaxation of the users' selection criteria Q to allow a top-k recommendation? (b) Adjustment recommendations: Can we update a bounded number of items in D such that the users' requirements can be satisfied? We parameterize each of these problems with various query languages $\mathcal{L}_Q$ in which selection criteria Q and compatibility constraints Q_c are expressed. We consider the following $\mathcal{L}_Q$, all with built-in predicates $=, \neq, <, \leq, >, \geq$:

- conjunctive queries (CQ),
- union of conjunctive queries (UCQ),
- positive existential FO queries ($\exists\text{FO}^+$),
- nonrecursive datalog queries ($\text{DATALOG}_{\mathsf{nr}}$),
- first-order queries (FO), and
- datalog (DATALOG).

Complexity results. For all these problems, we establish its combined complexity and data complexity. We also study special cases of package recommendations, such as when compatibility constraints are absent, when packages have a fixed size, and when both conditions are imposed (item recommendations). We provide their upper and lower bounds, *all matching*, for all the query languages given above.

These results give a complete characterization of the complexity in this model, from decision problems to function and counting problems. They tell us where complexity arises, complementing previously stated results.

(a) Query languages dominate the complexity of recommendation problems, *e.g.,* the problem for deciding the maximum bound for top-k package recommendations ranges from $\mathsf{D}_2^{\mathsf{p}}$-complete for CQ, PSPACE-complete for FO and $\text{DATALOG}_{\mathsf{nr}}$, to EXPTIME-complete for DATALOG.

(b) Variable package sizes do not make our lives harder when combined complexity is concerned for all the languages given above. Indeed, when packages may have variable sizes, all these problems have the same combined complexity as their counterparts when packages are restricted to be singleton sets. In fact, variable sizes of packages have impact only on data complexity, or when $\mathcal{L}_Q$ is a simple language with a PTIME complexity for its membership problem. These clarify the impact of package sizes studied in, *e.g.,* [34].

(c) The presence of compatibility constraints does not increase the complexity when the query language $\mathcal{L}_Q$ is FO, $\text{DATALOG}_{\mathsf{nr}}$ or DATALOG. Indeed, for these languages, all the problems for package recommendations and their counterparts for item recommendations have the same complexity. Moreover, these constraints do *not* complicate the data complexity analyses. However, compatibility constraints increase combined complexity when $\mathcal{L}_Q$ is contained in $\exists\text{FO}^+$.

(d) In the absence of compatibility constraints, the decision problem for top-k package recommendations is DP-complete and its function problem is $\mathsf{FP}^{\mathsf{NP}}$-complete when $\mathcal{L}_Q$ is CQ. They are coNP-hard and $\mathsf{FP}^{\mathsf{NP}}$-hard, respectively, even when selection criteria are given by an identity query. These give precise bounds for the problems studied in, *e.g.,* [34].

These results are also of interest to the study of top-k query answering, among other things. A variety of techniques are used to prove the results, including a wide range of reductions, and constructive proofs with algorithms (*e.g.,* for the function problems). In particular, the proofs demonstrate that the complexity of these problems for CQ, UCQ and $\exists\text{FO}^+$is inherent to top-k package querying itself, rather than a consequence of the complexity of the query languages.

Related work. Traditional recommendation systems aim to find, for each user, items that maximize the user's utility (see, *e.g.,* [3] for a survey). Selection criteria are decided by content-based, collaborative and hybrid approaches, which consider preferences of each user in isolation, or preferences of similar users [3]. The prior work has mostly focused on how to choose appropriate utility functions, and how to extrapolate such functions when they are not defined on the entire item space, by deriving unknown values from known ones. Our model supports content-based, collaborative and hybrid criteria in terms of various queries. We *assume a given utility function* that is total, and focus on the computational complexity of recommendation problems.

Recently recommendation systems have been extended to finding packages, which are presented to the user in a ranked order based on some rating function [6, 22, 26, 27, 34]. A number of algorithms have been developed for recommending packages of a fixed size [6, 22] or variable sizes [26, 27, 34]. Compatibility constraints [22, 26, 27, 34] and budget restrictions [34] on packages have also been studied. Instead of considering domain-specific applications, we model recommendations of both items and packages (fixed size or polynomial size) by specifying general selection criteria and compatibility constraints as queries, and supporting aggregate constraints defined with cost budgets and rating bounds.

Several decision problems for course package recommendations have been shown NP-hard [26, 27]. It was claimed that problems of forming a team with compatibility constraints [22] and the problem of finding packages that satisfy some budget restrictions (without compatibility constraints) [34] are NP-complete. In contrast, we establish the precise complexity of a variety of problems associated with POI recommendations (Table 1, Section 7). Moreover, we provide the complexity of query relaxation and adjustment recommendations, which have not been studied by prior work.

There has also been a host of work on recommending items and packages taken from views of the data [2, 7, 19, 23, 34]. Such views are expressed as relational queries, representing preferences or points of interest [2, 7, 19]. Here recommendations often correspond to top-k query answers. Indeed, top-k query answering retrieves the k-items (tuples) from a query result that are top-ranked by some scoring function [16]. Such queries either simply select tuples, or join and aggregate multiple inputs to find the top-k tuples, by possibly incorporating user preference information [19, 29]. A number of top-k query evaluation algorithms have been developed (*e.g.,* [12, 23, 28]; see [16] for a survey), as well as algorithms for incremental computation of ranked query results [10, 14, 24] that retrieve the top-k query answers one at a time. A central issue there concerns how to combine different ratings of the same item based on multiple criteria. Our work also retrieves tuples from the result of a query. It differs from the previous work in the following. (1) In contrast to top-k query answering, we are to find items and sets of items (packages) provided that a utility or rating function is given. (2) We focus on the complexity of recommendations problems rather than the efficiency or optimization of query evaluation. (3) Beyond recommendations of POI, we also study query relaxation and adjustment recommendations.

Query relaxations have been studied in, *e.g.,* [8, 13, 17, 18]. Several query generalization rules are introduced in [8], assuming that query acceptance conditions are monotonic. Heuristic query relaxation algorithms are developed in [13, 17]. The topic is also studied for top-k query answering [18]. We focus on the main idea of query relaxation recommendations, and borrow query generalization rules from [8]. We consider acceptance conditions (*i.e.,* rating functions, compatibility constraints and aggregate constraints) that are not necessarily monotonic. Moreover, none of the previous work supports queries beyond CQ, while we consider more powerful languages such as FO and DATALOG. In addition, the prior work focuses on the design of efficient relaxation algorithms, but does not study computational complexity.

Organization. Section 2 introduces the model for package recommendations. Section 3 formulates and studies fundamental problems in connection with POI recommendations, followed by special cases in Section 4. Query relaxation recommendations are studied in Section 5, followed by adjustment recommendations in Section 6. Section 7 summarizes the main results of the paper and identifies open issues.

2. MODELING RECOMMENDATIONS

We first specify recommendations of packages and items. We then review query languages considered in this work.

Item collections. Following [2, 7, 19, 26, 27, 34], we assume a database D consisting of items for selection. The database is specified with a relational schema $\mathcal{R}$, with a collection of relation schemas $(R_1, \ldots, R_n)$. Each schema R_i is defined over a fixed set of attributes. For each attribute A in R_i, its domain is specified in R_i, denoted as $\mathsf{dom}(R_i.A)$.

Package recommendations. As remarked earlier, in practice one often wants packages of items, *e.g.,* combinations of courses to be taken to satisfy the requirements for a degree [27], travel plans including multiple POI [34], and teams of experts [22]. Package recommendation is to find top-k packages such that the items in each package (a) meet the selection criteria, (b) satisfy some compatibility constraints, *i.e.,* they have no conflicts, and moreover, (c) their ratings and costs satisfy certain aggregate constraints. To specify these, we extend the models proposed in [27, 34] as follows.

Selection criteria. We use a query Q in a query language $\mathcal{L}_Q$ to specify multi-criteria for selecting items from D. For instance, as shown in Example 1.1, we use a query to specify what flights and sites a user wants to find.

Compatibility constraints. To specify the compatibility constraints for a package N, we use a query Q_c such that N satisfies Q_c iff $Q_c(N, D) = \emptyset$. That is, Q_c identifies inconsistencies among items in N. In Example 1.1, to assert "no more than 2 museums" in a travel package N [34], we use the following Q_c that selects 3 distinct museums from N:

$$\begin{aligned} Q_c() = \exists\ & \mathsf{f\#}, \mathsf{Pr}, n_1, t_1, p_1, n_2, t_2, p_2, n_3, t_3, p_3 \\ & \big(R_Q(\mathsf{f\#},\ \mathsf{Pr},\ n_1,\ \mathit{museum},\ p_1,\ t_1) \wedge \\ & \ R_Q(\mathsf{f\#},\ \mathsf{Pr},\ n_2,\ \mathit{museum},\ p_2,\ t_2) \wedge \\ & \ R_Q(\mathsf{f\#},\ \mathsf{Pr},\ n_3,\ \mathit{museum},\ p_3,\ t_3) \wedge \\ & \qquad n_1 \neq n_2 \wedge n_1 \neq n_3 \wedge n_2 \neq n_3\big), \end{aligned}$$

where R_Q denotes the schema of the query answer $Q(D)$. As another example, for a course package N, we use a query Q_c to assure that for each course in N, its prerequisites are also included in N [26]. This query needs to access not only courses in N but also the prerequisite relation stored in D.

To simplify the discussion, we assume that query Q_c for specifying compatibility constraints and query Q for specifying selection criteria are in the same language $\mathcal{L}_Q$. If a system supports compatibility constraints in $\mathcal{L}_Q$, there is no reason for not supporting queries in the same language for selecting items. We defer to future work the study in the setting when Q_c and Q are expressed in different languages. Note that queries in various query languages are capable of expressing compatibility constraints commonly found in practice, including those studied in [19, 22, 26, 27, 34].

Aggregate constraints. To specify aggregate constraints, we define a cost function and a rating function over packages, following [34]: (1) $\mathsf{cost}(N)$ computes a value in $\mathbb{R}$ as the cost of package N; and (2) $\mathsf{val}(N)$ computes a value in $\mathbb{R}$ as the overall rating of N. For instance, $\mathsf{cost}(N)$ in Example 1.1 is computed from the total time taken for visiting POI, while $\mathsf{val}(N)$ is defined in terms of airfare and total ticket prices.

We just assume that $\mathsf{cost}()$ and $\mathsf{val}()$ are PTIME computable aggregate functions, defined in terms of *e.g.,* max, min, sum, avg, as commonly found in practice.

We also assume a cost budget C, and specify an aggregate constraint $\mathsf{cost}(N) \leq C$. For instance, the cost budget C in Example 1.1 is the total time allowed for visiting POI in 5 days, and the aggregate constraint $\mathsf{cost}(N) \leq C$ imposes a bound on the number of POI in a package N.

Top-k package selections. For a database D, queries Q and Q_c in $\mathcal{L}_Q$, a natural number $k \geq 1$, a cost budget C, and functions $\mathsf{cost}()$ and $\mathsf{val}()$, a *top-k package selection* is a set $\mathcal{N} = \{N_i \mid i \in [1, k]\}$ of packages such that for each $i \in [1, k]$,

(1) $N_i \subseteq Q(D)$, *i.e.,* its items meet the criteria given in Q;

(2) $Q_c(N_i, D) = \emptyset$, *i.e.,* the items in the package satisfy the compatibility constraints specified by query Q_c;

(3) $\mathsf{cost}(N_i) \leq C$, *i.e.,* its cost is below the budget;

(4) the number $|N_i|$ of items in N_i is no larger than $p(|D|)$, where p is a predefined polynomial and $|D|$ is the size of D; indeed, it is not of much practical use to find a package with exponentially many items; as will be seen in Section 4, we shall also consider a constant bound B_p for $|N_i|$;

(5) for all packages $N' \notin \mathcal{N}$ that satisfies conditions (1–4) given above, $\mathsf{val}(N') \leq \mathsf{val}(N_i)$, *i.e.,* packages in $\mathcal{N}$ have the k highest overall ratings among all feasible packages; and

(6) $N_i \neq N_j$ if $i \neq j$, *i.e.,* the packages are pairwise distinct.

Note that packages in $\mathcal{N}$ may have *variable* sizes. That is, the number of items in each package is not bounded by a constant. We just require that N_i satisfies the constraint $\mathsf{cost}(N_i) \leq C$ and $|N_i|$ does not exceed a polynomial in $|D|$.

Package recommendation is to find a *top-k package selection* for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$, if there exists one.

As shown in Example 1.1, users may want to find, *e.g.,* a top-k travel-plan selection with the minimum price.

Item recommendations. To rank items, we use a *utility function* $f()$ to measure the usefulness of items selected by $Q(D)$ to a user [3]. It is a PTIME-computable function that takes a tuple s from $Q(D)$ and returns a real number $f(s)$ as the rating of item s. The functions may incorporate users' preference [29], and may be *different for different users*.

Given a constant $k \geq 1$, a *top-k selection for* (Q, D, f) is a set $S = \{s_i \mid i \in [1, k]\}$ such that (a) $S \subseteq Q(D)$, *i.e.,* items in S satisfy the criteria specified by Q; (b) for all $s \in Q(D) \setminus S$ and $i \in [1, k]$, $f(s) \leq f(s_i)$, *i.e.,* items in S have the highest ratings; and (c) $s_i \neq s_j$ if $i \neq j$, *i.e.,* items in S are *distinct*.

Given D, Q, f and k, *item recommendation* is to find a top-k selection for (Q, D, f) if there exists one.

For instance, a top-3 item selection is described in Example 1.1, where items are flights and the utility function $f()$ is defined in terms of the airfare and duration of each flight.

The connection between item and package selections. Item selections are a special case of package selections. Indeed, a top-k selection $S=\{s_i \mid i\in[1,k]\}$ for (Q,D,f) is a top-k package selection $\mathcal{N}$ for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C)$, where $\mathcal{N}=\{N_i \mid i\in[1,k]\}$, and for each $i\in[1,k]$, (a) $N_i=\{s_i\}$, (b) Q_c is a constant query that returns $\emptyset$ on any input, referred to as the empty query; (c) $\mathsf{cost}(N_i)=|N_i|$ if $N_i\neq\emptyset$, and $\mathsf{cost}(\emptyset)=\infty$; that is, $\mathsf{cost}(N_i)$ counts the number of items in N_i if $N_i\neq\emptyset$, and the empty set is not taken as a recommendation; (d) the cost budget $C=1$, and hence, N_i consists of a single item by $\mathsf{cost}(N_i)\leq C$; and (e) $\mathsf{val}(N_i)=f(s_i)$.

In the sequel, we use top-k package selection *specified in terms of* (Q,D,f) to refer to a top-k selection S for (Q,D,f), *i.e.,* a top-k package selection for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C)$ in which $Q_c,\mathsf{cost}(),\mathsf{val}()$ and C are defined as above.

We say that compatibility constraints are *absent* if Q_c is the empty query; *e.g.,* Q_c is absent in item selections.

One might want to consider general PTIME compatibility constraints Q_c. As will be seen in Section 4, the complexity when Q_c is in PTIME remains the same as its counterpart when Q_c is absent for all the problems studied in this paper.

Query languages. We consider Q,Q_c in a query language $\mathcal{L}_Q$, ranging over the following (see *e.g.,* [1] for details):

(a) conjunctive queries (CQ), built up from atomic formulas with constants and variables, *i.e.,* relation atoms in database schema $\mathcal{R}$ and built-in predicates $(=,\neq,<,\leq,>,\geq)$, by closing under conjunction $\wedge$ and existential quantification $\exists$;

(b) union of conjunctive queries (UCQ) of the form $Q_1\cup\cdots\cup Q_r$, where for each $i\in[1,r]$, Q_i is in CQ;

(c) positive existential FO queries ($\exists$FO$^+$), built from atomic formulas by closing under $\wedge$, *disjunction* $\vee$ and $\exists$;

(d) nonrecursive datalog queries (DATALOG$_{\mathsf{nr}}$), defined as a collection of rules of the form $p(\bar{x})\leftarrow p_1(\bar{x}_1),\ldots,p_n(\bar{x}_n)$, where the head p is an IDB predicate and each p_i is either an atomic formula or an IDB predicate, such that its dependency graph is acyclic; the *dependency graph* of a DATALOG query Q is a directed graph $G_Q=(V,E)$, where V includes all the predicates of Q, and (p',p) is an edge in E iff p' is a predicate that appears in a rule with p as its head [9];

(e) first-order logic queries (FO) built from atomic formulas using $\wedge$, $\vee$, *negation* $\neg$, $\exists$ and universal quantification $\forall$; and

(f) datalog queries (DATALOG), defined as a collection of rules $p(\bar{x})\leftarrow p_1(\bar{x}_1),\ldots,p_n(\bar{x}_n)$, for which the dependency graph may possibly be cyclic, *i.e.,* DATALOG is an extension of DATALOG$_{\mathsf{nr}}$ with an inflational fixpoint operator.

These languages specify both multi-criteria for item selections and compatibility constraints for package selections.

3. RECOMMENDATIONS OF POI'S

In this section we investigate POI recommendations. We identify four problems for package recommendations (Section 3.1), and establish their complexity (Section 3.2).

3.1 Recommendation Problems

We investigate four problems, stated as follows, which are fundamental to computing package recommendations. We start with a decision problem for package selections. Consider a database D, queries Q and Q_c in a query language $\mathcal{L}_Q$, functions $\mathsf{val}()$ and $\mathsf{cost}()$, a cost budget C, and a natural number $k\geq 1$. Given a set $\mathcal{N}$ consisting of k packages, it is to decide whether $\mathcal{N}$ makes a top-k package selection. That is, each package N in $\mathcal{N}$ satisfies the selection criteria Q, compatibility constraint Q_c, and aggregate constraints $\mathsf{cost}(N)\leq C$ and $\mathsf{val}(N)\geq\mathsf{val}(N')$ for all $N'\notin\mathcal{N}$. As remarked earlier, we assume a predefined polynomial such that $|N|\leq p(|D|)$ (omitted from the problem statement below for simplicity). Intuitively, this problem is to decide whether a set $\mathcal{N}$ of packages should be recommended.

$\mathsf{RPP}(\mathcal{L}_Q)$:	*The recommendation problem (package).*
INPUT:	A database D, two queries Q and Q_c in $\mathcal{L}_Q$, two functions $\mathsf{cost}()$ and $\mathsf{val}()$, natural numbers C and $k\geq 1$, and a set $\mathcal{N}=\{N_i \mid i\in[1,k]\}$.
QUESTION:	Is $\mathcal{N}$ a top-k package selection for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C)$?

After all, recommendation systems have to compute top-k packages, rather than expecting that candidate selections are already in place. This highlights the need for studying the function problem below, to compute top-k packages.

$\mathsf{FRP}(\mathcal{L}_Q)$:	*The function rec. problem (packages).*
INPUT:	$D,Q,Q_c,\mathsf{cost}(),\mathsf{val}(),C,k$ as in RPP.
OUTPUT:	A top-k package selection for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C)$ if it exists.

The next question concerns how to find a maximum rating bound for computing top-k packages. We say that a constant B is a *rating bound* for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C,k)$ if (a) there exists a top-k package selection $\mathcal{N}=\{N_i \mid i\in[1,k]\}$ for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C)$ and moreover, (b) $\mathsf{val}(N_i)\geq B$ for each $i\in[1,k]$. That is, B allows a top-k package selection. We say that B is *the maximum bound for packages* with $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C,k)$ if for all bounds B', $B\geq B'$. Obviously B is unique if it exists. Intuitively, when B is identified, we can capitalize on B to compute top-rated packages. Furthermore, vendors could decide, *e.g.,* price for certain items on sale with such a bound, for risk assessment.

$\mathsf{MBP}(\mathcal{L}_Q)$:	*The maximum bound problem (packages).*
INPUT:	$D,Q,Q_c,\mathsf{cost}(),\mathsf{val}(),C,k$ as in RPP, and a natural number B.
QUESTION:	Is B the maximum bound for packages with $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C,k)$?

A package N is *valid* for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C,B)$ if (a) $N\subseteq Q(D)$, (b) $Q_c(N,D)=\emptyset$, (c) $\mathsf{cost}(N)\leq C$, and (d) $\mathsf{val}(N)\geq B$, where $|N|$ is bounded by a polynomial in $|D|$. Given B, one naturally wants to know how many valid packages are out there, and hence, can be selected. This suggests that we study the following counting problem.

$\mathsf{CPP}(\mathcal{L}_Q)$:	*The counting problem (packages).*
INPUT:	$D,Q,Q_c,\mathsf{cost}(),\mathsf{val}(),C,B$ as in MBP.
OUTPUT:	The number of packages that are valid for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C,B)$.

3.2 Deciding, Finding and Counting Top-k Packages

We now establish the complexity of $\mathsf{RPP}(\mathcal{L}_Q)$, $\mathsf{FRP}(\mathcal{L}_Q)$, $\mathsf{MBP}(\mathcal{L}_Q)$ and $\mathsf{CPP}(\mathcal{L}_Q)$, including their (1) combined complexity, when the query Q, compatibility constraint Q_c and database D may vary, and (2) data complexity, when only D varies, while Q and Q_c are predefined and fixed. We study these problems for all the query languages $\mathcal{L}_Q$ of Section 2.

Deciding package selections. We start with $\mathsf{RPP}(\mathcal{L}_Q)$. The result below tells us that the combined complexity of the problem is mostly determined by what query language $\mathcal{L}_Q$ we use to specify selection criteria and compatibility constraints. Indeed, it is Π_2^p-complete when $\mathcal{L}_Q$ is CQ, PSPACE-complete for DATALOG$_{\mathsf{nr}}$ and FO, and it becomes EXPTIME-complete when $\mathcal{L}_Q$ is DATALOG. The data complexity is coNP-complete for all the languages considered.

Theorem 3.1: For $\mathsf{RPP}(\mathcal{L}_Q)$, the combined complexity is

- Π_2^p-complete when $\mathcal{L}_Q$ is CQ, UCQ, or $\exists$FO$^+$;
- PSPACE-complete when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO; and
- EXPTIME-complete when $\mathcal{L}_Q$ is DATALOG.

The data complexity is coNP-complete for all the languages presented in Section 2, *i.e.,* when $\mathcal{L}_Q$ is CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$, FO or DATALOG. □

Proof sketch: (1) We show that RPP is Π_2^p-hard for CQ by reduction from the complement of *the compatibility problem*. The latter is to decide whether there exists a valid package N with $\mathsf{val}(N) > B$ for some bound B. We show that the compatibility problem is Σ_2^p-complete for CQ by reduction from the $\exists^*\forall^*$3DNF problem, which is Σ_2^p-complete [30].

For $\mathsf{RPP}(\exists\mathrm{FO}^+)$, we give a Π_2^p algorithm that first tests whether a given set $\mathcal{N}$ of packages satisfies the criteria and constraints, in DP; it then checks whether there exists *no* package with a higher rating than some $N \in \mathcal{N}$, in Π_2^p.

(2) We show that RPP is PSPACE-hard for DATALOG$_{\mathsf{nr}}$ by reduction from Q3SAT (cf. [25]), and for FO by reduction from the membership problem for FO ("given a query Q, a database D and a tuple t, whether $t \in Q(D)$") [32], which are PSPACE-complete. We provide an NPSPACE (= PSPACE) algorithm to check RPP for these two languages.

(3) For DATALOG, we show that RPP is EXPTIME-hard by reduction from the membership problem for DATALOG, which is EXPTIME-complete [32]. For the upper bound, we give an EXPTIME algorithm to check $\mathsf{RPP}(\mathrm{DATALOG})$.

(4) For the data complexity, we first show that the compatibility problem is NP-complete when Q and Q_c are fixed CQ queries, by reduction from 3SAT, an NP-complete problem (cf. [25]). From this it follows that $\mathsf{RPP}(\mathrm{CQ})$ is already coNP-hard. We also give a coNP algorithm for RPP when Q and Q_c are fixed queries in either FO or DATALOG. □

One might think that the absence of compatibility constraints Q_c would make our lives easier. Indeed, $\mathsf{RPP}(\mathrm{CQ})$ becomes DP-complete in the absence of Q_c, as opposed to Π_2^p-complete in the presence of Q_c. However, when $\mathcal{L}_Q$ is powerful enough to express FO or DATALOG$_{\mathsf{nr}}$ queries, dropping Q_c does not help: $\mathsf{RPP}(\mathcal{L}_Q)$ in this case has the same complexity as its counterpart when Q_c is present.

Theorem 3.2: In the absence of Q_c, $\mathsf{RPP}(\mathcal{L}_Q)$ is

- DP-complete when $\mathcal{L}_Q$ is CQ, UCQ, or $\exists$FO$^+$;
- PSPACE-complete when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO; and
- EXPTIME-complete when $\mathcal{L}_Q$ is DATALOG.

Its data complexity remains coNP-complete for all the query languages given in Section 2. □

Proof sketch: (1) We show that $\mathsf{RPP}(\mathrm{CQ})$ is DP-hard by reduction from SAT-UNSAT. The latter is to decide, given a pair (φ_1, φ_2) of 3SAT instances, whether φ_1 is satisfiable and φ_2 is not satisfiable. It is DP-complete (cf. [25]).

In the absence of Q_c, the algorithm given earlier for $\mathsf{RPP}(\exists\mathrm{FO}^+)$ is in DP, and hence so is $\mathsf{RPP}(\exists\mathrm{FO}^+)$.

(2-4) The lower bound proofs (2-4) of Theorem 3.1 do not use compatibility constraints and hence remain intact. The upper bounds obviously carry over to this special case. □

Computing top-k packages. We give the complexity of the function problem $\mathsf{FRP}(\mathcal{L}_Q)$ as follows:

Theorem 3.3: For $\mathsf{FRP}(\mathcal{L}_Q)$, the combined complexity is

- $\mathsf{FP}^{\Sigma_2^p}$-complete when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$;
- FPSPACE(poly)-complete if $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO;
- FEXPTIME(poly)-complete when $\mathcal{L}_Q$ is DATALOG.

In the absence of compatibility constraints, its combined complexity remains unchanged for DATALOG$_{\mathsf{nr}}$, FO and DATALOG, but it is $\mathsf{FP}^{\mathsf{NP}}$-complete for CQ, UCQ and $\exists$FO$^+$.

Its data complexity is $\mathsf{FP}^{\mathsf{NP}}$-complete for all the languages, in the presence or absence of compatibility constraints. □

Here $\mathsf{FP}^{\mathsf{NP}}$ is the class of all functions from strings to strings that can be computed by a PTIME Turing machine with an NP oracle (cf. [25]), and $\mathsf{FP}^{\Sigma_2^p}$ is the class of all functions computable by a PTIME 2-alternating max-min Turing machine [20]. By FPSPACE(poly) (resp. FEXPTIME(poly)) we mean the class of all functions associated with a two-argument predicate R_L that satisfies the following conditions: (a) R_L is *polynomially balanced, i.e.,* there is a polynomial q such that for all strings x and y, if $R_L(x,y)$ then $|y| \leq q(|x|)$, and (b) the decision problem "given x and y, whether $R_L(x,y)$" is in PSPACE (resp. EXPTIME) [21]. Given a string x, the function associated with R_L is to find a string y such that $R_L(x,y)$ if such a string exists.

These results tell us that it is nontrivial to find top-k packages. Indeed, to express compatibility constraints on travel plans given in [34], we need at least CQ; for course combination constraints of [19, 26, 27], we need FO; and for connectivity of flights we need DATALOG. These place FRP in $\mathsf{FP}^{\Sigma_2^p}$, FPSPACE(poly) and FEXPTIME(poly), respectively.

It was claimed in several earlier papers that when $k=1$, it is NP-complete to find a top-1 package. Unfortunately, it is not the case. Indeed, the proofs of Theorems 3.1, 3.2 and 3.3 tell us that when $k=1$, the function problem $\mathsf{FRP}(\mathcal{L}_Q)$ remains $\mathsf{FP}^{\Sigma_2^p}$-complete and the decision problem $\mathsf{RPP}(\mathcal{L}_Q)$ is Π_2^p-complete even when $\mathcal{L}_Q$ is CQ, not to mention more expressive $\mathcal{L}_Q$. Even when Q and Q_c are both fixed, FRP is $\mathsf{FP}^{\mathsf{NP}}$-complete and RPP is coNP-complete when $k=1$.

In the absence of compatibility constraints, only the analyses of the combined complexity of FRP for CQ, UCQ and $\exists$FO$^+$are simplified. This is consistent with Theorem 3.2.

Proof sketch: (1) We show that $\mathsf{FRP}(\mathrm{CQ})$ is $\mathsf{FP}^{\Sigma_2^p}$-hard by reduction from the MAXIMUM Σ_2^p problem, which is $\mathsf{FP}^{\Sigma_2^p}$-complete [20]. The latter is to find, given a formula $\varphi(X) = \forall Y \psi(X,Y)$, the truth assignment μ_X^{last} of X that satisfies φ and comes last in the lexicographical ordering if it exists, where ψ is a 3SAT instance. We give an $\mathsf{FP}^{\Sigma_2^p}$ algorithm for $\mathsf{FRP}(\exists\mathrm{FO}^+)$ to compute a top-k package selection if it exists.

(2) We show that $\mathsf{FRP}(\mathcal{L}_Q)$ is $\mathsf{FPSPACE(poly)}$-hard by reducing to it all functions computable by a PSPACE Turing machine in which the output on the working tape is bounded by a polynomial, when $\mathcal{L}_Q$ is $\mathrm{DATALOG_{nr}}$ or FO; similarly for $\mathsf{FRP}(\mathrm{DATALOG})$. For the upper bounds, we give an algorithm in $\mathsf{FPSPACE(poly)}$ (resp. $\mathsf{FEXPTIME(poly)}$) for $\mathsf{FRP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ is $\mathrm{DATALOG_{nr}}$ or FO (resp. DATALOG).

(3) When Q and Q_c are fixed, we show that $\mathsf{FRP}(\mathrm{CQ})$ in the absence of Q_c is $\mathsf{FP}^{\mathsf{NP}}$-hard by reduction from MAX-WEIGHT SAT, which is $\mathsf{FP}^{\mathsf{NP}}$-complete (cf. [25]). Given a set $\mathcal{C}$ of clauses with weights, MAX-WEIGHT SAT is to find a truth assignment that satisfies a set of clauses in $\mathcal{C}$ with the most total weight. For the upper bound, we give an $\mathsf{FP}^{\mathsf{NP}}$ algorithm for $\mathsf{FRP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ is FO or DATALOG.

(4) When Q_c is absent, $\mathsf{FRP}(\mathrm{CQ})$ is $\mathsf{FP}^{\mathsf{NP}}$-hard by proof (3) given above, and the algorithm for $\mathsf{FRP}(\exists\mathrm{FO}^+)$ given in proof (1) is now in $\mathsf{FP}^{\mathsf{NP}}$. The proofs for $\mathrm{DATALOG_{nr}}$, FO and DATALOG given in (2) still work in this special case, as no Q_c is used there when verifying the lower bounds. □

Deciding the maximum bound. We show that $\mathsf{MBP}(\mathrm{CQ})$ is $\mathsf{D}_2^{\mathsf{p}}$-complete. Here $\mathsf{D}_2^{\mathsf{p}}$ is the class of languages recognized by oracle machines that make a query to a Σ_2^p oracle and a query to a Π_2^p oracle. That is, L is in $\mathsf{D}_2^{\mathsf{p}}$ if there exist languages $L_1 \in \Sigma_2^p$ and $L_2 \in \Pi_2^p$ such that $L = L_1 \cap L_2$ [33], analogous to how DP is defined with NP and coNP [25].

When $\mathcal{L}_Q$ is FO, $\mathrm{DATALOG_{nr}}$ or DATALOG, $\mathsf{MBP}(\mathcal{L}_Q)$ and $\mathsf{RPP}(\mathcal{L}_Q)$ have the same complexity. Moreover, the absence of Q_c has the same impact on $\mathsf{MBP}(\mathcal{L}_Q)$ as on $\mathsf{RPP}(\mathcal{L}_Q)$.

Theorem 3.4: For $\mathsf{MBP}(\mathcal{L}_Q)$, the combined complexity is

- $\mathsf{D}_2^{\mathsf{p}}$-complete when $\mathcal{L}_Q$ is CQ, UCQ or $\exists\mathrm{FO}^+$;
- PSPACE-complete when $\mathcal{L}_Q$ is $\mathrm{DATALOG_{nr}}$ or FO; and
- $\mathsf{EXPTIME}$-complete when $\mathcal{L}_Q$ is DATALOG.

When compatibility constraints are absent, its combined complexity remains unchanged for $\mathrm{DATALOG_{nr}}$, FO and DATALOG, but it is DP-complete for CQ, UCQ and $\exists\mathrm{FO}^+$.

Its data complexity is DP-complete for all the languages, in the presence or absence of compatibility constraints. □

Proof sketch: (1) We show that $\mathsf{MBP}(\mathrm{CQ})$ is $\mathsf{D}_2^{\mathsf{p}}$-hard by reduction from $\exists^*\forall^*$3DNF–$\forall^*\exists^*$3CNF. Given a pair (φ_1, φ_2) of $\exists^*\forall^*$3DNF instances, the latter is to decide whether φ_1 is true and φ_2 is false, and is $\mathsf{D}_2^{\mathsf{p}}$-complete [33]. We show that $\mathsf{MBP}(\exists\mathrm{FO}^+)$ is in $\mathsf{D}_2^{\mathsf{p}}$ by giving an algorithm that makes a query to a Σ_2^p oracle and a query to a Π_2^p oracle.

(2) For $\mathrm{DATALOG_{nr}}$, FO and DATALOG, the proof for MBP extends its counterpart in the proofs (2-3) of Theorem 3.1.

(3) When Q and Q_c are fixed, we show that $\mathsf{MBP}(\mathrm{CQ})$ is DP-hard by reduction from SAT-UNSAT, and give a DP algorithm for $\mathsf{MBP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ is FO or DATALOG.

(4) In the absence of Q_c, the lower bound proofs given in (3) for CQ and in (2) for the others remain intact, since no Q_c is used there. The upper bounds given in (2-3) still hold. □

Counting valid packages. When it comes to the counting problem $\mathsf{CPP}(\mathcal{L}_Q)$, we provide its complexity as follows.

Theorem 3.5: For $\mathsf{CPP}(\mathcal{L}_Q)$, the combined complexity is

- $\#\cdot\mathsf{coNP}$-complete when $\mathcal{L}_Q$ is CQ, UCQ or $\exists\mathrm{FO}^+$;
- $\#\cdot\mathsf{PSPACE}$-complete when $\mathcal{L}_Q$ is $\mathrm{DATALOG_{nr}}$ or FO;
- $\#\cdot\mathsf{EXPTIME}$-complete when $\mathcal{L}_Q$ is DATALOG.

In the absence of compatibility constraints, its combined complexity remains unchanged for $\mathrm{DATALOG_{nr}}$, FO and DATALOG, but it is $\#\cdot\mathsf{NP}$-complete for CQ, UCQ and $\exists\mathrm{FO}^+$.

Its data complexity is $\#\cdot\mathsf{P}$-complete for all the languages in the presence or absence of compatibility constraints. □

Here we use the framework of predicate-based counting classes introduced in [15]. For a complexity class C of decision problems, $\#\cdot\mathsf{C}$ is the class of all counting problems associated with a predicate R_L that satisfies the following conditions: (a) R_L is polynomially balanced (see its definition above); and (b) the decision problem "given x and y, whether $R_L(x,y)$" is in C. A counting problem is to compute the cardinality of the set $\{y \mid R_L(x,y)\}$, *i.e.,* it is to find how many y there are such that $R_L(x,y)$ is satisfied.

It is known that $\#\cdot\mathsf{P} = \#\mathsf{P}$, $\#\cdot\mathsf{NP} \subseteq \#\mathsf{NP} = \#\cdot\mathsf{P}^{\mathsf{NP}} = \#\cdot\mathsf{coNP}$, but $\#\cdot\mathsf{NP} = \#\cdot\mathsf{coNP}$ iff $\mathsf{NP} = \mathsf{coNP}$, where $\#\mathsf{P}$ and $\#\mathsf{NP}$ are counting classes in the machine-based framework of [31]. From these we know that the combined complexity of $\mathsf{CPP}(\mathrm{CQ})$ is $\#\mathsf{NP}$-complete, and the data complexity of $\mathsf{CPP}(\mathcal{L}_Q)$ is $\#\mathsf{P}$-complete for all the languages considered.

Proof sketch: (1) We show that $\mathsf{CPP}(\mathrm{CQ})$ is $\#\cdot\mathsf{coNP}$-hard by reduction from $\#\Pi_1$SAT. Given $\varphi(X,Y) = \forall X\, \psi(X,Y)$, where ψ is a 3DNF, $\#\Pi_1$SAT is to count the number of truth assignments of Y that satisfy φ, and is $\#\cdot\mathsf{coNP}$-complete [11]. The reduction is an 1-1 mapping from the solutions to $\mathsf{CPP}(\mathrm{CQ})$ to the truth assignments for $\varphi(X,Y)$, and hence is *parsimonious*. We also show that $\mathsf{CPP}(\exists\mathrm{FO}^+)$ is in $\#\cdot\mathsf{coNP}$.

(2) We show that CPP is $\#\cdot\mathsf{PSPACE}$-hard for $\mathrm{DATALOG_{nr}}$ and FO by parsimonious reductions from #QBF, which is $\#\cdot\mathsf{PSPACE}$-complete [21]. Given $\varphi = \exists X \forall y_1 P_2 y_2 \cdots P_n y_n\, \psi$, where ψ is a 3SAT instance over X and $\{y_i \mid i \in [1,n]\}$, and P_i is $\forall$ or $\exists$, #QBF is to count the number of truth assignments of X that satisfy φ. For DATALOG, we verify that CPP is $\#\cdot\mathsf{EXPTIME}$-hard by parsimonious reduction from all counting problems in $\#\cdot\mathsf{EXPTIME}$. We also show that CPP is in $\#\cdot\mathsf{PSPACE}$ (resp. $\#\cdot\mathsf{EXPTIME}$) for $\mathrm{DATALOG_{nr}}$ and FO (resp. DATALOG) by the definition of $\#\cdot\mathsf{C}$ classes.

(3) When Q and Q_c are fixed, we show that $\mathsf{CPP}(\mathrm{CQ})$ is $\#\cdot\mathsf{P}$-complete by parsimonious reduction from #SAT, which is $\#\cdot\mathsf{P}$-complete (cf. [25], by $\#\mathsf{P} = \#\cdot\mathsf{P}$). Given an instance ψ of 3SAT, #SAT is to count truth assignments that satisfy ψ. We also show that CPP is in $\#\cdot\mathsf{P}$ for all the languages.

(4) When Q_c is absent, we show that $\mathsf{CPP}(\mathrm{CQ})$ is $\#\cdot\mathsf{NP}$-hard by parsimonious reduction from $\#\Sigma_1$SAT, which is $\#\cdot\mathsf{NP}$-complete [11]. Given $\varphi(X,Y) = \exists X\ \psi(X,Y)$, $\#\Sigma_1$SAT is to count truth assignments of Y that satisfy φ, where ψ is a 3DNF. We also show that $\mathsf{CPP}(\exists\mathrm{FO}^+)$ is in $\#\cdot\mathsf{NP}$. When $\mathcal{L}_Q$ is $\mathrm{DATALOG_{nr}}$, FO or DATALOG, the proofs of (2) given above carry over (its lower bound proofs do not use Q_c). □

4. SPECIAL CASES OF POI RECOMMENDATIONS

The results of Section 3 tell us that RPP, FRP, MBP and CPP have rather high complexity. In this section we revisit these problems for special cases of package recommendations, to explore the impact of various parameters of these problems on their complexity. We consider the settings when packages are bounded by a constant instead of a polynomial, when $\mathcal{L}_Q$ is a language for which the membership problem is in PTIME, and when compatibility constraints are simply PTIME functions. We also study item recommendations, for which each package has a single item, and compatibility constraints are absent. Our main conclusion of this section is that the complexity of these problem is rather *robust*: these restrictions simplify the analyses, but not much.

Packages with a fixed bound. One might be tempted to think that fixing package size would simplify the analyses. Below we study the impact of fixing package sizes on package selections, in the presence of compatibility constraints Q_c, by considering packages N such that $|N| \leq B_p$, where B_p is a predefined *constant* rather than a polynomial.

We show that fixing package sizes does not make our lives easier when combined complexity is concerned. In contrast, this does simplify the analyses of data complexity.

Corollary 4.1: For packages with a constant bound B_p, the combined complexity bounds of RPP, FRP, MBP and CPP are the same as given in Theorems 3.1, 3.3, 3.4 and 3.5, respectively; and the data complexity is

- in PTIME for RPP,
- in FP for FRP,
- in PTIME for MBP, and
- in FP for CPP,

for all the languages of Section 2. The complexity remains unchanged even when B_p is fixed to be 1. □

Proof sketch: (1) The lower bounds of RPP, FRP, MBP and CPP in the presence of Q_c hold here, since their proofs (Theorems 3.1, 3.3, 3.4 and 3.5) use only top-1 package with one item, and all the upper bounds carry over here. (2) For fixed Q and Q_c, we give algorithms in PTIME, FP, PTIME and FP for RPP, FRP, MBP and CPP, respectively. □

SP queries. In contrast, for queries that have a PTIME complexity for their membership problem, variable package sizes lead to higher complexity of RPP, FRP, MBP and CPP than their counterparts for packages with a fixed bound.

To illustrate this, we consider SP queries, a simple fragment of CQ queries that support projection and selection operators only. An SP query is of the form

$$Q(\vec{x}) = \exists \vec{x}\vec{y}\ (R(\vec{x},\vec{y}) \wedge \psi(\vec{x},\vec{y})),$$

where ψ is a conjunction of predicates $=, \neq, <, \leq, >$ and $\geq$.

The result below holds for all query languages with a PTIME membership problem, including but not limited to SP. In fact the lower bounds remain intact even when the selection criteria are specified by an *identity query*, when $|\vec{y}| = 0$ and ψ is a tautology in an SP query.

Corollary 4.2: For SP queries, the combined complexity and data complexity are

- coNP-complete for RPP, but in PTIME for packages with a fixed (constant) bound B_p;
- $\mathsf{FP}^{\mathsf{NP}}$-complete for FRP, but in FP for fixed B_p;
- DP-complete for MBP, but in PTIME for fixed B_p; and
- $\#\cdot$P-complete for CPP, but in FP for fixed B_p.

when compatibility constraints are present or absent. □

Proof sketch: (1) For packages of variable sizes, the lower bounds of RPP, FRP, MBP and CPP with fixed Q in CQ hold for SP. Indeed, their proofs of Theorems 3.1, 3.3, 3.4 and 3.5 use an identity query as Q, which is in SP. For the upper bounds, the algorithms given there for RPP, FRP, MBP and CPP with a fixed Q apply to arbitrary SP queries.
(2) For packages with a constant bound, the algorithms for fixed Q of Corollary 4.1 apply to SP queries, fixed or not. □

PTIME compatibility constraints. One might also think that we would get lower complexity with PTIME compatibility constraints. That is, we simply treat compatibility constraints as PTIME functions rather than queries in $\mathcal{L}_Q$. In this setting, the complexity remains the same as its counterpart when Q_c is absent, no better and no worse.

Corollary 4.3: With PTIME compatibility constraints Q_c, the combined complexity and data complexity of RPP, FRP MBP and CPP remain the same as their counterparts in the absence of Q_c, as given in Theorems 3.2, 3.3, 3.4 and 3.5, respectively, for all the languages of Section 2. □

Proof sketch: The lower bounds of RPP, FRP, MBP and CPP in the absence of Q_c obviously carry over to this setting, since when Q_c is empty (see Section 2), Q_c is in PTIME. The upper bound proofs for Theorems 3.2, 3.3, 3.4 and 3.5 in the absence of Q_c also remain intact here. Indeed, adding an extra PTIME step for checking $Q_c(N, D) = \emptyset$ does not increase the complexity of the algorithms given there. □

Item recommendations. As remarked in Section 2, item recommendations are a special case of package recommendations when (a) compatibility constraints Q_c are *absent*, and (b) each package consists of a single item, *i.e.,* with *a fixed size* 1. Given a database D, a query $Q \in \mathcal{L}_Q$, a utility function $f()$ and a natural number $k \geq 1$, a top-k item selection is a top-k package selection specified in terms of (Q, D, f).

When Q_c is absent and packages have a fixed size 1, one might expect that the recommendation analyses would become much simpler. Unfortunately, this is not the case.

Theorem 4.4: For items, RPP, FRP, MBP and CPP have

- the same combined complexity as their counterparts in the absence of Q_c (Theorems 3.2, 3.3, 3.4, 3.5), and
- the same data complexity as their counterparts for packages with a constant bound (Corollary 4.1),

for all the query languages given in Section 2. □

Proof sketch: (1) Combined complexity. The upper bounds of these problems in the absence of Q_c (Theorems 3.2, 3.3, 3.4, 3.5) obviously remain intact here. The lower bound proofs for RPP and CPP given there are still valid for item recommendations, since they require only top-1 packages with a single item. For FRP and MBP, however, new lower bound proofs are required for item recommendations.

More specifically, we show that FRP(CQ) is $\mathsf{FP}^{\mathsf{NP}}$-hard by reduction from MAX-WEIGHT SAT, and that MBP(CQ) is DP-hard by reduction from SAT-UNSAT, for item recommendations. For other languages $\mathcal{L}_Q$, the proofs for FRP($\mathcal{L}_Q$) and MBP($\mathcal{L}_Q$) are given along the same lines as their counterparts for Theorems 3.3 and 3.4, respectively.

(2) Data complexity. The algorithms developed for Corollary 4.1 suffice for item selections when Q is fixed. □

Summary. From these results we find the following.

Variable sizes of packages. (1) For simple queries that have a PTIME membership problem, such as SP, the problems with variable package sizes have higher combined *and* data complexity than their counterparts with a fixed (constant) package size. This is in line with the claim of [34]. (2) In contrast, for any query language that subsumes CQ, variable sizes of packages have no impact on the *combined complexity* of these problems. This is consistent with the observation of [26]. (3) When it comes to *the data complexity*, however, variable (polynomially) package sizes make our lives harder: RPP, FRP, MBP and CPP in this setting have a higher data complexity than their counterparts with a fixed package size.

Compatibility constraints. (1) For CQ, UCQ and $\exists$FO$^+$, the presence of Q_c increases the combined complexity of the analyses. (2) In contrast, for more powerful languages such as DATALOG$_{\mathsf{nr}}$, FO and DATALOG, neither Q_c nor variable sizes make any difference. Indeed, RPP, FRP, MBP and CPP have exactly the same combined complexity as their counterparts for item recommendations, in the presence or absence of Q_c. (3) For data complexity, the presence of Q_c has no impact. Indeed, when Q_c is fixed, it is in PTIME to check $Q_c(N,D)=\emptyset$ for all $\mathcal{L}_Q$ in which Q_c is expressed; hence Q_c can be encoded in the $\mathsf{cost}()$ function, and no longer needs to be treated separately. (4) To simplify the discussion we use $\mathcal{L}_Q$ to specify Q_c. Nonetheless, all the complexity results remain intact for any class $\mathcal{C}$ of Q_c whose satisfiability problem has the same complexity as the membership problem for $\mathcal{L}_Q$. In particular, when $\mathcal{C}$ is a class of PTIME functions, the presence of Q_c has no impact on the complexity.

The number k of packages. All the lower bounds of RPP, FRP and MBP remain intact *when* $k=1$ (k is irrelevant to CPP), *i.e.,* they carry over to top-1 package selections.

5. RECOMMENDATIONS OF QUERY RELAXATIONS

We next study query relaxation recommendations. In practice a selection query Q often finds no sensible packages. When this happens, the users naturally want the recommendation system to suggests how to revise their selection criteria by relaxing the query Q. We are not aware of any recommendation systems that support this functionality.

Below we first present query relaxations (Section 5.1). We then identify two query relaxation recommendation problems, and establish their complexity bounds (Section 5.2).

5.1 Query Relaxations

Consider a query Q, in which a set X of variables (free or bound) and a set E of constants are parameters that can be modified, *e.g.,* variables or constants indicating departure time and date of flights. Following [8], we relax Q by replacing constants in E with variables, and replacing repeated variables in X with distinct variables, as follows.

(1) For each constant $c\in E$, we associate a variable w_c with c. We denote the tuple consisting of all such variables as $\vec{w}$.

(2) For each variable $x\in X$ that appears at least twice in atoms of Q, we introduce a new variable u_x and substitute u_x for one of the occurrences of x. For instance, an equijoin $Q_1(\vec{v},y)\wedge Q_2(y,\vec{v}')$ is converted to $Q_1(\vec{v},y)\wedge Q_2(u_y,\vec{v}')$, a Cartesian product. This is repeated until no variable has multiple occurrences. Let $\vec{u}$ be the tuple of all such variables.

We denote the domain of w_c (resp. u_x) as $\mathsf{dom}(R.A)$ if c (resp. x) appears in Q as an A-attribute value in relation R.

To prevent relaxations that are too general, we constrain variables in $\vec{w}$ and $\vec{u}$ with certain ranges, by means of techniques developed for query relaxations [8, 18] and preference queries [29]. To simplify the discussion, we assume that for each attribute A in a relation R, a distance function $\mathsf{dist}_{R.A}(a,b)$ is defined. Intuitively, if $\mathsf{dist}_{R.A}(a,b)$ is within a bound, then b is close enough to a, and we can relax Q by replacing a with its "neighbor" b. For instance, *DB* can be generalized to *CS* if $\mathsf{dist}(DB, CS)$ is small enough [8]. We denote by Γ the set of all such distance functions.

Given Γ, we define a *relaxed query* Q_Γ *of* $Q(\vec{x})$ as:

$$Q_\Gamma(\vec{x}) \;=\; \exists\vec{w}\;\exists\vec{u}\;\big(Q'(\vec{x},\vec{w},\vec{u})\;\wedge\;\psi_w(\vec{w})\;\wedge\;\psi_u(\vec{u})\big),$$

where Q' is obtained from Q by substituting w_c for constant c, and u_x for a repeated occurrence of x. Here $\psi_w(\vec{w})$ is a conjunction of predicates of either (a) $\mathsf{dist}_{R.A}(w_c,c)\leq d$, where the domain of w_c is $\mathsf{dom}(R.A)$, and d is a constant, or (b) $w_c=c$, *i.e.,* the constant c is unchanged. Query $\psi_w(\vec{w})$ includes such a conjunct for each $w_c\in\vec{w}$; similarly for $\psi_u(\vec{u})$.

We define *the level* $\mathsf{gap}(\gamma)$ *of relaxation of a predicate* γ in $\psi_w(\vec{w})$ as follows: $\mathsf{gap}(\gamma)=d$ if γ is $\mathsf{dist}_{R.A}(w_c,c)\leq d$, and $\mathsf{gap}(\gamma)=0$ if γ is $w_c=c$; similarly for a predicate in $\psi_u(\vec{u})$.

We define *the level of relaxation of query* Q_Γ, denoted by $\mathsf{gap}(Q_\Gamma)$, to be $\mathsf{sum}_{\gamma\in(\psi_w(\vec{w})\cup\psi_u(\vec{u}))}\mathsf{gap}(\gamma)$.

Example 5.1: Recall query Q defined on flight and POI in Example 1.1. The query finds no items, as there is no direct flight from EDI to NYC. Suppose that E has constants EDI, NYC, 1/1/2012 and $X=\{x_{\mathsf{To}}\}$, and that the user accepts a city within 15 miles of the original departure city (resp. destination) as From (resp. To), where dist( ) measures the distances between cities. Then we can relax Q as:

$$\begin{aligned}Q_1(&\mathsf{f\#,Pr,nm,tp,tkt,tm}) = \exists \mathsf{DT,AT,AD},u_{\mathsf{To}},w_{\mathsf{Edi}},w_{\mathsf{NYC}},w_{\mathsf{DD}}\\ &\big(\mathsf{flight}(\mathsf{f\#},w_{\mathsf{Edi}},x_{\mathsf{To}},\mathsf{DT},\ w_{\mathsf{DD}},\ \mathsf{AT},\ \mathsf{AD},\ \mathsf{Pr})\wedge x_{\mathsf{To}}=w_{\mathsf{NYC}}\\ &\wedge \mathsf{POI}(\mathsf{nm},\ u_{\mathsf{To}},\ \mathsf{tp},\ \mathsf{tkt},\ \mathsf{tm})\wedge w_{\mathsf{DD}}=1/1/2012\\ &\wedge \mathsf{dist}(w_{\mathsf{NYC}},\ \text{NYC})\leq 15\wedge \mathsf{dist}(w_{\mathsf{Edi}},\ \text{EDI})\leq 15\wedge x_{\mathsf{To}}=u_{\mathsf{To}}\big).\end{aligned}$$

The relaxed Q_1 finds direct flights from EDI to EWR, since the distance between NYC to EWR is within 15 miles.

We can relax Q_1 by allowing w_{DD} to be within 3 days of 1/12/11, where the distance function for dates is $\mathsf{dist}_d()$:

$$\begin{aligned}Q_2(&\mathsf{f\#,Pr,nm,tp,tkt,tm}) = \exists \mathsf{DT,AT,AD},u_{\mathsf{To}},w_{\mathsf{Edi}},w_{\mathsf{NYC}},w_{\mathsf{DD}}\\ &\big(\mathsf{flight}(\mathsf{f\#},\ w_{\mathsf{Edi}},\ x_{\mathsf{To}},\ \mathsf{DT},\ w_{\mathsf{DD}},\ \mathsf{AT},\ \mathsf{AD},\ \mathsf{Pr})\wedge x_{\mathsf{To}}=w_{\mathsf{NYC}}\\ &\wedge \mathsf{POI}(\mathsf{nm},\ u_{\mathsf{To}},\ \mathsf{tp},\ \mathsf{tkt},\ \mathsf{tm})\wedge \mathsf{dist}_d(w_{\mathsf{DD}},\ 1/1/2012)\leq 3\\ &\wedge \mathsf{dist}(w_{\mathsf{NYC}},\ \text{NYC})\leq 15\wedge \mathsf{dist}(w_{\mathsf{Edi}},\ \text{EDI})\leq 15\wedge x_{\mathsf{To}}=u_{\mathsf{To}}\big).\end{aligned}$$

Then Q_2 may find more available direct flights than Q_1, with possibly cheaper airfare. One can further relax Q_2 by allowing u_{To} and x_{To} to match different cities nearby, *i.e.,* we convert the equijoin to a Cartesian product. □

We consider simple query relaxation rules here just to illustrate the main idea of query relaxation recommendations, and defer a full treatment of this issue to future work.

5.2 Query Relaxation Recommendations

We now study recommendation problems for query relaxations, for package selections and for item selections.

The query relaxation problem for packages. Consider a database D, queries Q and Q_c in $\mathcal{L}_Q$, functions $\mathsf{cost}()$ and $\mathsf{val}()$, a cost budget C, a rating bound B, and a natural number $k\geq 1$. When there exists no top-k package selection for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C)$, we need to relax Q to find more packages for the users. More specifically, let Γ be a collection of distance functions, and X and E be sets of variables and constants in Q, respectively, which are parameters that can be modified. We want to find a relaxed query Q_Γ of Q such that there exists a set $\mathcal{N}$ of k valid packages for $(Q_\Gamma,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C,B)$, *i.e.,* for each $N\in\mathcal{N}$, $N\subseteq Q_\Gamma(D)$, $Q_c(N,D)=\emptyset$, $\mathsf{cost}(N)\leq C$, $\mathsf{val}(N)\geq B$, and $|N|$

is bounded by a polynomial in $|D|$. Moreover, we want Q_Γ to *minimally differ* from the original Q, stated as follows.

For a constant g, a relaxed query Q_Γ of Q is called *a relaxation of Q for* $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$ if (a) there exists a set $\mathcal{N}$ of k distinct valid packages for $(Q_\Gamma, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$, and (b) $\mathsf{gap}(Q_\Gamma) \le g$.

QRPP($\mathcal{L}_Q$): *The query relaxation rec. problem (packages)*	
INPUT:	A database D, a query $Q \in \mathcal{L}_Q$ with sets X and E identified, a query $Q_c \in \mathcal{L}_Q$, two functions $\mathsf{cost}()$ and $\mathsf{val}()$, natural numbers C, B, g and $k \ge 1$, and a collection Γ of distance functions.
QUESTION:	Does there exist a relaxation Q_Γ of Q for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$?

No matter how important, QRPP is nontrivial: it is Σ_2^p-complete for CQ, PSPACE-complete for DATALOG$_{\mathsf{nr}}$ and FO, and EXPTIME-complete for DATALOG. It is NP-complete when selection criteria Q and compatibility constraints Q_c are both fixed. Fixing Q_c alone reduces the combined complexity of QRPP($\mathcal{L}_Q$) when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$, but it does not help when it comes to DATALOG$_{\mathsf{nr}}$, FO and DATALOG, or when the data complexity is concerned.

Theorem 5.1: For QRPP($\mathcal{L}_Q$), the combined complexity is

- Σ_2^p-complete when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$;
- PSPACE-complete when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO; and
- EXPTIME-complete when $\mathcal{L}_Q$ is DATALOG.

In the absence of compatibility constraints, its combined complexity remains unchanged for DATALOG$_{\mathsf{nr}}$, FO and DATALOG, and it is NP-complete for CQ, UCQ and $\exists$FO$^+$.

Its data complexity is NP-complete for all the languages, in the presence or absence of compatibility constraints. □

Proof sketch: (1) We verify that QRPP(CQ) is Σ_2^p-hard by reduction from the $\exists^*\forall^*$3DNF problem, by using relaxed queries. We show that QRPP($\exists$FO$^+$) is in Σ_2^p by giving a nondeterministic PTIME algorithm that calls an NP oracle.
(2) For DATALOG$_{\mathsf{nr}}$ and FO, we show that QRPP is PSPACE-hard by reductions from Q3SAT and the membership problem for FO, respectively, by using relaxed queries. We give a PSPACE algorithm to check QRPP. Along the same lines we show that QRPP(DATALOG) is EXPTIME-complete.
(3) When Q is fixed, we show that QRPP(CQ) is already NP-hard in the absence of Q_c, by reduction from 3SAT. We show the upper bound by giving an NP algorithm to check QRPP($\mathcal{L}_Q$) for fixed queries Q and Q_c in FO or DATALOG.
(4) In the absence of Q_c, it has been shown that QRPP(CQ) is NP-hard by the proof of (3) above, and the algorithm for QRPP($\exists$FO$^+$) given in (1) is now in NP. For DATALOG$_{\mathsf{nr}}$, FO and DATALOG, the proofs given in (2) above can be applied here, since their lower bound proofs do not use Q_c, and their upper bounds still hold in this special case. □

The query relaxation problem for items. We also study a special case of QRPP, for item selections. Given D, Q, Γ, B, k, and a utility function $f()$, QRPP *for items* is to decide whether there exist a relaxation Q_Γ of Q for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$, when Q_c is empty, and $\mathsf{cost}(), \mathsf{val}()$ and C are derived from $f()$ as given in Section 2.

Compared to its package counterpart, item selections simplify the data complexity analyses of query relaxation recommendations. However, it gets no better than QRPP in the absence of Q_c when the combined complexity is concerned.

Corollary 5.2: For all the query languages $\mathcal{L}_Q$ given in Section 2, QRPP($\mathcal{L}_Q$) for items (1) has the same combined complexity as QRPP($\mathcal{L}_Q$) in the absence of compatibility constraints; and (2) its data complexity is in PTIME. □

Proof sketch: For items, (1) we show that QRPP(CQ) is NP-hard by reduction from 3SAT. To check QRPP($\exists$FO$^+$) we give an NP algorithm. For DATALOG$_{\mathsf{nr}}$, FO or DATALOG, the lower bounds of Theorem 5.1 hold here since their proofs use top-1 items only. The upper bounds of the combined complexity also carry over. (2) We give a PTIME algorithm to check QRPP for fixed Q in FO or DATALOG. □

Remarks. (1) All the lower bounds of this section remains intact when $k = 1$, *i.e.,* for top-1 package or item selections. (2) The proofs of Theorem 5.1 and Corollary 5.2 also tell us that for packages with a constant bound, QRPP($\mathcal{L}_Q$) has the same combined complexity as its counterpart for packages with variable sizes, and it has the same data complexity as its counterpart for items. (3) In addition, when Q_c is a PTIME function, QRPP($\mathcal{L}_Q$) has the same combined and data complexity as its counterpart in the absence of Q_c. These are consistent with Corollaries 4.1 and 4.3.

6. ADJUSTMENT RECOMMENDATIONS

We next study adjustment recommendations. In practice the collection D of items maintained by a recommendation system may fail to provide items that most users want. When this happens, the vendors of the system would want the system to recommend how to "minimally" modify D such that users' requests could be satisfied. Below we first present adjustments to D (Section 6.1). We then study adjustment recommendations problems (Section 6.2).

6.1 Adjustments to Item Collections

Consider a database D consisting of items provided by a system, and a collection D' of additional items. We use $\Delta(D, D')$ to denote *adjustments to* D, which is a set consisting of (a) tuples to be deleted from D, and (b) tuples from D' to be inserted into D. We use $D \oplus \Delta(D, D')$ to denote the database obtained by modifying D with $\Delta(D, D')$.

Consider queries Q, Q_c in $\mathcal{L}_Q$, functions $\mathsf{cost}()$ and $\mathsf{val}()$, a cost budget C, a rating bound B, and a natural number $k \ge 1$, such that there exists no top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$. We want to find a set $\Delta(D, D')$ of adjustments to D such that there exists a set $\mathcal{N}$ of k valid packages for $(Q, D \oplus \Delta(D, D'), Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$, *i.e.,* $D \oplus \Delta(D, D')$ yields k packages N that are rated above B, and satisfy the selection criteria Q, compatibility constraints Q_c as well as aggregate constraints $\mathsf{cost}(N) \le C$.

One naturally wants to find a "minimum" $\Delta(D, D')$ to adjust D. For a constant $k' \ge 1$, we call $\Delta(D, D')$ *a package adjustment* for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$ if (a) $|\Delta(D, D')| \le k'$, and (b) there exist k distinct valid packages for $(Q, D \oplus \Delta(D, D'), Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$.

6.2 Deciding Adjustment Recommendations

These suggest that we study the following problem.

The adjustment recommendation problem. Given D, D', Q, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, k and k', the *adjustment recommendation problem for packages*, ARPP, is to decide whether there is a package adjustment $\Delta(D, D')$ for

$(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$. This problem is no easier than the analyses of query relaxation recommendations. Indeed, $\mathsf{ARPP}(\mathcal{L}_Q)$ has the same combined and data complexity as $\mathsf{QRPP}(\mathcal{L}_Q)$, although their proofs are quite different.

Theorem 6.1: The combined complexity of $\mathsf{ARPP}(\mathcal{L}_Q)$ is

- Σ_2^p-complete when $\mathcal{L}_Q$ is CQ, UCQ or $\exists\text{FO}^+$;
- PSPACE-complete when $\mathcal{L}_Q$ is $\text{DATALOG}_{\mathsf{nr}}$ or FO; and
- EXPTIME-complete when $\mathcal{L}_Q$ is DATALOG.

In the absence of compatibility constraints, its combined complexity remains unchanged for $\text{DATALOG}_{\mathsf{nr}}$, FO and DATALOG, and it is NP-complete for CQ, UCQ and $\exists\text{FO}^+$.

Its data complexity is NP-complete for all the languages, in the presence or absence of compatibility constraints. □

Proof sketch: (1) We show that ARPP(CQ) is Σ_2^p-hard by reduction from the $\exists^*\forall^*$3DNF problem. The reduction here makes use of updates $\Delta(D, D')$, and is different from the one for QRPP(CQ). To verify the upper bound, we give an NP algorithm that uses an NP oracle to check $\mathsf{ARPP}(\exists\text{FO}^+)$.

(2) For $\text{DATALOG}_{\mathsf{nr}}$ and FO, we show that ARPP is PSPACE-hard by reductions from Q3SAT and the membership problem for FO, respectively, which again use $\Delta(D, D')$ and are different from the ones used earlier. We give a PSPACE algorithm to check ARPP for $\text{DATALOG}_{\mathsf{nr}}$ and FO. Similarly we show that ARPP(DATALOG) is EXPTIME-complete.

(3) When Q and Q_c are fixed, we show that ARPP(CQ) is NP-hard without Q_c by reduction from 3SAT. We also provide an NP algorithm to check ARPP for FO and DATALOG.

(4) When Q_c is absent, ARPP(CQ) is NP-hard even when Q is fixed by the proof of (3), and the algorithm for $\exists\text{FO}^+$given in (1) is now in NP. For the languages considered in (2), their lower bound proofs do not use Q_c, and their upper bound proofs cover this special case. Moreover, the algorithm developed in (3) and the lower bound of ARPP(CQ) verify that the data complexity is NP-complete. □

The adjustment recommendation problem for items. Given D, D', Q, B, k, k' and a utility function $f()$, ARPP *for items* is to decide whether there is an adjustment $\Delta(D, D')$ for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$, where Q_c is empty, and $\mathsf{cost}(), \mathsf{val}()$, C are derived from $f()$ (see Section 2).

One might expect that fixing package sizes in item selections would simplify the analyses of adjustment recommendations. Recall that all the problems we have studied so far have a lower data complexity for item selections than their counterparts for packages. For instance, the data complexity of QRPP for items is in PTIME while it is NP-complete for packages; similarly for RPP, FRP, MBP and CPP. In contrast, we show below that the data complexity of ARPP for packages is *robust*: it remains intact for items. In other words, fixing package sizes does not help here.

Corollary 6.2: For all the languages $\mathcal{L}_Q$ given in Section 2, ARPP in the absence of compatibility constraints and ARPP for items have the same combined and data complexity. □

Proof sketch: All the lower bound proofs for $\mathsf{ARPP}(\mathcal{L}_Q)$ in the absence of Q_c use top-k item selections, and ARPP(CQ) is NP-hard for fixed Q. Hence for all $\mathcal{L}_Q$, ARPP without Q_c has the same combined complexity as ARPP for items, and the data complexity of ARPP carries over to item selections. Note that when proving ARPP(CQ) is NP-hard for fixed Q, we do not use $k = 1$, the only case in the entire paper. □

Remarks. One can find the following from the proofs of Theorem 6.1 and Corollary 6.2. (1) For packages with a constant bound, $\mathsf{ARPP}(\mathcal{L}_Q)$ has the same combined complexity as $\mathsf{ARPP}(\mathcal{L}_Q)$ for packages with variable sizes, and it has the same data complexity as $\mathsf{ARPP}(\mathcal{L}_Q)$ for items. (3) When Q_c is in PTIME, $\mathsf{ARPP}(\mathcal{L}_Q)$ has the same combined and data complexity as $\mathsf{ARPP}(\mathcal{L}_Q)$ in the absence of Q_c.

7. CONCLUSIONS

We have studied a general model for recommendation systems, and investigated several fundamental problems in the model, from decision problems RPP, MBP to function problem FRP and counting problem CPP. Beyond POI recommendations, we have proposed and studied QRPP for query relaxation recommendations, and ARPP for adjustment recommendations. We have also investigated special cases of these problems, when compatibility constraints Q_c are absent or in PTIME, when all packages are bounded by a constant B_p, and when both Q_c is absent and B_p is fixed to be 1 for item selections. We have provided a complete picture of the lower and upper bounds of these problems, *all matching*, for both their data complexity and combined complexity, when $\mathcal{L}_Q$ ranges over a variety of query languages. These results tell us where complexity of these problems arises.

The main complexity results are summarized in Table 1, annotated with their corresponding theorems (the results for SP (Corollary 4.2) are excluded). As remarked earlier, (1) the data complexity is *independent of* query languages, and remains *unchanged* in the presence of compatibility constraints Q_c or not. However, it *varies* when packages have variable sizes or a constant bound, as shown in Table 1. (2) The complexity bounds of these problems for CQ, UCQ and $\exists\text{FO}^+$ *vary* when Q_c is present or not, and when packages have a constant bound or not. In contrast, the bounds for FO, $\text{DATALOG}_{\mathsf{nr}}$ and DATALOG are *robust*, *regardless of* the presence of Q_c and package sizes. (3) When Q_c is a PTIME function, these problems have the same complexity as their counterparts in the absence of Q_c. (4) Item selections do not come with Q_c and have a fixed package size (see Table 1).

The study of recommendation problems is still preliminary. First, we have only considered simple rules for query relaxations and adjustment recommendations, to focus on the main ideas. These issues deserve a full investigation. Second, this work aims to study a general model that subsumes previous models developed for various applications, and hence adopts generic functions $\mathsf{cost}()$, $\mathsf{val}()$ and $f()$. These need to be fine tuned by incorporating information about users, collaborative filtering and specific aggregate functions. Third, to simplify the discussion we assume that selection criteria Q and compatibility constraints Q_c are expressed in the same language (albeit PTIME Q_c). It is worth studying different languages for Q and Q_c. Fourth, the recommendation problems are mostly intractable. An interesting topic is to identify practical and tractable cases. Another issue to consider are group recommendations [5], to a group of users instead of a single user.

Acknowledgments. Fan and Geerts are supported in part by an IBM scalable data analytics award, and the RSE-NSFC Joint Project Scheme. Fan is also supported in part by the 973 Program 2012CB316200 and NSFC 61133002 of China. Deng is supported in part by 863 2012AA011203, NSFC 61103031 and CPSF 2011M500208 of China.

Problems	Languages	Combined complexity		Data complexity	
		With Q_c	Without Q_c	Poly-bounded	Constant bound
RPP	CQ, UCQ, $\exists$FO$^+$	Π_2^p-complete$^{(\S)}$	DP-complete$^{(\star,\dagger)}$	coNP-complete$^{(\dagger)}$	PTIME $^{(\star)}$
	DATALOG$_{\text{nr}}$, FO	PSPACE-complete$^{(\S)}$	PSPACE-complete$^{(\star,\dagger)}$	coNP-complete$^{(\dagger)}$	PTIME $^{(\star)}$
	DATALOG	EXPTIME-complete$^{(\S)}$	EXPTIME-complete$^{(\star,\dagger)}$	coNP-complete$^{(\dagger)}$	PTIME $^{(\star)}$
		(Th. 3.1)	(Th. 3.2)	(Th. 3.1)	(Cor. 4.1)
FRP	CQ, UCQ, $\exists$FO$^+$	FP$^{\Sigma_2^p}$-complete$^{(\S)}$	FP$^{\text{NP}}$-complete$^{(\star,\dagger)}$	FP$^{\text{NP}}$-complete$^{(\dagger)}$	FP $^{(\star)}$
	DATALOG$_{\text{nr}}$, FO	FPSPACE(poly)-complete$^{(\S)}$	FPSPACE(poly)-complete$^{(\star,\dagger)}$	FP$^{\text{NP}}$-complete$^{(\dagger)}$	FP $^{(\star)}$
	DATALOG	FEXPTIME(poly)-complete$^{(\S)}$	FEXPTIME(poly)-complete$^{(\star,\dagger)}$	FP$^{\text{NP}}$-complete$^{(\dagger)}$	FP $^{(\star)}$
		(Th. 3.3)	(Th. 3.3)	(Th. 3.3)	(Cor. 4.1)
MBP	CQ, UCQ, $\exists$FO$^+$	D$_2^p$-complete	DP-complete$^{(\star,\dagger)}$	DP-complete$^{(\dagger)}$	PTIME $^{(\star)}$
	DATALOG$_{\text{nr}}$, FO	PSPACE-complete$^{(\S)}$	PSPACE-complete$^{(\star,\dagger)}$	DP-complete$^{(\dagger)}$	PTIME $^{(\star)}$
	DATALOG	EXPTIME-complete	EXPTIME-complete$^{(\star,\dagger)}$	DP-complete$^{(\dagger)}$	PTIME $^{(\star)}$
		(Th. 3.4)	(Th. 3.4)	(Th. 3.4)	(Cor. 4.1)
CPP	CQ, UCQ, $\exists$FO$^+$	#·coNP-complete$^{(\S)}$	#·NP-complete$^{(\star,\dagger)}$	#·P-complete$^{(\dagger)}$	FP $^{(\star)}$
	DATALOG$_{\text{nr}}$, FO	#·PSPACE-complete$^{(\S)}$	#·PSPACE-complete$^{(\star,\dagger)}$	#·P-complete$^{(\dagger)}$	FP $^{(\star)}$
	DATALOG	#·EXPTIME-complete$^{(\S)}$	#·EXPTIME-complete$^{(\star,\dagger)}$	#·P-complete$^{(\dagger)}$	FP $^{(\star)}$
		(Th. 3.5)	(Th. 3.5)	(Th. 3.5)	(Cor. 4.1)
QRPP	CQ, UCQ, $\exists$FO$^+$	Σ_2^p-complete$^{(\S)}$	NP-complete$^{(\star,\dagger)}$	NP-complete$^{(\dagger)}$	PTIME $^{(\star)}$
	DATALOG$_{\text{nr}}$, FO	PSPACE-complete$^{(\S)}$	PSPACE-complete$^{(\star,\dagger)}$	NP-complete$^{(\dagger)}$	PTIME $^{(\star)}$
	DATALOG	EXPTIME-complete$^{(\S)}$	EXPTIME-complete$^{(\star,\dagger)}$	NP-complete$^{(\dagger)}$	PTIME $^{(\star)}$
		(Th. 5.1)	(Th. 5.1)	(Th. 5.1)	(Cor. 5.2)
ARPP	CQ, UCQ, $\exists$FO$^+$	Σ_2^p-complete$^{(\S)}$	NP-complete$^{(\star,\dagger)}$	NP-complete$^{(\dagger)}$	NP-complete$^{(\star)}$
	DATALOG$_{\text{nr}}$, FO	PSPACE-complete$^{(\S)}$	PSPACE-complete$^{(\star,\dagger)}$	NP-complete$^{(\dagger)}$	NP-complete$^{(\star)}$
	DATALOG	EXPTIME-complete$^{(\S)}$	EXPTIME-complete$^{(\star,\dagger)}$	NP-complete$^{(\dagger)}$	NP-complete$^{(\star)}$
		(Th. 6.1)	(Th. 6.1)	(Th. 6.1)	(Cor. 6.2)

Table 1: Complexity results ($^{(\star)}$: items (Th. 4.4); $^{(\S)}$: constant bound (Cor. 4.1); $^{(\dagger)}$: PTIME Q_c (Cor. 4.3))

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] G. Adomavicius and A. Tuzhilin. Multidimensional recommender systems: A data warehousing approach. In *WELCOM*, 2001.

[3] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *TKDE*, 17(6):734–749, 2005.

[4] S. Amer-Yahia. Recommendation projects at Yahoo! *IEEE Data Eng. Bull.*, 34(2):69–77, 2011.

[5] S. Amer-Yahia, S. B. Roy, A. Chawla, G. Das, and C. Yu. Group recommendation: Semantics and efficiency. *PVLDB*, 2(1):754–765, 2009.

[6] A. Angel, S. Chaudhuri, G. Das, and N. Koudas. Ranking objects based on relationships and fixed associations. In *EDBT*, 2009.

[7] A. Brodsky, S. Henshaw, and J. Whittle. CARD: A decision-guidance framework and application for recommending composite alternatives. In *RecSys*, 2008.

[8] S. Chaudhuri. Generalization and a framework for query modification. In *ICDE*, 1990.

[9] S. Chaudhuri and M. Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. *JCSS*, 54(1):61–78, 1997.

[10] S. Cohen and Y. Sagiv. An incremental algorithm for computing ranked full disjunctions. *JCSS*, 73(4):648–668, 2007.

[11] A. Durand, M. Hermann, and P. G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *TCS*, 340(3):496–513, 2005.

[12] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66(4):614–656, 2003.

[13] T. Gaasterland and J. Lobo. Qualifying answers according to user needs and preferences. *Fundam. Inform.*, 32(2):121–137, 1997.

[14] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Optimizing and parallelizing ranked enumeration. *PVLDB*, 4(11):1028–1039, 2011.

[15] L. A. Hemaspaandra and H. Vollmer. The satanic notations: counting classes beyond #P and other definitional adventures. *SIGACT News*, 26(1):2–13, 1995.

[16] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, 2008.

[17] A. Kadlag, A. V. Wanjari, J. Freire, and J. R. Haritsa. Supporting exploratory queries in databases. In *DASFAA*, 2004.

[18] N. Koudas, C. Li, A. K. H. Tung, and R. Vernica. Relaxing join and selection queries. In *VLDB*, 2006.

[19] G. Koutrika, B. Bercovitz, and H. Garcia-Molina. FlexRecs: expressing and combining flexible recommendations. In *SIGMOD*, 2009.

[20] M. W. Krentel. Generalizations of Opt P to the polynomial hierarchy. *TCS*, 97(2):183–198, 1992.

[21] R. E. Ladner. Polynomial space counting problems. *SIAM J. Comput.*, 18(6):1087–1097, 1989.

[22] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, 2009.

[23] C. Li, M. A. Soliman, K. C.-C. Chang, and I. F. Ilyas. RankSQL: supporting ranking queries in relational database management systems. In *VLDB*, 2005.

[24] A. Natsev, Y.-C. Chang, J. R. Smith, C.-S. Li, and J. S. Vitter. Supporting incremental join queries on ranked inputs. In *VLDB*, 2001.

[25] C. H. Papadimitriou. *Computational Complexity*. AW, 1994.

[26] A. G. Parameswaran, H. Garcia-Molina, and J. D. Ullman. Evaluating, combining and generalizing recommendations with prerequisites. In *CIKM*, 2010.

[27] A. G. Parameswaran, P. Venetis, and H. Garcia-Molina. Recommendation systems with complex constraints: A course recommendation perspective. *TOIS*, 29(4), 2011.

[28] K. Schnaitter and N. Polyzotis. Evaluating rank joins with optimal cost. In *PODS*, 2008.

[29] K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *TODS*, 36(3), 2011.

[30] L. J. Stockmeyer. The polynomial-time hierarchy. *TCS*, 3(1):1–22, 1976.

[31] L. Valiant. The complexity of computing the permanent. *TCS*, 8(2):189 – 201, 1979.

[32] M. Y. Vardi. The complexity of relational query languages. In *STOC*, 1982.

[33] M. Wooldridge and P. E. Dunne. On the computational complexity of qualitative coalitional games. *Artif. Intell.*, 158(1):27–73, 2004.

[34] M. Xie, L. V. S. Lakshmanan, and P. T. Wood. Breaking out of the box of recommendations: from items to packages. In *RecSys*, 2010.