On the Complexity of Query Result Diversification

ON THE COMPLEXITY OF PACKAGE RECOMMENDATION PROBLEMS

TING DENG*, WENFEI FAN†, AND FLORIS GEERTS‡

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.

Key words. Recommendation problems, Complexity, Query relaxation

AMS subject classifications. 97R50, 03D15

1. Introduction and examples.

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 [36], teams of players [23] and various course combinations [20, 27, 28]. 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 [23] and course prerequisites [27]. Packages may have variable sizes subject to a cost budget, and are ranked by overall ratings of their items [36].

Recommendation systems are increasingly becoming an integral part of Web services [36], Web search [4], social networks [4], education software [28] and commerce services [3]. A number of systems have been developed for recommending items or packages, known as *points of interest (POI)* [36] (see [3, 4] for surveys). These systems use relational queries to specify selection criteria and compatibility constraints [2, 7, 20, 28, 36]. There has also been work on the complexity of computing POI recommendations [23, 27, 28, 36].

*School of Computer Science and Engineering, Beihang University, Beijing, China (dengting@act.buaa.edu.cn).

†School of Informatics, University of Edinburgh, Edinburgh EH8 9LE, UK (wenfei@inf.ed.ac.uk).

‡Department of Mathematics and Computer Science, University of Antwerp, Antwerp, Belgium (Floris.Geerts@ua.ac.be).

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.,* [27, 28]). 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 the precise causes for high complexity. 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

$$\mathsf{flight}(\mathsf{f\#}, \mathsf{From}, \mathsf{To}, \mathsf{DT}, \mathsf{DD}, \mathsf{AT}, \mathsf{AD}, \mathsf{Pr}) \text{ and } \mathsf{POI}(\mathsf{name}, \mathsf{city}, \mathsf{type}, \mathsf{ticket}, \mathsf{time}).$$

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 Edinburgh (EDI) to New York City (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, no more than 2 museums should be in a package, which is a compatibility constraint [36]. Moreover, plans should 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:

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

(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 in previous work.

(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 (Newark). This suggests that we need to help the user revise his or 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 visit too many of, 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. $\diamond$

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, 20, 27, 28, 36] 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 of package recommendations. 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 recommendation 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 the functions $\mathsf{cost}()$, $\mathsf{val}()$ 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 N is rated above B, *i.e.,* $\mathsf{val}(N) \geq B$? (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 recommendation 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 query languages $\mathcal{L}_Q$, all with built-in predicates $=, \neq, <, \leq, >, \geq$:

- conjunctive queries (CQ),
- union of conjunctive queries (UCQ),
- positive existential FO queries ($\exists$FO$^+$),
- non-recursive datalog queries (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 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.,* [36].

(c) The presence of compatibility constraints does not increase combined the complexity when the query language $\mathcal{L}_Q$ is FO, 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. Furthermore, these constraints also do *not* complicate the data complexity analyses. However, compatibility constraints increase combined complexity when $\mathcal{L}_Q$ is contained in $\exists$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,

check lower bounds; should be DP-hard?

respectively, even when selection criteria are given by an identity query. These give precise bounds for the problems studied in, *e.g.,* [36].

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 our 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 $\exists$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, *i.e.,* they are defined on the entire item space, 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, 23, 27, 28, 36]. A number of algorithms have been developed for recommending packages of a fixed size [6, 23] or variable sizes [27, 28, 36]. Compatibility constraints [23, 27, 28, 36] and budget restrictions [36] 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 [27, 28]. It was claimed that problems of forming a team with compatibility constraints [23] and the problem of finding packages that satisfy some budget restrictions (without compatibility constraints) [36] are NP-complete. In contrast, we establish the precise complexity of a variety of problems associated with POI recommendations (Table 8.2 and 8.2, Section 9). 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, 20, 24, 36]. Such views are expressed as relational queries, representing preferences or points of interest [2, 7, 20]. 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 [17]. Such queries either simply select tuples, or join and aggregate multiple inputs to find the top-k tuples, by possibly incorporating user preference information [20, 30]. A number of top-k query evaluation algorithms have been developed (*e.g.,*[13, 24, 29]; see [17] for a survey), as well as algorithms for incremental computation of ranked query results [10, 15, 25] 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, 14, 18, 19]. Several query generalization rules are introduced in [8], assuming that query acceptance conditions are monotonic. Heuristic query relaxation algorithms are developed in [14, 18]. The topic is also studied for top-k query answering [19]. 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.

This paper is extends [11] by including detailed proofs.

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 6. Query relaxation recommendations are studied in Section 7, followed by adjustment recommendations in Section 8. Section 9 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 the query languages considered in this work.

Item collections. Following [2, 7, 20, 27, 28, 36], we assume a database D consisting of items for selection. The database is specified with a relational schema $\mathcal{R}$ composed of 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 and is denoted by $\mathsf{dom}(R_i.A)$.

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 [28], travel plans including multiple POI [36], and teams of experts [23]. 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 [28, 36] 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 the constraint "no more than 2 museums" in a travel package N [36], we use the following Q_c that selects 3 distinct museums from N:

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

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 [27]. 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 [20, 23, 27, 28, 36].

Aggregate constraints. To specify aggregate constraints, we define a cost function and a rating function over packages, following [36]: (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 $\mathcal{N} = \big\{N_i \mid i \in [1, k]\big\}$ 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 6, we shall also consider a fixed size bound for $|N_i|$;

PREDEFINED polynomial?

(5) for all packages $N' \notin \mathcal{N}$ that satisfy 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 necessarily 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 $|D|$.

The *package recommendation problem* 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 [30], 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, the *item recommendation problem* 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 as imposed 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.

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 $p(\vec{x}\,) \leftarrow p_1(\vec{x}_1), \ldots, p_n(\vec{x}_n)$, 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 $p(\vec{x}) \leftarrow p_1(\vec{x}_1), \ldots, p_n(\vec{x}_n)$, for which the dependency graph may possibly be cyclic, *i.e.,* DATALOG is an extension of $\text{DATALOG}_{\mathsf{nr}}$ 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 4), and establish their complexity (Section 5.1).

3.1. Recommendation Problems. We investigate four problems, stated as follows, which are fundamental to computing package recommendations.

4. Deciding package selections. 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 (packages).*
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 $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$?

We start with the combined and data complexity of $\mathsf{RPP}(\mathcal{L}_Q)$ in the presence of compatibility constraints Q_c in Section 4.1 and 4.2, respectively, followed by the complexity of $\mathsf{RPP}(\mathcal{L}_Q)$ in the absence of Q_c in Section 4.3.

4.1. $\mathsf{RPP}(\mathcal{L}_Q)$ in the presence of Q_c (combined complexity). 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 $\text{DATALOG}_{\mathsf{nr}}$ 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 4.1. *For* $\mathsf{RPP}(\mathcal{L}_Q)$*, the combined complexity is*
- Π_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.*

Proof. We prove the combined complexity bounds of $\mathsf{RPP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ ranges over CQ, UCQ, $\exists\text{FO}^+$, $\text{DATALOG}_{\mathsf{nr}}$, FO and DATALOG.

▶ *When* $\mathcal{L}_Q$ *is CQ, UCQ or* $\exists\text{FO}^+$. It suffices to show that $\mathsf{RPP}(\mathcal{L}_Q)$ is Π_2^p-hard for CQ and is in Π_2^p for $\exists\text{FO}^+$.

Lower bound. To verify that $\mathsf{RPP}(\text{CQ})$ is Π_2^p-hard, we consider the *compatibility* problem. It is to determine, given Q, D, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C and a constant B, whether there exists a nonempty $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) > B$

and $Q_c(N, D) = \emptyset$. The proof of $\mathsf{RPP}(\text{CQ})$ consists of two parts. We first show that the compatibility problem is Σ_2^p-complete for CQ queries (see Lemma 4.2 below), then we verify that $\mathsf{RPP}(\text{CQ})$ is Π_2^p-hard by reduction from the complement of the compatibility problem. We first show the following lemma:

Lemma 4.2. *The combined complexity of the compatibility problem is Σ_2^p-complete for CQ queries.*

Proof. We show that the compatibility problem is in Σ_2^p by giving an NP algorithm that calls an NP oracle, as follows. The algorithm first guesses a package N and then verifies whether (a) $N \subseteq Q(D)$; (b) $Q_c(N, D) = \emptyset$; and (c) $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) > B$. When $\mathcal{L}_Q$ is CQ, UCQ or $\exists\text{FO}^+$, checking (a) and (b) requires NP and coNP, respectively. Indeed, for (a) the NP algorithm first guesses for each item $s \in N$, a CQ query Q_s from Q and a tableau from D for Q_s, and then checks whether these yield N. If so, the guess is accepted and the algorithm returns "yes". For (b) the coNP algorithm simply guesses a tuple t, a CQ query Q_t from Q and a tableau D for Q_t, and then checks whether these yield the tuple t. If so, the guess is accepted and the algorithm returns "no". In addition, verifying (c) requires PTIME. From this the Σ_2^p upper bound follows.

For the lower bound, we show that it is Σ_2^p-hard by reduction from the $\exists^*\forall^*$3DNF problem, which is known to be Σ_2^p-complete [31]. The $\exists^*\forall^*$3DNF problem is to decide, given a sentence $\varphi = \exists X \forall Y\, \psi(X, Y)$, whether φ is true. Here $X = \{x_1, \ldots, x_m\}$, $Y = \{y_1, \ldots, y_n\}$ and ψ is a disjunction $C_1 \vee \cdots \vee C_r$, where C_i is a conjunction of three literals defined in terms of variables in $X \cup Y$.

Given an instance $\varphi = \exists X \forall Y\, \psi(X, Y)$, we shortly define a database D, a query Q in CQ, a query Q_c in CQ for compatibility constraints, functions $\mathsf{cost}()$ and $\mathsf{val}()$, and two constants C and B, such that φ is true iff there exists a package $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) > B$, and $Q_c(N, D) = \emptyset$.

(1) The database D consists of four relations specified by schemas $R_{01}(X)$, $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$ and $R_\neg(A, \vec{A})$. Their instances are shown in Figure 4.1. More specifically, I_{01} encodes the Boolean domain, and $I_\vee$, $I_\wedge$ and $I_\neg$ encode disjunction, conjunction and negation, respectively, such that φ can be expressed in CQ in terms of these relations.

(2) We define a CQ query Q as $Q(\vec{x}\,) = R_{01}(x_1) \wedge \cdots \wedge R_{01}(x_m)$, where $\vec{x} = (x_1, \ldots, x_m)$. That is, $Q(\vec{x}\,)$ generates all truth assignments of X variables by means of Cartesian products of R_{01}.

(3) We define a CQ query Q_c as follows:

$$Q_c(b) = \exists \vec{x}\ \exists \vec{y}\ \big(R_Q(\vec{x}\,) \wedge Q_Y(\vec{y}\,) \wedge Q_\psi(\vec{x}, \vec{y}, b) \wedge\ b = 0\big).$$

Here R_Q is the schema of the result of $Q(D)$, and Q_Y generates all truth assignments of Y variables by means of Cartesian products of R_{01} in the same way as $Q(\vec{x}\,)$. Query Q_ψ in CQ encodes the truth value of $\psi(X, Y)$ for given truth assignments μ_X and μ_Y, in terms of $I_\vee$, $I_\wedge$ and $I_\neg$; it returns $b = 1$ if $\psi(X, Y)$ is satisfied by μ_X and μ_Y, and $b = 0$ otherwise. Intuitively, $Q_c(b) \neq \emptyset$ if for a given set $N \subseteq Q(D)$ that encodes a truth assignment μ_X for X, there exists a truth assignment of Y that makes $\psi(X, Y)$ false.

(4) We define $\mathsf{cost}(N) = |N|$ when $N \neq \emptyset$, *i.e.,* it counts the number of items in nonempty packages N, and define $\mathsf{cost}(\emptyset) = \infty$ otherwise. In addition, we use cost budget $C = 1$, *i.e.,* any recommended package N has exactly one item. Furthermore,

$$I_{01} = \begin{array}{|c|} X \\ \hline 1 \\ 0 \end{array} \qquad I_{\vee} = \begin{array}{|ccc|} B & A_1 & A_2 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{array} \qquad I_{\wedge} = \begin{array}{|ccc|} B & A_1 & A_2 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{array} \qquad I_{\neg} = \begin{array}{|cc|} A & \vec{A} \\ \hline 0 & 1 \\ 1 & 0 \end{array}$$

FIGURE 4.1. *Relation instances used in the lower bound proof of Theorem 4.1(1).*

we let $\mathsf{val}()$ be a constant function that assigns 1 to any package and set $B = 0$.

We next verify that φ is true iff there exists $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) > B$, and $Q_c(N, D) = \emptyset$.

$\boxed{\Rightarrow}$ First assume that φ is true. Then there exists a truth assignment μ_X^0 for X such that for all truth assignments μ_Y for Y, ψ is true. Let N consist of the tuple representing μ_X^0. Then Q_ψ does not return $b = 0$ for μ_X^0 and hence, $Q_c(N, D)$ is empty. Obviously, $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) > B$.

$\boxed{\Leftarrow}$ Conversely, assume that φ is false. Then for all truth assignment μ_X for X, there exists a truth assignment μ_Y for Y such that ψ is false for μ_X and μ_Y. Hence no matter how we select N, as long as N consists of a truth assignment of X, Q_ψ returns $b = 0$ and hence, $Q_c(N, D)$ is nonempty. Observe that the empty package $N = \emptyset$ cannot be recommended because $\mathsf{cost}(\emptyset) = \infty > C$. This completes the proof of the lemma. $\square$

We next show that $\mathsf{RPP}(\mathrm{CQ})$ is Π_2^p-hard by reduction from the complement of the compatibility problem. Given an instance Q, D, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, cost budget C and a constant B of the compatibility problem, we define a set $\mathcal{N}$ of packages, a function $\mathsf{val}'()$, and let $k = 1$. We show that there exists $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) > B$ and $Q_c(N, D) = \emptyset$ iff $\mathcal{N}$ is not a top-1 package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}'(), C)$.

To do this, we simply let $\mathcal{N}$ consist of a single package S, which is empty, *i.e.,* no recommendation is made. We define $\mathsf{val}'(N) = B$ if $N = S = \emptyset$, and $\mathsf{val}'(N) = \mathsf{val}(N)$ if $N \neq \emptyset$. These suffice. Indeed, first assume that $\mathcal{N}$ is a top-1 package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}'(), C)$. Then there exists no $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) > \mathsf{val}'(S) = B$, and $Q_c(N, D) = \emptyset$. Conversely, assume that $\mathcal{N}$ is not a top-1 package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}'(), C)$. Then there must exist an $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) > \mathsf{val}'(S) = B$, and $Q_c(N, D) = \emptyset$ by the definition of top-k package selections. Therefore, $\mathsf{RPP}(\mathrm{CQ})$ is Π_2^p-hard.

Upper bound. We present a Π_2^p algorithm to check whether $\mathcal{N}$ is a top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$, when $\mathcal{L}_Q$ is $\exists\mathrm{FO}^+$. The algorithm works as follows.

1. Test whether $\mathcal{N}$ is a valid package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$, in DP.
   (a) For each item s in $N_i \in \mathcal{N}$, guess a CQ query Q_s from Q and a tableau from D for Q_s. Check whether these tableaux yield $\mathcal{N}$. If so, continue; otherwise reject the guess and go back to step 1(a).
   (b) For each $N_i \in \mathcal{N}$, check whether $Q_c(N_i, D) = \emptyset$. If so, continue; otherwise return "no".
   (c) For each $N_i \in \mathcal{N}$, check whether $\mathsf{cost}(N_i) \leq C$. If so, continue; otherwise return "no".
   (d) Check whether $N_i \neq N_j$ for all $i \neq j$ and $i, j \in [1, k]$. If so, continue; otherwise return "no".

2. Test whether $\mathcal{N}$ is a top-k package selection (*i.e.,* there exists no valid package N such that (i) $N \notin \mathcal{N}$; and (ii) $\mathsf{val}(N) > \mathsf{val}(N_i)$ for some $N_i \in \mathcal{N}$) by the following Σ_2^p algorithm for the complement problem:
   (a) Guess polynomially many CQ queries from Q and for each CQ query, guess a tableau from D. These tableaux yield a package $N \subseteq Q(D)$. If $N \in \mathcal{N}$ then reject the guess and go back to step 2(a). Otherwise continue. Note that the polynomial bound on the number of queries is implied by the predefined polynomial bound on the size of packages as part of the input.
   (b) Check whether $Q_c(N, D) = \emptyset$. If so, continue; otherwise reject the guess and go back to step 2(a).
   (c) Check whether $\mathsf{cost}(N) \leq C$. If so, continue; otherwise reject the guess and go back to step 2(a).
   (d) Check whether $\mathsf{val}(N) > \mathsf{val}(N_i)$ for some $i \in [1, k]$. If so, return "no"; otherwise go back to step 2(a).

It is readily verified that step 1(a) is in NP and 1(b) is in coNP, along the same lines as in the proof of Lemma 4.2. In addition, steps 1(c) and 1(d) are in PTIME. Observe that step 1 is actually in DP. Indeed, step 1 decides the yes-instances of the intersection of two languages, $\big\{\mathcal{N} = \{N_1, \ldots, N_k\} \mid$ for each $N_i \in \mathcal{N}, N_i \subseteq Q(D)$, $\mathsf{cost}(N_i) \leq C$ and all N_i's are pairwise distinct$\big\}$ and $\big\{\mathcal{N} = \{N_1, \ldots, N_k\} \mid$ for each $N_i \in \mathcal{N}, Q_c(N_i, D) = \emptyset\big\}$, which are in NP and coNP, respectively. Step 2 is in Π_2^p since it consists of an Σ_2^p algorithm for deciding the complement problem, *i.e.,* to find a package that has higher rating than some package in $\mathcal{N}$. Indeed, step 2(a) is an NP step that calls step 2(b), which is a coNP oracle. Furthermore, steps 2(c) and 2(d) are in PTIME. Because $\mathsf{DP} \subseteq \Pi_2^p$, the algorithm is in Π_2^p.

► *When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO* . We first show that $\mathsf{RPP}(\mathcal{L}_Q)$ is PSPACE-hard for DATALOG$_{\mathsf{nr}}$ or FO. We then provide a PSPACE algorithm for $\mathsf{RPP}(\mathcal{L}_Q)$ that works for both DATALOG$_{\mathsf{nr}}$ and FO.

Lower bounds. We next show that RPP(DATALOG$_{\mathsf{nr}}$) and RPP(FO) are PSPACE-hard, by reduction from the membership problem for DATALOG$_{\mathsf{nr}}$ and FO, respectively. The membership problem to determine, given a query Q in DATALOG$_{\mathsf{nr}}$ or FO, a database D and a tuple t, whether $t \in Q(D)$. It is known that this problem is PSPACE-complete for queries in DATALOG$_{\mathsf{nr}}$ [34] and FO [33]. In the following, we let $\mathcal{L}_Q$ be DATALOG$_{\mathsf{nr}}$ of FO. Given an instance (Q, D, t) of the membership problem for $\mathcal{L}_Q$, we define a query Q' in $\mathcal{L}_Q$, Q_c as the empty query, function $\mathsf{cost}(N) = |N|$ when $N \neq \emptyset$ and $\mathsf{cost}(\emptyset) = \infty$, $C = 1$, and constant function $\mathsf{val}()$ that returns 1 on each package. Furthermore, we let $N = \{t\}$ and set $k = 1$. We show that $t \in Q(D)$ iff $\mathcal{N} = \{N\}$ is a top-1 package selection for $(Q', D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$. It suffices to define the query $Q'(\vec{x}\,) \leftarrow Q(\vec{x}\,), \vec{x} = t$ for Q in DATALOG$_{\mathsf{nr}}$, or $Q'(\vec{x}\,) = Q(\vec{x}\,) \wedge \vec{x} = t$ for Q in FO. Then it is easy to verify that $t \in Q(D)$ iff $\{t\}$ is a top-1 package selection for (Q', D, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C).

Upper bound. We give an NPSPACE algorithm for checking whether $\mathcal{N}$ is a top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$, when Q is in DATALOG$_{\mathsf{nr}}$ or FO. It works as follows.

1. Test whether $\mathcal{N}$ is a valid package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$, in PSPACE.
   (a) Check the following: for each item s in $N_i \in \mathcal{N}$, check whether $s \in Q(D)$, in PSPACE; for each $N_i \in \mathcal{N}$, check whether $\mathsf{cost}(N_i) \leq C$; and for all $i \neq j$ and $i, j \in [1, k]$, check whether $N_i \neq N_j$, in PTIME If all these conditions are

satisfied, continue; otherwise return "no".
(b) For each $N_i \in \mathcal{N}$, check whether $Q_c(N_i, D) = \emptyset$. If so, continue; otherwise return "no". This is done in PSPACE.

2. Test whether $\mathcal{N}$ is a top-k package selection, in NPSPACE.
(a) Guess a package N consisting of polynomially many tuples of the schema R_Q of query Q.
(b) Check the following: whether $N \subseteq Q(D)$ and $N \notin \mathcal{N}$, in PSPACE; and whether $\mathsf{cost}(N) \leq C$. If so, continue; otherwise reject the guess and go back to step 2(a).
(c) Check whether $Q_c(N, D) = \emptyset$, in PSPACE. If so, continue; otherwise reject the guess and go back to step 2(a).
(d) Check whether $\mathsf{val}(N) > \mathsf{val}(N_i)$ for some $i \in [1, k]$. If so, return "no"; otherwise go back to step 2(a).

Observe that steps 1(a), 1(b), 2(b) and 2(c) are in indeed in PSPACE since they rely on the membership problems for DATALOG$_{\mathsf{nr}}$ and FO. Including step 2(a), the overall algorithm is thus in NPSPACE = PSPACE when Q is in either DATALOG$_{\mathsf{nr}}$ or FO. Hence the problem is in PSPACE.

▶ *When $\mathcal{L}_Q$ is DATALOG* . We show that RPP(DATALOG) is EXPTIME-complete.

Lower bound. We show that RPP($\mathcal{L}_Q$) is EXPTIME-hard when $\mathcal{L}_Q$ is DATALOG, by reduction from the membership problem for DATALOG. The latter problem is to determine, given a DATALOG query Q, a database D and a tuple t, whether $t \in Q(D)$. It is known that this problem is EXPTIME-complete [33].

Given an instance (Q, D, t) of the membership problem for DATALOG, we define a DATALOG query Q', and Q_c as the empty query. We let $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$ and $\mathsf{cost}(\emptyset) = \infty$, $C = 1$, and let $\mathsf{val}()$ be a constant function. In addition, we set $k = 1$ and let $N = \{t\}$. Here Q' is the same as its counterpart defined in the proof for DATALOG$_{\mathsf{nr}}$ given above. It is readily verified that $t \in Q(D)$ iff N is a top-1 package selection for $(Q', D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$.

Upper bound. We given an EXPTIME algorithm to check whether $\mathcal{N}$ is a top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ when Q is in DATALOG:

1. Compute $Q(D)$, in EXPTIME.
2. For each $N_i \in \mathcal{N}$, check the following: (a) whether $N_i \subseteq Q(D)$, in EXPTIME, (b) whether $Q_c(N_i, D) = \emptyset$, in EXPTIME, and (c) whether $\mathsf{cost}(N_i) \leq C$, in PTIME. For all $i \neq j$ and $i, j \in [1, k]$, check (d) whether $N_i \neq N_j$, in PTIME. If all these conditions are satisfied, continue; otherwise return "no".
3. Enumerate all subsets of $Q(D)$ consisting of polynomially many tuples. For each such set N, do the following.
(a) Check (i) whether $N \notin \mathcal{N}$, in PTIME, (ii) whether $Q_c(N, D) = \emptyset$, in EXPTIME, and (iii) whether $\mathsf{cost}(N) \leq C$, in PTIME. If all these conditions are satisfied, continue; otherwise check the next set.
(b) Check whether $\mathsf{val}(N) > \mathsf{val}(N_i)$ for some $i \in [1, k]$, in PTIME. If so, return "no"; otherwise check the next set.
4. Return "yes" after all the sets are inspected.

This algorithm is in EXPTIME. In particular, step 3(a) is executed exponentially many times, and still takes EXPTIME in total. Hence the problem is in EXPTIME.

This completes the proof of Theorem 4.1. □

4.2. RPP($\mathcal{L}_Q$) **in the presence of** Q_c **(data complexity).** We next consider the data complexity of RPP($\mathcal{L}_Q$).

THEOREM 4.3. *For* $\mathsf{RPP}(\mathcal{L}_Q)$*, 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. We first show that $\mathsf{RPP}(\mathcal{L}_Q)$ is already coNP-hard when $\mathcal{L}_Q$ is CQ.

Lower bound. We show that RPP(CQ) is coNP-hard as follows. First we prove that the data complexity of the compatibility problem is NP-complete for CQ queries, and then we verify that RPP(CQ) is coNP-hard by reduction from the complement of the compatibility problem, defined in the proof of the combined complexity of RPP(CQ) in Theorem 4.1.

LEMMA 4.4. *The data complexity of the compatibility problem is* NP*-complete for CQ queries.*

Proof. The NP upper bound for the compatibility problem is readily verified by simply guessing a package and testing whether it satisfies the condition. We verify the NP lower bound by reduction from 3SAT, which is known to be NP-complete (cf. [26]). An instance φ of 3SAT is a formula $C_1 \wedge \cdots \wedge C_r$ in which each clause C_i is a disjunction of three variables or negations thereof taken from $X = \{x_1, \ldots, x_n\}$. Given φ, 3SAT is to decide whether φ is satisfiable, *i.e.,* whether there exists a truth assignment for variables in X that satisfies φ. We define a database D, a query Q in CQ, a query Q_c in CQ for compatibility constraints, functions cost() and val(), and constants C and B. We show that φ is satisfiable iff there exists $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) > B$, where Q is fixed.

(1) The database D is defined over a single relation $R_C(\mathsf{cid}, L_1, V_1, L_2, V_2, L_3, V_3)$. Its corresponding instance I_C consists of the following set of tuples. For each $i \in [1, r]$, let $C_i = \ell_1^i \vee \ell_2^i \vee \ell_3^i$. For any possible truth assignment μ_i of variables in the literals in C_i that make C_i true, we add a tuple $(i, x_k, v_k, x_l, v_l, x_m, v_m)$, where $x_k = \ell_1^i$ in case $\ell_1^i \in X$ and $x_k = \vec{\ell_1^i}$ in case $\ell_1^i = \vec{x}_k$. We set $v_k = \mu_i(x_k)$; similarly for x_l, x_m and v_l and v_m.

(2) We take Q as the identity query. We define Q_c to be the empty query.

(3) We define for each package N, $\mathsf{val}(N) = |N|$ and set $B = r - 1$. That is, any package must consist of at least r tuples.

(4) For each package N, we define $\mathsf{cost}(N) = 1$ if there exists no two distinct tuples in N which have the same cid value or have different values for a variable appearing in both of them. Furthermore, for any other N, we define $\mathsf{cost}(N) = 2$. We set $C = 1$.

We next verify that φ is true iff there exists an $N \subseteq Q(D)$ such that $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) > B$.

$\boxed{\Rightarrow}$ First assume that φ is satisfiable. Then there exists a truth assignment μ_X^0 for X that satisfies ψ, *i.e.,* every clause C_j of φ is true by μ_X^0. Let N consist of r tuples from D, one for each clause, in which the values for the variables correspond to μ_X^0. Then $\mathsf{val}(N) = r > B$ and $\mathsf{cost}(N) = 1 \leq C$.

$\boxed{\Leftarrow}$ Conversely, assume that φ is not satisfiable. Suppose by contradiction that there exists an $N \subseteq Q(D)$ with $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) > B$. Then N consists of r tuples and since $\mathsf{cost}(N) \leq C$ one can construct a truth assignment μ_N for X that makes all clauses in φ true. A contradiction. This completes the proof of the Lemma. ☐

Next, one can verify that RPP(CQ) is coNP-hard using the same argument given earlier for the combined complexity of RPP(CQ), by using $k = 1$.

Upper bound. We show that $\mathsf{RPP}(\mathcal{L}_Q)$ is in coNP (data complexity) for all the query languages considered. Observe that the algorithm for $\mathsf{RPP}(\mathrm{FO})$ (combined complexity) works correctly for all those query languages. It suffices to observe that when data complexity is concerned, steps 1(a) and 1(b) are in PTIME, and similarly, steps 2(b) and 2(c) are in PTIME. This follows from the fact that the data complexity of the membership problem for $\mathcal{L}_Q$ is in PTIME, even when $\mathcal{L}_Q$ is FO or DATALOG. Since step 2 involves guessing a package and deciding the existence of package with higher rating than some package in $\mathcal{N}$ (*i.e.,* the complement problem), the algorithm is thus in coNP overall.

This completes the proof of Theorem 4.1. □

4.3. $\mathsf{RPP}(\mathcal{L}_Q)$ **in the absence of** Q_c**.** 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 $\mathrm{DATALOG}_{\mathsf{nr}}$ 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 4.5. *In the absence of* Q_c, $\mathsf{RPP}(\mathcal{L}_Q)$ *is*

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

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

Proof. We next revisit the combined complexity and data complexity of $\mathsf{RPP}(\mathcal{L}_Q)$ in the special case when the compatibility constraints Q_c are absent,*i.e.,* when Q_c is the empty query.

(1) Combined complexity. We first show the combined complexity results at first.

When $\mathcal{L}_Q$ **is CQ, UCQ or** $\exists\mathbf{FO}^+$. It suffices to verify that $\mathsf{RPP}(\mathcal{L}_Q)$ is DP-hard for CQ and is in DP for $\exists\mathrm{FO}^+$.

Lower bound. We show that $\mathsf{RPP}(\mathrm{CQ})$ is DP-hard by reduction from SAT-UNSAT, which is known to be DP-complete (cf. [26]). An instance of SAT-UNSAT is a pair of 3SAT instances (φ_1, φ_2), where φ_1 is a 3SAT instance over $X = \{x_1, \ldots, x_m\}$, and φ_2 is a 3SAT instance over $Y = \{y_1, \ldots, y_n\}$. Given (φ_1, φ_2), SAT-UNSAT is to determine whether φ_1 is satisfiable and φ_2 is not satisfiable (recall the definition of 3SAT from the proof of Theorem 4.1 for the $\mathrm{DATALOG}_{\mathsf{nr}}$ case).

Given an instance (φ_1, φ_2) of SAT-UNSAT, we construct a query Q, a database D, and Q_c as the empty query, We define $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$ and $\mathsf{cost}(\emptyset) = \infty$. We set $C = 1$, and define a rating function $\mathsf{val}()$ and a package N. In addition, we set $k = 1$. In other words, we consider top-1 package selections in which a package consists of one tuple. We show that $\mathcal{N} = \{N\}$ is a top-1 package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ iff φ_1 is satisfiable and φ_2 is not satisfiable.

(1) The database D consists of four relations as shown in Figure 4.1, specified by schemas $R_{01}(X)$, $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$ and $R_\neg(A, \vec{A})$ given in the proof of Theorem 4.1. The formulas φ_1 and φ_2 can be expressed in CQ in terms of these relations.

(2) We define the CQ query Q as follows:

$$Q(b,b') \;=\; \exists\vec{x}\,\exists\vec{y}\,\big(Q_X(\vec{x}\,)\;\wedge Q_{\varphi_1}(\vec{x},b)\;\wedge\;Q_Y(\vec{y}\,)\wedge Q_{\varphi_2}(\vec{y},b')\big).$$

Here $\vec{x}=(x_1,\ldots,x_m)$ and $\vec{y}=(y_1,\ldots,y_n)$. Furthermore, the queries $Q_X(\vec{x}\,)$ and $Q_Y(\vec{y}\,)$ generate all truth assignments of X variables and Y variables for φ_1 and φ_2, respectively, by means of Cartesian products of R_{01}. The sub-query $Q_{\varphi_1}(\vec{x},b)$ encodes the truth value of φ_1 for a given truth assignment μ_X such that $b=1$ if μ_X satisfies φ_1, and $b=0$ otherwise; similarly for $Q_{\varphi_2}(\vec{y},b')$ and φ_2. Obviously, $Q_{\varphi_1}(\vec{x},b)$ and $Q_{\varphi_2}(\vec{y},b')$ can be expressed in CQ in terms of $R_\vee$, $R_\wedge$, and $R_\neg$. Observe that given D, the answer to Q in D is a subset of $\big\{(1,0),\;(1,1),\;(0,0),\;(0,1)\big\}$.

(3) It suffices to define $\mathsf{val}()$ on singleton sets. We define $\mathsf{val}(\{(1,0)\})=2$, $\mathsf{val}(\{(1,1)\})$ $=\mathsf{val}(\{(0,1)\})=3$ and $\mathsf{val}(\{(0,0)\})=1$. Furthermore, we let N consist of the single tuple $(1,0)$.

We show that $\mathcal{N}=\{N\}$ is a top-1 package selection for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C)$ iff φ_1 is satisfiable and φ_2 is not satisfiable.

$\boxed{\Rightarrow}$ First assume that N is a top-1 package selection for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C)$. Then $(1,1)$ and $(0,1)$ cannot be in $Q(D)$. Therefore, there exists a truth assignment for X making φ_1 true and moreover, there exist no assignments for Y making φ_2 true, *i.e.,* φ_1 is satisfiable and φ_2 is not satisfiable.

$\boxed{\Leftarrow}$ Conversely, assume that φ_1 is satisfiable and φ_2 is not satisfiable. Then by the definition of Q, either $Q(D)=\big\{(1,0),(0,0)\big\}$ or $Q(D)=\big\{(1,0)\big\}$. Hence $N=\big\{(1,0)\big\}$ is a top-1 package selection for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C)$ by the definition of $\mathsf{val}()$ given above.

Upper bound. Consider the algorithm for RPP(FO) given in the proof of Theorem 4.1 for FO. Obviously, the algorithm can work here. Note that steps 1 and 2 are in NP and coNP, respectively, when the Q_c test is not needed (*i.e.,* without steps 1(b) and 2(c)). Thus RPP is in DP.

When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$, FO or DATALOG. We show that the absence of Q_c does not affect the combined complexity. For the lower bounds, it suffices to observe that the proofs of the PSPACE and $\mathsf{EXPTIME}$ lower bounds given in the proof of Theorem 4.1 for DATALOG$_{\mathsf{nr}}$, FO and DATALOG cases do no use compatibility constraints. Clearly, the PSPACE and $\mathsf{EXPTIME}$ algorithms given in that proof remain valid after the Q_c test is removed. As a consequence, $\mathsf{RPP}(\mathcal{L}_Q)$ remains PSPACE-complete when $\mathcal{L}_Q$ is either DATALOG$_{\mathsf{nr}}$ or FO, and $\mathsf{EXPTIME}$-complete when $\mathcal{L}_Q$ is DATALOG, in the absence of compatibility constraints.

(2) Data complexity. We show that the data complexity remains coNP-complete in the absence of Q_c. Indeed, for the upper bound, the algorithm developed for the data complexity analysis in the presence of Q_c (in the proof of Theorem 4.1) remains valid when Q_c is the empty query. Hence the data complexity of $\mathsf{RPP}(\mathcal{L}_Q)$ remains in coNP. In addition, the proof of Lemma 4.4 already shows that the compatibility problem for CQ is NP-hard in the absence of Q_c and when Q is fixed. Since RPP(CQ) can be reduced from the complement of the compatibility problem, RPP(CQ) is coNP-hard in the absence of Q_c.

This completes the proof of Theorem 4.5. □

5. Computing. 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 recommendation problem (packages).*
INPUT:	$D, Q, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k$ as in the problem 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).

5.1. 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.

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

THEOREM 5.1. *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$^+$;*
- $\mathsf{FPSPACE(poly)}$*-complete if $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO;*
- $\mathsf{FEXPTIME(poly)}$*-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. [26]), and $\mathsf{FP}^{\Sigma_2^p}$ is the class of all functions computable by a PTIME 2-alternating max-min Turing machine [21]. 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) [22]. 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 [36], we need at least CQ; for course combination constraints of [20, 27, 28], 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 4.1, 4.5 and 5.1 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 $\exists\mathrm{FO}^+$are simplified. This is consistent with Theorem 4.5.

Proof. Below we first prove the combined complexity results of $\mathsf{FRP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ ranges over CQ, UCQ, $\exists\mathrm{FO}^+$, $\mathrm{DATALOG}_{\mathsf{nr}}$, FO and DATALOG. We then show their data complexity. Finally, we revisit the combined complexity and data complexity of $\mathsf{FRP}(\mathcal{L}_Q)$, when the compatibility constraints Q_c are absent.

(1) Combined complexity. We first verify the combined complexity bounds.

When $\mathcal{L}_Q$ is CQ, UCQ or $\exists\mathbf{FO}^+$. It suffices to show that $\mathsf{FRP}(\mathcal{L}_Q)$ is $\mathsf{FP}^{\Sigma_2^p}$-hard when $\mathcal{L}_Q$ is CQ and is in $\mathsf{FP}^{\Sigma_2^p}$ when $\mathcal{L}_Q$ is $\exists\mathrm{FO}^+$.

Lower bound. We show that $\mathsf{FRP}(\mathcal{L}_Q)$ is $\mathsf{FP}^{\Sigma_2^p}$-hard by reduction from the MAXIMUM Σ_2^p problem, which is known to be complete for the class of functions computable by a polynomial 2-alternating max-min Turing machine. This class of functions is often denoted by Σ_2^{MM} [21]. It is easily verified that a complete problem for Σ_2^{MM} is also complete for the class of functions that are computable by a $\mathsf{P}^{\Sigma_2^p}$ Turing machine, or in other words, the class of $\mathsf{FP}^{\Sigma_2^p}$ computable functions [21].

An instance of MAXIMUM Σ_2^p consists of a universally quantified formula $\varphi(X) = \forall Y\, \psi(X,Y)$, where $X = \{x_1,\ldots,x_m\}$, $Y = \{y_1,\ldots,y_n\}$ and ψ is a instance of 3DNF over the variables in $X \cup Y$. That is, ψ is a disjunction $C_1 \vee \cdots \vee C_r$, where each clause C_i is a conjunction of three literals over $X \cup Y$. Given φ, MAXIMUM Σ_2^p is to find the truth assignment μ_X^{last} of X that makes φ true and comes last in the lexicographical ordering on m-ary binary tuples, if it exists.

Given a MAXIMUM Σ_2^p instance φ, we construct a database D, a query Q, a query Q_c for compatibility constraints, functions $\mathsf{cost}()$ and $\mathsf{val}()$, and a cost budget C. In particular, The database D consists of four relations as shown in Figure 4.1, specified by schemas $R_{01}(X)$, $R_\vee(B,A_1,A_2)$, $R_\wedge(B,A_1,A_2)$ and $R_\neg(A,\vec{A})$ given in the proof of Theorem 4.1. Furthermore, we set $k = 1$, define $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$, $\mathsf{cost}(\emptyset) = \infty$

and set $C = 1$.That is, only packages consisting of a single tuple can be recommended. The query Q and compatibility constraint Q_c are as given in the proof of Lemma 4.2. That is, Q returns all truth assignments of X and a package N consists of a single tuple t such that (i) t represents a truth assignment μ_X of X; and (ii) $Q_c(\{t\}, D) = \emptyset$, enforcing that μ_X makes φ true. Finally, for a tuple $t = (x_1, \ldots, x_m)$, we define $\mathsf{val}(\{t\})$ to be t, denoting the value it encodes in binary.

We next show that $\{N\}$ is a top-1 package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$, where N consists of a single tuple $t = (\vec{x}\,)$, iff the truth assignment μ_X encoded by $\vec{x}$ coincides with μ_X^{last}.

$\boxed{\Rightarrow}$ Assume that $t = (\vec{x}\,)$ is a top-1 package selection. Then for each $t' \in Q(D)$, $\mathsf{val}(\{t\}) \geq \mathsf{val}(\{t'\})$. As a result, the truth assignment μ_X determined by $\vec{x}$ makes φ true and has the highest rating over all such truth assignments. It suffices to observe that for any two truth assignments μ_X and μ'_X of X, μ_X comes after μ'_X in the lexicographical ordering if and only if $\mathsf{val}(t) > \mathsf{val}(t')$, where t represents μ_X and t' represents μ'_X. As a consequence, $\mu_X = \mu_X^{last}$. Note that when no top-1 package selection exists, φ is not satisfiable and hence only the empty package could be recommended. However, by the definition of the cost function, $\mathsf{val}(\emptyset) > C$, and hence no recommendation will be made.

$\boxed{\Leftarrow}$ Conversely, assume that φ is satisfiable and consider μ_X^{last}. Let t be the tuple that represents μ_X^{last}. Then by the same argument as above, $\{t\}$ will be the top-1 package by the definition of $\mathsf{val}()$. If φ is not satisfiable, then no recommendation will be returned, as argued above.

Upper bound. We show that when $\mathcal{L}_Q$ is $\exists\mathrm{FO}^+$, $\mathsf{FRP}(\mathcal{L}_Q)$ is in $\mathsf{FP}^{\Sigma_2^p}$ by providing an $\mathsf{FP}^{\Sigma_2^p}$-algorithm that on input $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$, returns a top-$k$ package selection $\mathcal{N} = \{N_i \mid i \in [1, k]\}$, if it exists. That is, we develop an algorithm that runs in polynomial time with access to an Σ_2^p-oracle.

Let $\mathsf{EXISTPACK}^{\geq}(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, \mathcal{N}, N, v)$ be a procedure that returns "yes" if there exists a package $N \subseteq Q(D)$ such that $Q_c(N, D) = \emptyset$, $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) \geq v$ and N is not equal to any package already in $\mathcal{N}$. It is easily verified that this is an Σ_2^p procedure. Indeed, one simply needs to guess polynomially many tuples from D to fill in the tableaux of CQ queries obtained from Q and verify whether these produce a package N that satisfies the conditions. Since checking the conditions requires calls to an NP and a coNP oracle, the complexity of the procedure is indeed in Σ_2^p. Given this procedure, the algorithm that returns a top-k package selection $\mathcal{N} = \{N_1, \ldots, N_k\}$, if it exists, works as follows:

1. Let $B_{\max} = 2^{p(n)}$, where $p(n)$ is a polynomial that represents the length of the encoding of D, Q and that takes into account that $\mathsf{cost}()$ and $\mathsf{val}()$ are PTIME functions. Any $\mathsf{val}()$-rating of packages in $Q(D)$ lies within the interval $[0, B_{\max}]$.

2. Let $\mathcal{N} = \emptyset$ and $\ell = 1$.

3. While $\ell < k + 1$ do the following:
   (a) Perform a binary search over the interval $[0, B_{\max}]$ to find the maximal value $B \in [0, B_{\max}]$ such that there exists a valid package $N \subseteq Q(D)$ with $\mathsf{val}(N) = B$ and N is not equal to any package in $\mathcal{N}$, Clearly, B can be found in this way by making $\log(2^{p(n)}) = p(n)$ calls to the Σ_2^p oracle $\mathsf{EXISTPACK}^{\geq}($ Q, D, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C, $\mathcal{N}$, $N, v)$. Such a value will always be found, unless no k distinct packages exist. In that case, we return the empty set and terminate

the algorithm. Otherwise, we continue.

(b) Given B, we know that there exists a package N such that $N \subseteq Q(D)$, $Q_c(N, D) = \emptyset$, $\mathsf{cost}(N) \leq C$, $\mathsf{val}(N) = B$ and N is distinct from any other package already in $\mathcal{N}$. It remains to find such an *optimal* package N. To do this, we proceed as follows: Let $s(n)$ be a polynomial that bounds the size of packages, *i.e.,* any package consists of at most $s(|D|)$ tuples. Let $N = \emptyset$. We will add tuples to N, one at a time, hereby guaranteeing that N can grow to an *optimal* package with $\mathsf{val}(N) = B$. Let $l = 1$.

(c) While $l < s(|D|) + 1$ do the following:

(i). We first check whether N is optimal. That is, whether $\mathsf{val}(N) = B$. If so, we add N to $\mathcal{N}$ and set $B_{\max} = B$. No further tuples need to be added to N in this case and we go to step 3 and let $\ell = \ell + 1$. That is, we extend $\mathcal{N}$ with one package (*i.e.,* N) and continue with adding a next package (if needed).

(ii). Otherwise, if $\mathsf{val}(N) < B$ then we still need to add tuples to N, as follows: Let $m = \mathsf{arity}(R_Q)$ and $n = |\mathsf{adom}(Q, D)|$, where $\mathsf{arity}(R_Q)$ denotes the arity of the output schema of Q and $\mathsf{adom}(Q, D)$ is the set of constants appearing in D or Q. Denote by $\mathcal{C} = (c_{ij})$ the $m \times n$ array, where c_{ij} is the jth constant in some arbitrary ordering of $\mathsf{adom}(Q, D)$. We next show how to transform $\mathcal{C}$ into an $m \times n$ array $\mathcal{D} = (d_{ij})$ such that for each $i \in [1, m]$, there exists a unique j such that $d_{ij} = c_{ij}$, whereas for $j' \neq j$, $d_{ij'} = \sqcup$. The semantics of $\mathcal{D}$ is that we can derive a tuple $s \in Q(D)$ such that for each $i \in [1, m]$, $s[i]$ takes the unique value in the ith row of $\mathcal{D}$ different from $\sqcup$. This tuple s will be added to N. We next show how $\mathcal{D}$ is constructed from $\mathcal{C}$. Let $i = 1$, $j = 1$ and $\mathcal{D} = \mathcal{C}$.

(iii). While $i < m + 1$ and $j < n + 1$ do the following:

(A). Let $c = c_{ij}$ and consider the rating function $\mathsf{val}_{c,i,N}$ such that $\mathsf{val}_{c,i,N}(N') = B - 1$ if $N \subsetneq N'$ and N' contains a tuple $s \notin N$ with $s[i] = c$; and $\mathsf{val}_{c,i,N}(N') = \mathsf{val}(N')$ otherwise.

(B). We next call the oracle. If $\mathsf{EXISTPACK}^{\geq}(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}_{c,i,N}, C, \mathcal{N}, N, B)$ returns true, then this implies that there exists a package N', which is larger than N, $N' \subseteq Q(D)$, $\mathsf{cost}(N') \leq C$ and $\mathsf{val}_{c,i,N}(N') = B$, and N' is not equal to any package already in $\mathcal{N}$. That is, there exists an optimal N' for which we can safely assume (by the definition of $\mathsf{val}_{c,i,N}$) that $N' \setminus N$ consists of tuples that do not carry value c in their ith attribute. In other words, we can forget about any package N' such that $N' \setminus N$ carries tuples with constant c. We thus change d_{ij} to $\sqcup$ (indicating that we can ignore this value when looking for an optimal package) and set $j = j + 1$. That is, we move to the next constant in $\mathsf{adom}(Q, D)$. Furthermore, we replace $\mathsf{val}()$ with $\mathsf{val}_{c,i,N}$, enforcing optimal extensions of N to carry values different from c.

(C). On the other hand, if $\mathsf{EXISTPACK}^{\geq}(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}_{c,i,N}, C, \mathcal{N}, N, B)$ returns false, then any optimal package N' must contain a tuple in $N' \setminus N$ that carries c in its ith attribute. In this case, we cannot disregard constant c when looking for optimal packages and thus we do not change $d_{ij} = c$. However, we do so for all other values $d_{ij'}$ with $j' > j$. We then set $i = i + 1$ (we move to the next attribute) and define $\mathsf{val}(N') = B-1$ in case that N' does *not* contain a tuple in $N' \setminus N$ which carries c in its ith attribute. For all other N' we keep the original $\mathsf{val}(N')$ value. In other words, the choice of rating function will limit our search for an optimal package by only looking for packages that carry an additional tuple with a specific constant (c) in its ith attribute.

(iv). When all attributes and values are considered, we know the following: The array $\mathcal{D}$ has a unique entry that is set to a constant different from $\sqcup$ in each of its rows. Let s be the tuple obtained by combining these constants. Furthermore, by the construction, we know that there exists an optimal package N' such that $N \subsetneq N'$ and $N' \setminus N$ contains tuple s. We thus add s to N, reset $\mathsf{val}()$ to the original rating function and increase l by 1. That is, we extend N with one tuple and look for the next tuple to add.

4. If successful, then k packages have been added to $\mathcal{N}$ in the order of decreasing rating value. We return $\mathcal{N}$.

The above algorithm runs in $\mathsf{FP}^{\Sigma_2^p}$. Indeed, the Σ_2^p oracle is called polynomially many times. We next argue for the correctness of the algorithm. First, observe that the algorithm returns a set of packages $\mathcal{N} = \{N_1, \ldots, N_k\}$ only if a top-k package selection exists. Second, the algorithm finds packages N_i in the decreasing order of their $\mathsf{val}()$-value. Indeed, in step (a), the binary search guarantees that the maximal $\mathsf{val}()$-value is selected for which there still exists a package in $Q(D)$ that differs from all previously constructed packages and that satisfies the cost budget constraint; step (c) then constructs such a optimal package in $Q(D)$ with the maximal $\mathsf{val}()$-value.

When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO. We show that $\mathsf{FRP}(\mathcal{L}_Q)$ is $\mathsf{FPSPACE(poly)}$- complete when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO.

Lower bound. We show that $\mathsf{FRP}(\mathcal{L}_Q)$ is $\mathsf{FPSPACE(poly)}$-hard when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$. Indeed, consider a function g in $\mathsf{FPSPACE(poly)}$. Here we mean by $\mathsf{FPSPACE(poly)}$ the class of functions computable by a PSPACE Turing machine which represents the output on the working tape as well [22]. Since g is in $\mathsf{FPSPACE(poly)}$, one can decide in PSPACE whether the ith bit of $g(\vec{x})$ is 1, for a given input $\vec{x}$. We show that testing whether the ith bit of $g(\vec{x})$ is set to 1 reduces to testing whether a package N_i, consisting of a single tuple t_i is the top-1 package selection for $(Q_i, D_i, Q_c^i = \emptyset, \mathsf{cost}_i(), \mathsf{val}_i(), C_i)$, where Q_i is a DATALOG$_{\mathsf{nr}}$ query, D_i is a database, $\mathsf{cost}_i()$ and $\mathsf{val}_i()$ are functions and C_i is a constant. This suffices. For if it holds, then we can compute $g(\vec{x})$ by identifying each bit of its output, and hence all the functions in $\mathsf{FPSPACE(poly)}$ are reduced to computing top-1 package selections for DATALOG$_{\mathsf{nr}}$. From these it follows that FRP(DATALOG$_{\mathsf{nr}}$) is $\mathsf{FPSPACE(poly)}$-hard.

To see how to determine the ith bit of $g(\vec{x})$, we first observe that due to the PSPACE-completeness of Q3SAT, there exists a Q3SAT instance φ_i such that φ_i is true iff the ith bit of $g(\vec{x})$ is set to 1. Next, note that the package recommendation problem for DATALOG$_{\mathsf{nr}}$ is PSPACE-complete by reduction from Q3SAT as shown in the proof of Theorem 4.1. Consider the Q3SAT instance φ_i. A minor modification of the proof of Theorem 4.1 for DATALOG$_{\mathsf{nr}}$ results in a fixed database D_i, a DATALOG$_{\mathsf{nr}}$ query Q_i (encoding φ_i), the cost function $\mathsf{cost}_i(N) = |N|$ if $N \neq \emptyset$ and $\mathsf{cost}_i(\emptyset) = \infty$, $C_i = 1$, and a rating function $\mathsf{val}_i()$, such that the tuple $\{(1)\}$ is the top-1 package selection for $(Q_i, D_i, Q_c^i = \emptyset, \mathsf{cost}_i(), \mathsf{val}_i(), C_i)$ if φ_i evaluates to true, and tuple $\{(0)\}$ is the top-1 package selection otherwise. More specifically, the construction of Q_i always adds $\{(0)\}$ to $Q_i(D_i)$ but adds the tuple $\{(1)\} \in Q_i(D_i)$ iff φ_i is true. In addition, $\mathsf{val}_i(\{(0)\}) = 1$ and $\mathsf{val}_i(\{(1)\}) = 2$. As a consequence, $\{(1)\}$ will be recommended if it is present in $Q_i(D)$. Since $g(\vec{x})$ is of polynomial size, a polynomial number of instances $(Q_i, D_i, Q_c^i = \emptyset, \mathsf{cost}_i(), \mathsf{val}_i(), C_i)$ are needed to decide all the bits of $g(\vec{x})$. Assume that we need $p(|\vec{x}|)$ bits. Then, consider the DATALOG$_{\mathsf{nr}}$ query Q that combines all Q_i such that it outputs tuples of arity $p(|\vec{x}|)$, where the ith attribute corresponds to the output of $Q_i(D_i)$. Furthermore, observe that we have a constant database D_i that does not depend on i; similarly for $\mathsf{cost}_i()$ and C_i. We let $D = D_i$ and $\mathsf{cost}() = \mathsf{cost}_i()$ for some i, and set $C = 1$. Finally, we define $\mathsf{val}(\{s\}) = \sum s_i 2^i$ in binary, *i.e.,* $\mathsf{val}(\{s\}) = (s_{p(|\vec{x}|)}, \ldots, s_0)$, where s_i denotes the ith attribute value of s. In this way, tuples $s \in Q(D)$ encode values and it is now readily verified that the tuple representing $g(\vec{x})$ is the top-1 package selection for $(Q, D, Q_c = \emptyset, \mathsf{cost}(), \mathsf{val}(), C)$.

When $\mathcal{L}_Q$ is FO, we use a similar proof as in the previous case, but using FO formulas instead of queries in DATALOG$_{\mathsf{nr}}$, and by modifying the reduction from the membership problem for FO as given in the proof of Theorem 4.1 for FO.

Upper bound. We provide an FPSPACE(poly) algorithm to find a top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ when Q is in DATALOG$_{\mathsf{nr}}$ or FO. The algorithm is a variation of the algorithm given for FRP($\exists$FO$^+$) above. Indeed, it suffices to observe that the Σ_2^p-oracle used in that algorithm can be replaced by a PSPACE oracle, when the queries and compatibility constraints involved are in DATALOG$_{\mathsf{nr}}$ or FO. As a consequence, the modified algorithm will make polynomially many calls to a PSPACE oracle and is therefore in FPSPACE(poly).

When $\mathcal{L}_Q$ is DATALOG. We show that FRP($\mathcal{L}_Q$) is FEXPTIME(poly)-complete when $\mathcal{L}_Q$ is DATALOG.

Lower bound. We show that FRP(DATALOG) is FEXPTIME(poly)-hard. The proof is along the same lines as the FPSPACE(poly)-hardness proof given earlier.

Given a function h in FEXPTIME(poly), it is in EXPTIME to decide whether the ith bit of $h(\vec{x})$ is 1, for a given input $\vec{x}$. Furthermore, $h(\vec{x})$ is of size polynomial in $|\vec{x}|$. We show that testing whether the ith bit of $h(\vec{x})$ is set to 1 reduces to testing whether a package N_i, consisting of a single tuple t_i, is the top-1 package selection for $(Q_i, D_i, Q_c^i = \emptyset, \mathsf{cost}_i(), \mathsf{val}_i(), C_i)$, where Q_i is a DATALOG query, D_i is a database, $\mathsf{cost}_i()$ and $\mathsf{val}_i()$ are functions and C_i is a constant. To do this, we first consider an instance (Q_i', D_i', t_i') of the membership problem for DATALOG such that $t_i' \in Q_i'(D_i')$ iff the ith bit of $h(\vec{x})$ is set to 1. Since the membership problem for DATALOG is EXPTIME-complete, such an instance always exists. Next, a minor modification of the proof of Theorem 4.1 for DATALOG results in a database D_i (encoding D_i'), a DATALOG query Q_i (encoding Q_i' and t_i'), the cost function $\mathsf{cost}_i(N) = |N|$ if $N \neq \emptyset$ and $\mathsf{cost}_i(\emptyset) = \infty$, $C_i = 1$, and a rating function $\mathsf{val}_i()$, such that the tuple $\{(1)\}$ is the top-1 package selection for $(Q_i, D_i, Q_c^i = \emptyset, \mathsf{cost}_i(), \mathsf{val}_i(), C_i)$ if $t_i' \in Q_i'(D_i')$ (*i.e.,* the membership problem for DATALOG), and tuple $\{(0)\}$ is the top-1 package selection otherwise (we enforce (0) to be always in $Q_i(D_i)$). Since $h(\vec{x})$ is of polynomial size, a polynomial number of instances $(Q_i, D_i, Q_c^i = \emptyset, \mathsf{cost}_i(), \mathsf{val}_i(), C_i)$ are needed to decide all the bits of $h(\vec{x})$. Assume that we need $p(|\vec{x}|)$ bits and consider the DATALOG query Q that combines all Q_i such that it outputs tuples of arity $p(|\vec{x}|)$, where the ith attribute denotes the output of $Q_i(D_i)$. Furthermore, we let D consist of the union of all D_i's in which we keep the D_i's distinct by adding an identifier to each tuple. We let $\mathsf{cost}()$, $\mathsf{val}()$ and C as in the previous proof. One can verify that the tuple denoting $h(\vec{x})$ is the top-1 package selection for $(Q, D, Q_c = \emptyset, \mathsf{cost}(), \mathsf{val}(), C)$.

Upper bound. We provide an $\mathsf{FEXPTIME(poly)}$ algorithm to find a top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ when Q is in DATALOG. The algorithm is a variation of the algorithm given for $\mathsf{FRP}(\exists\text{FO}^+)$ above. Indeed, it suffices to observe that the Σ_2^p-oracle used in that algorithm can be replaced by an $\mathsf{EXPTIME}$ oracle. As a consequence, the algorithm will return a top-k selection of packages by making polynomially many calls to an $\mathsf{EXPTIME}$ oracle, and is thus in $\mathsf{FEXPTIME(poly)}$.

(2) Data complexity. We show that $\mathsf{FRP}(\mathcal{L}_Q)$ is $\mathsf{FP}^{\mathsf{NP}}$-complete when $\mathcal{L}_Q$ is CQ, UCQ, $\exists\text{FO}^+$, $\text{DATALOG}_{\mathsf{nr}}$, FO or DATALOG.

Lower bound. For the lower bound, it suffices to show that $\mathsf{FRP}(\text{CQ})$ is $\mathsf{FP}^{\mathsf{NP}}$-hard. We verify this by reduction from MAX-WEIGHT SAT, which is known to be $\mathsf{FP}^{\mathsf{NP}}$-complete (cf. [26]). An instance of MAX-WEIGHT SAT consists of a set $\mathcal{C}$ of clauses $\{C_1, \ldots, C_r\}$ such that each clause C_i has an integer weight w_i associated with it. Furthermore, for each $i \in [1, r]$, the clause C_i is of the form $\ell_1^i \vee \ell_2^i \vee \ell_3^i$, where for each $j \in [1, 3]$, ℓ_j^i is either a variable or the negation of a variable in $X = \{x_1, \ldots, x_m\}$. Given $(\mathcal{C}, \{w_1, \ldots, w_r\})$, MAX-WEIGHT SAT is to find the truth assignment of X that satisfies a set of clauses in $\mathcal{C}$ with the most total weight, *i.e.,* it is to find a truth assignment μ_X of X such that $\sum_{\{i \mid C_i(\mu_X) \text{ is true}\}} w_i$ is maximized.

Given a MAX-WEIGHT SAT instance $\mathcal{C} = \{C_1, \ldots, C_r\}$ in which each clause C_i has a weight w_i, we define the same database D, identity queries Q, empty query Q_c, and function $\mathsf{cost}()$ as their counterparts given in the proof of Lemma 4.4. That is, each clause C_i is encoded by tuples in D such that such tuples encode all truth assignments for variables in C_i which make C_i true, and moreover, for each package N such that $\mathsf{cost}(N) \leq 1$, N encodes a valid truth assignment for (part of) variables in X which make some clauses in $\{C_1, \ldots, C_r\}$ true. Furthermore, we set $C = 1$ and $k = 1$. Finally, We define $\mathsf{val}(N)$ as the sum of all weights associated with the cid-values (*i.e.,* clauses) of tuples in N.

We next show that for a package $N \subseteq Q(D)$ for which $\mathsf{cost}(N) \leq C$, $\{N\}$ is a top-1 package selection iff N encodes the truth assignment of X that satisfies a set of clauses with the most total weight. Clearly, a valid package N consists of tuples $t_1, \ldots, t_s$, at most one for each clause in φ, such that each variable in X that occurs in one of the tuples t_i has a unique value (0 or 1) in N. In other words, a package corresponds to a partial truth assignment. Clearly, the top-1 package selection will be $\{N\}$, where N is a valid package N (*i.e.,* partial truth assignment of X) that maximizes $\mathsf{val}()$. By the definition of $\mathsf{val}()$, the package that corresponds to a partial truth assignment with most total weight will be selected as the top-1 package. By completing the partial truth assignment in an arbitrary way, we obtain a truth assignment that maximizes the total weight of all clauses that it satisfies. Conversely, giving a truth assignment μ_X that maximizes the weights, we can easily construct a package N that consists of tuples corresponding to the clauses satisfied by μ_X. Again, $\{N\}$ will be a top-1 package selection.

Upper bound. For the upper bound, it suffices to observe that the algorithm presented for the $\mathsf{FRP}(\exists\text{FO}^+)$ case above works for all the languages considered and moreover, the oracle used in the algorithm reduces to an NP oracle. Indeed, the oracle guesses a package and then verifies whether (i) it is valid, (ii) has a certain rating value, and (iii) is distinct from a number of other packages. When data complexity is concerned, condition (i) is in PTIME for all considered languages. Conditions (ii) and (iii) are always in PTIME. As a consequence, the algorithm makes polynomially many calls to an NP oracle, from which the result follows.

(3) In the absence of compatibility constraints. We next show the complexity results when the compatibility constraints Q_c is absent.

Combined complexity. When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$, FO or DATALOG, we show that the absence of Q_c makes no difference when combined complexity is concerned. In contrast, the absence of Q_c has an impact when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$.

When $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$. We show that $\mathsf{FRP}(\mathcal{L}_Q)$ is $\mathsf{FP}^{\mathsf{NP}}$-hard for CQ and is in $\mathsf{FP}^{\mathsf{NP}}$ for $\exists$FO$^+$.

For the lower bound observe that the proof of $\mathsf{FP}^{\mathsf{NP}}$-hardness of FRP(CQ) (data complexity) given above does not use compatibility constraints Q_c. Thus FRP(CQ) is $\mathsf{FP}^{\mathsf{NP}}$-hard in the absence of Q_c, even when the query Q is fixed. For the upper bound, consider the algorithm for $\mathsf{FRP}(\exists\text{FO}^+)$ given above. It is readily verified that the oracle used in the algorithm reduces to an NP oracle, from which the result follows. Hence $\mathsf{FRP}(\exists\text{FO}^+)$ is in $\mathsf{FP}^{\mathsf{NP}}$.

When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$, FO or DATALOG. We show that the absence of Q_c does not affect the combined complexity. For the lower bounds, it suffices to observe that the reductions of the $\mathsf{FPSPACE(poly)}$ and $\mathsf{FEXPTIME(poly)}$ lower bounds given above for DATALOG$_{\mathsf{nr}}$, FO and DATALOG do no use compatibility constraints (*i.e.,* each Q_c^i is the empty query). Furthermore, the $\mathsf{FPSPACE(poly)}$ and $\mathsf{FEXPTIME(poly)}$ algorithms given in that proof remain valid after the Q_c test is removed. As a consequence, $\mathsf{FRP}(\mathcal{L}_Q)$ remains $\mathsf{FPSPACE(poly)}$-complete when $\mathcal{L}_Q$ is either DATALOG$_{\mathsf{nr}}$ or FO, and $\mathsf{FEXPTIME(poly)}$-complete when $\mathcal{L}_Q$ is DATALOG, even in the absence of compatibility constraints.

Data complexity. For data complexity, observe that the $\mathsf{FP}^{\mathsf{NP}}$-hardness proof of the data complexity of FRP does not use compatibility constraints. Since the data complexity of $\mathsf{FRP}(\mathcal{L}_Q)$ is in $\mathsf{FP}^{\mathsf{NP}}$ for all languages considered even in the presence of Q_c, we obtain the desired complexity bounds.

This completes the proof of Theorem 5.1. $\square$

Deciding the maximum bound. We show that MBP(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 $L = L_1 \cap L_2$ [35], analogous to how DP is defined with NP and coNP [26].

When $\mathcal{L}_Q$ is FO, DATALOG$_{\mathsf{nr}}$ 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 5.2. *For $\mathsf{MBP}(\mathcal{L}_Q)$, the combined complexity is*
- $\mathsf{D}_2^{\mathsf{p}}$-*complete when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$;*
- PSPACE-*complete when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO; and*
- $\mathsf{EXPTIME}$-*complete when $\mathcal{L}_Q$ is DATALOG.*

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

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

Proof. Below we first verify the combined complexity bounds of $\mathsf{MBP}(\mathcal{L}_Q)$ when

$\mathcal{L}_Q$ ranges over various query languages. We then give the data complexity. Finally, we provide the complexity bounds for a special cases, when compatibility constraints Q_c are absent.

(1) Combined complexity. We first verify the combined complexity bounds.

When $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$. It suffices to show that $\mathsf{MBP}(\mathcal{L}_Q)$ is D_2^p-hard for CQ and is in D_2^p for $\exists$FO$^+$.

Lower bound. We show that MBP(CQ) is D_2^p-hard by reduction from $\exists^*\forall^*$3DNF–$\forall^*\exists^*$3CNF, which is D_2^p-complete [35]. An instance of $\exists^*\forall^*$3DNF–$\forall^*\exists^*$3CNF is a pair (φ_1, φ_2) of $\exists^*\forall^*$3DNF instances as described in the proof of Theorem 4.1 for CQ. It is to decide whether φ_1 is true while φ_2 is false. Given (φ_1, φ_2), we define a database D, a query Q in CQ, a query Q_c in CQ for compatibility constraints, functions $\mathsf{cost}()$ and $\mathsf{val}()$ and constants C, k and B such that φ_1 is true and φ_2 is false iff B is the maximum bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$. We show this holds even when $k = 1$.

Consider (φ_1, φ_2), where $\varphi_1 = \exists X_1 \forall Y_1\, \psi_1(X_1, Y_1)$ and $\varphi_2 = \exists X_2 \forall Y_2\, \psi_1(X_2, Y_2)$. We give the reduction as follows.

(1) The database D consists of the four relations I_{01}, $I_\vee$, $I_\wedge$ and $I_\neg$ given in Figure 4.1, specified by $R_{01}(X)$, $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$ and $R_\neg(A, \vec{A})$, as well as a relation $I_c = \{(1,0,0), (1,1,1), (0,0,1), (0,1,1)\}$ specified by schema $R_c = (C_1, C_2, C)$. As will be seen shortly, I_c will be used to inspect the truth values of (ψ_1, ψ_2): for a tuple $t \in I_c$, $t[C] = 0$ if and only if $t[C_1] = 1$ and $t[C_2] = 0$, where $t[C_1]$ and $t[C_2]$ indicate whether ψ_1 and ψ_2 are satisfied by some truth assignments, respectively.

(2) We define a CQ query Q as follows:

$$Q(\vec{x}_1, b_1, \vec{x}_2, b_2) \;=\; \exists \vec{y}_1, \vec{y}_2 \big((Q_{X_1}(\vec{x}_1) \wedge Q_{Y_1}(\vec{y}_1) \wedge Q_{\psi_1}(\vec{x}_1, \vec{y}_1, b_1) \;\wedge$$
$$(Q_{X_2}(\vec{x}_2) \wedge Q_{Y_2}(\vec{y}_2) \;\wedge Q_{\psi_2}(\vec{x}_2, \vec{y}_2, b_2))\big)$$

where $Q_{X_1}(\vec{x}_1)$ generates all truth assignments of X_1 variables in φ_1 by means of Cartesian products of R_{01}; similarly $Q_{Y_1}(\vec{y}_1)$ for Y_1, $Q_{X_2}(\vec{x}_2)$ for X_2 and $Q_{Y_2}(\vec{y}_2)$ for Y_2. Query Q_{ψ_1} encodes the truth value of $\psi_1(X_1, Y_1)$ for given truth assignments μ_{X_1} and μ_{Y_1}, expressed in CQ in terms of $I_\vee$, $I_\wedge$ and $I_\neg$; it returns $b_1 = 1$ if $\psi_1(X_1, Y_1)$ is satisfied by μ_{X_1} and μ_{Y_1}, and $b_1 = 0$ otherwise. Similarly, Q_{ψ_2} encodes the truth value of $\psi_2(X_2, Y_2)$ for given truth assignments μ_{X_2} and μ_{Y_2}. Each tuple t in $Q(D)$ encodes a truth assignment μ_{X_1} (in $t[X_1]$) and a truth assignment μ_{X_2} (in $t[X_2]$), along with (b_1, b_2).

(3) We define a CQ query Q_c as follows:

$$\begin{aligned} Q_c = \exists &\vec{x}_1, \vec{x}_2, \vec{y}_1, \vec{y}_2, b_1, b_2, c_1, c_2, c \\ &\big(R_Q(\vec{x}_1, b_1, \vec{x}_2, b_2) \wedge (Q_{Y_1}(\vec{y}_1) \wedge Q_{\psi_1}(\vec{x}_1, \vec{y}_1, c_1)) \wedge \\ &\qquad (Q_{Y_2}(\vec{y}_2) \wedge Q_{\psi_2}(\vec{x}_2, \vec{y}_2, b_2)) \wedge \\ &\qquad (Q'_{\psi_2}(\vec{x}_2, c_2) \wedge c_2 = 0) \wedge \\ &\qquad R_c(c_1, b_2, c) \;\wedge\; c = 1\big). \end{aligned}$$

Here R_Q is the schema of the result of $Q(D)$, and Q_{Y_1}, Q_{Y_2}, Q_{ψ_1} and Q_{ψ_2} are as given above. The query Q_{ψ_2} only selects those truth values of ψ_2 that are the same as b_2. The query Q'_{ψ_2} tests whether there exists a truth assignment μ_{Y_2} for Y_2 such that μ_{X_2} and μ_{Y_2} do not satisfy ψ_2; it is nonempty iff this is the case. Intuitively, $Q_c(N)$ is nonempty if for a given $N \subseteq Q(D)$ that encodes a truth assignment μ_{X_1} for X_1 and a truth assignment μ_{X_2} for X_2, (a) there exists a truth assignment of Y_1 that makes $\psi_1(X_1, Y_1)$ false (*i.e.,* $b_1 = 0$); or (b) there exists a truth assignment μ_{Y_2} of Y_2 such that the truth value of $\psi_2(X_2, Y_2)$ with μ_{X_2} and μ_{Y_2} is the same as b_2, and moreover, $\psi_2(X_2, Y_2)$ with μ_{X_2} is not always true, *i.e.,* there exists some μ'_{Y_2} such that μ_{X_2} and μ'_{Y_2} make $\psi_2(X_2, Y_2)$ false; this is enforced by $R_c(c_1, b_2, c) \ \wedge \ c = 1$, among other things.

(4) We define, for each package N, $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$ and $\mathsf{cost}(\emptyset) = \infty$, and set $C = 1$, *i.e.,* a valid N consists of one tuple only. Given $N = \{t\}$, we define $\mathsf{val}(N) = 1$ if the (b_1, b_2) value in t is $(1, 0)$, $\mathsf{val}(N) = 2$ if the (b_1, b_2) value in t is $(1, 1)$, $\mathsf{val}(\emptyset) = 0$, and for any other package $N \neq \emptyset$, $\mathsf{val}(N) = 0$. We define bound $B = 1$.

We next verify that φ_1 is true and φ_2 is false iff B is the maximum bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$, where $k = 1$.

$\boxed{\Rightarrow}$ First assume that φ_1 is true and φ_2 is false. Then there exists a truth assignment $\mu^0_{X_1}$ for X_1 such that for all truth assignment μ_{Y_1} for Y_1, ψ_1 is true, and moreover, for all truth assignments μ_{X_2} for X_2, there exists a truth assignment μ_{Y_2} for Y_2 such that ψ_2 is not satisfied by μ_{X_2} and μ_{Y_2}. Let N consist of the tuple representing $\mu^0_{X_1}$ and an arbitrary μ_{X_2}, with $(b_1, b_2) = (1, 0)$. Then when evaluating $Q_c(N, D)$, b_1 is 1 for all μ_{Y_1}, and hence $Q_c(N, D) = \emptyset$ by the definitions of I_c and Q_c. Moreover, $\mathsf{val}(N) \geq B$ and $\mathsf{cost}(N) \leq C$. Hence $\mathcal{N} = \{N\}$ is a valid package selection. As a result, B is a bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$. In addition, there exists no $B' > B$ such that B' is also a bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$. Indeed, by the definition of $\mathsf{val}()$, the only possible N' with $\mathsf{val}(N')$ higher than B consists of some tuple in which the (b_1, b_2) value is $(1, 1)$. However, $Q_c(N', D)$ is nonempty because Q'_ψ is nonempty by the definitions of I_c and Q_c, among other things. Therefore, B is the maximum bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$.

$\boxed{\Leftarrow}$ Conversely, assume that either φ_1 is false or φ_2 is true. Then given any N consisting of a tuple representing a truth assignment μ_{X_1} for X_1 and a truth assignment μ_{X_2} for X_2, we have the following cases to consider. (a) If φ_1 is false, then when evaluating $Q_c(N, D)$, there must exist μ_{Y_1} such that ψ_1 is false, *i.e.,* $b_1 = 0$. By the definitions of I_c and Q_c, $Q_c(N, D)$ is nonempty. That is, B is not even a bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$. (b) If φ_1 and φ_2 are both true, then there exists a truth assignment $\mu^0_{X_1}$ for X_1 such that for all truth assignment μ_{Y_1} for Y_1, ψ_1 is true, and similarly, there exists such $\mu^0_{X_2}$ for X_2. We define a package N consisting of a single tuple t_0 that encodes $\mu^0_{X_1}$ and $\mu^0_{X_2}$ with $t_0(b_1, b_2) = (1, 1)$. Then N is a valid package selection. Indeed, $\mathsf{val}(N) \geq B$ and $\mathsf{cost}(N) \leq C$, and moreover, $Q_c(N, D) = \emptyset$ since Q'_ψ is empty here given $\mu^0_{X_2}$, among other things. Since $\mathsf{val}(N) = 2 > B$, B is not the maximum bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$.

Upper bound. By the definition of maximum bound, the set of yes-instances to $\mathsf{MBP}(\exists\mathrm{FO}^+)$ is $L_1 \cap L_2$, where

- $L_1 = \big\{(Q, D, Q_c, C, \mathsf{cost}(), \mathsf{val}(), B, k) \mid$ there exists a set $\mathcal{N} = \{N_i \mid i \in [1, k]\}$ of distinct packages such that for each $i \in [1, k]$, $N_i \subseteq Q(D)$, $\mathsf{cost}(N_i) \leq C$, $\mathsf{val}(N_i) \geq B$, N_i is of a polynomial size, and $Q_c(N_i, D) = \emptyset\big\}$; and
- $L_2 = \big\{(Q, D, Q_c, C, \mathsf{cost}(), \mathsf{val}(), B, k) \mid$ there exists *no* set $\mathcal{N}' = \{N'_i \mid i \in [1, k]\}$ of distinct packages such that for each $i \in [1, k]$, $N'_i \subseteq Q(D)$, $\mathsf{cost}(N'_i) \leq C$, $\mathsf{val}(N'_i) > B$, N'_i is of a polynomial size, and $Q_c(N'_i, D) = \emptyset\big\}$.

It suffices to show that $L_1 \in \Sigma^p_2$ and $L_2 \in \Pi^p_2$. For if it holds, then the membership in D_2^p is immediate by the definition of D_2^p. We show that L_1 is in Σ_2^p by giving an algorithm as follows:

1. Guess a set $\mathcal{N} = \{N_i \mid i \in [1,k]\}$ of distinct packages of polynomial sizes.
2. Check whether for each $i \in [1,k]$, $N_i \subseteq Q(D)$ (in NP), $\mathsf{cost}(N_i) \leq C$ (in PTIME), $\mathsf{val}(N_i) \geq B$ (in PTIME) and $Q_c(N_i, D) = \emptyset$ (in coNP). If so, return "yes", and otherwise reject the guess and go back to step 1.

Obviously the algorithm is in Σ_2^p, and hence so is L_1. Similarly, one can verify that L_2 is in Π_2^p.

When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO. We next show that for DATALOG$_{\mathsf{nr}}$ and FO, MBP is PSPACE-complete.

Lower bound. We show that $\mathsf{MBP}(\mathcal{L}_Q)$ is PSPACE-hard when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ by reduction from Q3SAT. Given an instance φ of Q3SAT, we define a database D, a query Q in DATALOG$_{\mathsf{nr}}$, empty compatibility constraints Q_c, functions $\mathsf{cost}()$, $\mathsf{val}()$, and constants C and k. These are the same as their counterparts given in the proof of Theorem 4.1 for DATALOG$_{\mathsf{nr}}$. We also set $B = 1$. We show that φ is true iff $B = 1$ is the maximum bound for $(Q, D, Q_c = \emptyset, \mathsf{cost}(), \mathsf{val}(), C, k)$, when $k = 1$.

We show that this is indeed a reduction. To see this, first assume that φ is true. Then $(\ ) \in Q(D)$. As a result, $B = 1$ is a maximum bound for $(Q, D, Q_c = \emptyset, \mathsf{cost}(), \mathsf{val}(), 1, 1)$ since $\mathsf{val}()$ assigns 1 to all packages. Conversely, if φ is false, then $Q(D) = \emptyset$ Consequently, $B = 1$ is not the maximum bound for $(Q', D, Q_c = \emptyset, \mathsf{cost}(), \mathsf{val}(), 1, 1)$ since $\mathsf{cost}(\emptyset) > C$.

When $\mathcal{L}_Q$ is FO, we show that $\mathsf{MBP}(\mathcal{L}_Q)$ is PSPACE-hard by reduction from the membership problem for FO as described in the proof of Theorem 4.1 for FO. Given an instance (Q, D, t) of the membership problem for FO, we define an FO query Q', a database D, Q_c as empty query, functions $\mathsf{cost}()$, $\mathsf{val}()$, and constants $C = 1$ and $k = 1$. These are the same as their counterparts given in the proof of Theorem 4.1 for FO. In addition, we set $B = 1$. It is readily verified that $t \in Q(D)$ iff $B = 1$ is the maximum bound for $(Q', D, Q_c = \emptyset, \mathsf{cost}(), \mathsf{val}(), 1, 1)$.

Upper bound. We show that MBP is in PSPACE for DATALOG$_{\mathsf{nr}}$ and FO. Indeed, consider the algorithm for L_1 given earlier. It is in NPSPACE for DATALOG$_{\mathsf{nr}}$ and FO. Similarly, the algorithm for L_2 is also in NPSPACE. Hence the algorithm is in NPSPACE = PSPACE, and so is MBP for FO and DATALOG$_{\mathsf{nr}}$.

When $\mathcal{L}_Q$ is DATALOG. We show that MBP(DATALOG) is EXPTIME-complete.

Lower bound. We show that MBP(DATALOG) is EXPTIME-hard by reduction from the membership problem for DATALOG as in the proof of Theorem 4.1 for DATALOG. Given an instance (Q, D, t) of the membership problem for DATALOG, we use the same DATALOG query Q', database D, empty compatibility constraint Q_c, functions $\mathsf{cost}()$ and $\mathsf{val}()$, and constants $C = 1$ and $k = 1$, as given in in the proof of Theorem 4.1 for DATALOG. We set $B = 1$. Then one can readily verify that $t \in Q(D)$ iff $B = 1$ is the maximum bound for $(Q', D, Q_c = \emptyset, \mathsf{cost}(), \mathsf{val}(), 1, 1)$.

Upper bound. We given an EXPTIME algorithm to check whether B is the maximum bound, as follows.

1. Compute $Q(D)$, in EXPTIME.
2. Enumerate all subsets of $Q(D)$ consisting of polynomially many tuples.
3. For each $\mathcal{N}$ consisting of k such pairwise distinct subsets, and for each set N_i in $\mathcal{N}$, check: (a) whether $Q_c(N_i, D) = \emptyset$, in EXPTIME, and (b) $\mathsf{cost}(N_i) \leq C$; and (c) whether $\mathsf{val}(N_i) \geq B$ in PTIME. If all these conditions are satisfied,

continue; otherwise returns "no".

4. For each $\mathcal{N}'$ of k such pairwise distinct subsets, check (a) and (b) above and (c'): whether $\mathsf{val}(N') > B$ for all $N' \in \mathcal{N}'$. If so, returns "no", otherwise continue to check the next $\mathcal{N}'$.
5. Return "yes" after all such sets are inspected.

Each of steps 1–4 takes EXPTIME. Hence MBP is in EXPTIME for DATALOG.

(2) Data complexity We next show the data complexity of $\mathsf{MBP}(\mathcal{L}_Q)$. It suffices to show that $\mathsf{MBP}(\mathrm{CQ})$ is DP-hard when Q is a fixed CQ query and Q_c is absent, and $\mathsf{MBP}(\mathcal{L}_Q)$ is in DP for fixed DATALOG and FO queries Q and Q_c.

Lower bound. We show that $\mathsf{MBP}(\mathrm{CQ})$ is DP-hard when Q is fixed and Q_c is absent, by reduction from SAT-UNSAT (see the proof of Theorem 4.5 (CQ case) for the statement of SAT-UNSAT). Given an instance (φ_1, φ_2) defined over variables X, Y, respectively, we define D, Q, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C, B and k. We show that φ_1 is satisfiable and φ_2 is not satisfiable iff B is the maximum bound for $(Q, D, Q_c, C, \mathsf{cost}(), \mathsf{val}(), C, k)$.

(1) Database D. The database is defined over a single relation $R_C(\mathsf{cid}, L_1, V_1, L_2, V_2, L_3, V_3)$. Its corresponding instance I_C consists of the following set of tuples. For each $i \in [1, r]$, let $C_i = \ell_1^i \vee \ell_2^i \vee \ell_3^i$ be the ith clause of φ_1. For any possible truth assignment μ_i of variables in the literals in C_i that make C_i true, we add a tuple $(i, x_k, v_k, x_l, v_l, x_m, v_m)$, where $x_k = \ell_1^i$ in case $\ell_1^i \in X$ and $x_k = \vec{\ell_1^i}$ in case $\ell_1^i = \vec{x}_k$. We set $v_k = \mu_i(x_k)$; similarly for x_l, x_m and v_l and v_m. Similarly, we add tuples for the clauses in φ_2 but using cid values ranging from $r+1$ to $r+s$, where s denotes the number of clauses in φ_2.

(2) We take Q as the identity query and define Q_c to be the empty query.

(3) We define $\mathsf{val}(N) = 1$ when N contains tuples that carry only variables in X; $\mathsf{val}(N) = 2$ if N contains tuples that carry variables in X *and* tuples that carry variables in Y; and $\mathsf{val}(N) = 0$ otherwise. We set $B = 1$ and $k = 1$.

(4) We define $\mathsf{cost}(N) = 1$ in case that (a) N contains precisely one tuple for each clause in φ_1; (b) if N additionally contains a tuple denoting a clause in φ_2, then N should also contain precisely one tuple for each clause in φ_2; and (c) all tuples in N should agree on the values of variables in X and Y. For any other N, we define $\mathsf{cost}(N) = 2$. We set $C = 1$.

We next show that φ_1 is satisfiable and φ_2 is not satisfiable iff B is the maximum bound for $(Q, D, Q_c, C, \mathsf{cost}(), \mathsf{val}(), C, k)$.

$\boxed{\Rightarrow}$ First assume that φ_1 is satisfiable and φ_2 is not satisfiable. Let μ_X^0 be a truth assignment that makes φ_1 true. Let N consist of r tuples, one for each clause in φ_1, such that the variables in these clauses take values as given by μ_X^0. Clearly, $N \subseteq Q(D)$. Furthermore, $\mathsf{cost}(N) = 1$ and $\mathsf{val}(N) = 1$, by the definition of $\mathsf{cost}()$ and $\mathsf{val}()$. Moreover, there exists no $B' > B$ such that there exists N' with $\mathsf{cost}(N') \leq C$ and $\mathsf{val}(N') \geq B'$. Indeed, if this happens then by the definition of $\mathsf{val}()$, N' must carry both variables in X and Y, and since $\mathsf{cost}(N') \leq 1$, N' must encode a truth assignment μ_Y of Y that satisfies φ_2. But this is impossible since φ_2 is not satisfiable.

$\boxed{\Leftarrow}$ Conversely, assume that φ_1 is not satisfiable or φ_2 is satisfiable. Consider the following cases. (1) If φ_1 is not satisfiable, then by the definition of $\mathsf{cost}()$, no N can exist that carries variables in X. That is, B is not even a bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$. (2) If φ_1 is satisfiable and φ_2 is satisfiable, we let N' be the package that encodes truth assignments μ_X^0 and μ_Y^0 that make φ_1 and φ_2

true, respectively. In other words, N' consists of $r+s$ tuples corresponding to the clauses in φ_1 and φ_2 that conform to the given truth assignments. Clearly, $\mathsf{cost}(N') = 1 \leq C$ and $\mathsf{val}(N') = 2 > B$. Hence B is not the maximum bound for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, k)$.

Upper bound. We show that MBP is in DP when Q and Q_c are fixed. Consider L_1 and L_2 defined earlier for $\exists$FO$^+$. When Q and Q_c are fixed, L_1 is in NP and L_2 is in coNP, for $\mathcal{L}_Q$ ranging over all the languages considered. As the set of yes-instances of MBP is $L_1 \cap L_2$, MBP is in DP.

(3) Special case: In the absence of compatibility constraints. We reinvestigate MBP($\mathcal{L}_Q$) when Q_c is absent.

Lower bound. We first consider combined complexity. For CQ, it is already shown in the proof of data complexity above that MBP is DP-complete for a fixed CQ query Q. Furthermore, the reduction in that proof uses the empty compatibility constraint. Thus MBP(CQ) is also DP-hard in the absence of Q_c.

For FO, DATALOG$_{\mathsf{nr}}$ and DATALOG, the proofs of the lower bounds given above do not use any compatibility constraints. Hence their combined complexity bounds remain unchanged.

For the data complexity, we have shown that MBP(CQ) is DP-hard when Q_c is absent and Q is fixed. Hence the data complexity of MBP remains DP-hard here for all the languages considered.

Upper bound. For FO, DATALOG$_{\mathsf{nr}}$ and DATALOG, the upper bound proofs given earlier remain intact when Q_c is absent. For $\exists$FO$^+$, we use the following algorithm to check membership in the language L_1 defined earlier:

1. Guess k sets, where each set consists of polynomially many CQ queries from Q, and for each CQ query in each set, guess a tableau from D. These tableaux yield a package $\mathcal{N} = \{N_i \mid i \in [1,k]\}$, where $N_i \subseteq Q(D)$ for all $i \in [1,k]$.
2. Check whether $\mathsf{cost}(N_i) \leq C$, $\mathsf{val}(N_i) \geq B$, and $N_i \neq N_j$ when $i \neq j$. If so, return "yes"; otherwise reject the guess and go back to step 1.

This is in NP since step 2 is in PTIME. Similarly, one can show that membership in L_2 can be decided in coNP. Hence MBP($\exists$FO$^+$) $= L_1 \cap L_2$ is in DP, the same as MBI($\exists$FO$^+$). This algorithm also works when Q is fixed. Hence the data complexity here is also in DP.

This completes the proof of Theorem 5.2. □

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

THEOREM 5.3. *For* CPP($\mathcal{L}_Q$)*, the combined complexity is*

- $\#\cdot$coNP*-complete when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$;*
- $\#\cdot$PSPACE*-complete when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO;*
- $\#\cdot$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, but it is $\#\cdot$NP*-complete for CQ, UCQ and $\exists$FO$^+$.*

Its data complexity is $\#\cdot$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 [16]. For a complexity class C of decision problems, $\#\cdot$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 [32]. 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. We first establish the combined complexity results and then consider data complexity. Finally, we consider the special case in the absence of compatibility constraints.

(1) Combined complexity. We first verify the combined complexity bounds.

When $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$. It suffices to show that $\mathsf{CPP}(\mathrm{CQ})$ is $\#\cdot\mathsf{coNP}$-hard and $\mathsf{CPP}(\exists\mathrm{FO}^+)$ is in $\#\cdot\mathsf{coNP}$.

Lower bound. We show that $\mathsf{CPP}(\mathrm{CQ})$ is $\#\cdot\mathsf{coNP}$-hard by a parsimonious reduction from $\#\Pi_1\mathrm{SAT}$, which is known to be $\#\cdot\mathsf{coNP}$-complete [12]. An instance of $\#\Pi_1\mathrm{SAT}$ consists of a universally quantified Boolean formula of the form $\varphi(X, Y) = \forall X\,(C_1 \vee \cdots \vee C_r)$, where the C_i's are conjunctions of variables or negated variables taken from $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$; $\#\Pi_1\mathrm{SAT}$ is to count the number of truth assignments of Y that make φ true.

Given an instance φ of $\#\Pi_1\mathrm{SAT}$ we define a database D, Q and Q_c in CQ, $\mathsf{cost}()$ and $\mathsf{val}()$, C and B such that the number of valid packages for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$ is equal to the number of truth assignments of Y that make φ true.

(1) The database D consists of three relations specified by schemas $R_{01}(X)$, $R_\vee(B, A_1, A_2)$ and $R_\neg(A, \vec{A})$ given in the proof of Theorem 4.1. Their corresponding instances are shown in Figure 4.1. More specifically, I_{01} encodes the Boolean domain, and $I_\vee$ and $I_\neg$ encode disjunction and negation, respectively.

(2) The query Q simply returns truth assignment for Y, that is,

$$Q(\vec{y}\,) = R_{01}(y_1) \wedge \cdots \wedge R_{01}(y_n),$$

where $\vec{y} = (y_1, \ldots, y_n)$.

(3) We consider the following CQ query Q_c:

$$Q_c(\vec{y}\,) = R_Q(\vec{y}\,) \wedge \exists \vec{x}\big(\bigwedge_{i\in[1,m]} R_{01}(x_i) \wedge \bigwedge_{i\in[1,r]} Q_{\vec{C}_i}(\vec{x}, \vec{y}\,)\big),$$

where $\vec{x} = (x_1, \ldots, x_m)$, $\vec{y} = (y_1, \ldots, y_m)$, and $Q_{\vec{C}_i}$ leverages R_{01}, $R_\vee$ and $R_\neg$ to encode the disjunctions in the negated clause $\vec{C}_i$ of C_i. The semantics of $Q_{\vec{C}_i}$ is that for a given truth assignment μ_X of X and μ_Y for Y, $Q_{\vec{C}_i}(\mu_X, \mu_Y)$ evaluates to true if $\vec{C}_i$ holds for μ_X and μ_Y; and $Q_{\vec{C}_i}(\mu_X, \mu_Y)$ returns false otherwise.

(4) We define $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$, $\mathsf{cost}(\emptyset) = \infty$, and set $C = 1$. That is, packages consist of a single tuple. Furthermore, $\mathsf{val}(N) = b$ for some constant b for all packages N. We set $B = b$.

To see that this is a reduction, observe that $N = \{s\} \subseteq Q(D)$ iff s represents a truth assignment for Y in φ, as returned by Q, and in addition, $Q_c(N, D) = \emptyset$. That is, there does not exist a truth assignment μ_X of X which makes $\vec{C}_1 \wedge \cdots \wedge \vec{C}_r$ false. In other words, all truth assignments μ_X of X make at least one of the clause C_i true, and hence make φ true. Furthermore, since the condition $\mathsf{val}(N) \geq B$ does not remove any packages we have that the number of valid packages for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$ is equal to the number of truth assignments of Y that make φ true.

Upper bound. Consider D, Q, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C and B as input. Given a package N, it is readily verified that (i) checking whether $N \subseteq Q(D)$ is in NP; (ii) testing $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) \geq B$ is in PTIME; and (iii) checking $Q_c(N, D) = \emptyset$ is in coNP. In other words, there exists two Turing machines: an NP machine $\mathcal{M}_1$ and coNP machine $\mathcal{M}_2$, such that $(\vec{x}, \vec{y})$ is accepted by both $\mathcal{M}_1$ and $\mathcal{M}_2$, where $\vec{x}$ is an encoding of D, Q, Q_c, C, B, $\mathsf{cost}()$ and $\mathsf{val}()$, and $\vec{y}$ is an encoding of a tuple s satisfying the conditions above. That is, the witness function is in DP. Furthermore, since no new values are invented by queries, the encoding $\vec{y}$ of tuples s in $Q(D)$ is bounded by $\mathsf{arity}(R_Q) \times \log|\mathsf{adom}(Q, D)|$, where $\mathsf{arity}(R_Q)$ denotes the arity of the output schema of Q, and $\mathsf{adom}(Q, D)$ is the set of constants appearing in D or Q. Since packages are of size polynomial in $|D|$, we may conclude that $|\vec{y}|$ is bounded by a polynomial in $|\vec{x}|$. Putting these together, we have that $\mathsf{CPP}(\exists\mathrm{FO}^+)$ is in #·DP. Note however, that $\#\cdot\mathsf{DP} \subseteq \# \cdot \mathsf{P}^{\mathsf{NP}}$ simply because the # operator is monotonic in its argument. The #·coNP upper bound then follows from $\# \cdot \mathsf{P}^{\mathsf{NP}} = \#\cdot\mathsf{coNP}$ [16].

When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO. We next show that $\mathsf{CPP}(\mathcal{L}_Q)$ is #·PSPACE-complete when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO.

Lower bound. We show that $\mathsf{CPP}(\mathrm{DATALOG}_{\mathsf{nr}})$ is #· PSPACE-hard by a parsimonious reduction from #QBF, which is #·PSPACE-complete (implicit in [22]). An instance of #QBF consists of a Boolean formula of the form $\varphi = \exists X\, \forall y_1 P_2 y_2 \cdots P_n y_n\, \psi$, where $P_i \in \{\exists, \forall\}$, for $i \in [2, n]$, and ψ is quantifier-free Boolean formula over the variables in $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$; #QBF is to count the number of truth assignments of X that make φ true.

Given an instance φ of #QBF we construct a database D, query Q, empty compatibility constraint Q_c, functions $\mathsf{cost}()$, $\mathsf{val}()$, cost budget C and a constant B. We show that $|\{N \mid N \subseteq Q(D) \text{ is a valid package with } \mathsf{val}(N) \geq B\}|$ is equal to the number of truth assignments of X that make φ true.

The reduction from #QBF is similar to the reductions given in the proof of Theorem 4.1 for DATALOG$_{\mathsf{nr}}$, except that the query carries m additional free variables for variables in X. Recall that in that proof, $\mathsf{cost}(N) = |N|$ in case $N \neq \emptyset$, $\mathsf{cost}(\emptyset) = \infty$, and $C = 1$. That is, valid packages consist of a single tuple only. It is readily verified that $N = \{s\} \subseteq Q(D)$ is a valid package iff s corresponds to a truth assignment of X which makes φ true. By letting $\mathsf{val}()$ assign the same value to all tuples, say b, and setting $B = b$, we thus obtain that $|\{N = \{s\} \mid s \in Q(D) \wedge \mathsf{val}(s) \geq B\}|$ is equal to the number of truth assignments of X that make φ true.

The lower bound for $\mathsf{CPP}(\mathrm{FO})$ is verified in the same way, but by providing a reduction from QBF by means of FO queries.

Upper bound. As implied by Theorem 5.2, the witness function for $\mathsf{CPP}(\mathcal{L}_Q)$ is in PSPACE when $\mathcal{L}_Q$ is FO or DATALOG. Hence it suffices to observe that the size of encodings of recommended packages is polynomially bounded by the size of an encoding of the input. Indeed, this readily follows from the fact that the encoding of single tuples is bounded and packages consisting of a number of tuples that is polynomial in the size of the input. Putting these together, we have that $\mathsf{CPP}(\mathcal{L}_Q)$ is in #·PSPACE for FO and DATALOG.

When $\mathcal{L}_Q$ is DATALOG. We show that $\mathsf{CPP}(\text{DATALOG})$ is $\#\cdot\mathsf{EXPTIME}$-complete.

Lower bound. We show that $\mathsf{CPP}(\text{DATALOG})$ is $\#\cdot\mathsf{EXPTIME}$ by showing that for any function for which there exists an alternating polynomial space-bounded Turing machine $\mathcal{M} = (\text{St}, \Sigma, \delta, \imath, s_0)$ such that $h(\vec{x}) = |\{\vec{y} \mid (\vec{x}, \vec{y}) \text{ is accepted by } \mathcal{M}\}|$ and $|\vec{y}| \leq |\vec{x}|^k$ for some k, there exist Q, D, empty compatibility constraint Q_c, functions $\mathsf{cost}()$, $\mathsf{val}()$, and constants C and B such that $|\{N \mid N \subseteq Q(D)$is a valid package with $\mathsf{val}(N) \geq B\}|$ is equal to $h(\vec{x})$. In particular, we set $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$, $\mathsf{cost}(\emptyset) = \infty$, and $C = 1$. In addition, $\mathsf{val}()$ assigns the same value b to all packages. We let $B = b$. Note that packages consist of a single tuple only and since Q_c is the empty query, a tuples is valid iff it belongs to $Q(D)$.

Recall that $\mathsf{EXPTIME}$ coincides with languages accepted by alternating polynomial space-bounded Turing machines. An alternating Turing machine (ATM) is of the form $\mathcal{M} = (\text{St}, \Sigma, \delta, \imath, s_0)$, where St is a set of states with initial state s_0; Σ is a finite tape alphabet; transition function $\delta : \text{St} \times \Sigma \to 2^{\text{St} \times \Sigma \times \{L,R\}}$ and $\imath : \text{St} \to \{\wedge, \vee, \mathsf{acc}, \mathsf{rej}\}$. Here L shifts the head to the left and R shifts it to the right. If $\mathcal{M}$ is in a configuration with state s and $\imath(s) = \mathsf{acc}$, then that configuration is accepting; if it is in a state s with $\imath(s) = \mathsf{rej}$ then the configuration is rejecting. A configuration with s such that $\imath(s) = \wedge$ is accepting if all configurations reachable in one step are accepting; and it is rejecting otherwise. A configuration with s such that $\imath(s) = \vee$ is accepting if one of the configurations reachable in one step is accepting; and it is rejecting otherwise. An ATM $\mathcal{M}$ accepts a string $\vec{x}$ if the initial configuration with state s_0 and head positioned to the left of the input string $\vec{x}$ is accepting.

Let h be a function for which there exists an alternating polynomial space-bounded Turing machine $\mathcal{M} = (\text{St}, \Sigma, \delta, \imath, s_0)$ such that $h(\vec{x}) = |\{\vec{y} \mid (\vec{x}, \vec{y}) \text{ is accepted by } \mathcal{M}\}|$ and $|\vec{y}| \leq |\vec{x}|^k$ for some k. Let p be a polynomial such that $\mathcal{M}$ on an input of length n uses at most $g(n) = n^k + p(n)$ cells of its tape for input $\vec{x}$ and computation; the first n^k cells are reserved for $\vec{y}$ (possible padded with blanks $\sqcup$ to fill all n^k cells). Let $\vec{x} = (x_1, \ldots, x_n)$ be an input string.

Given $\mathcal{M}$ and $\vec{x}$, we define a database D, a DATALOG query Q, and functions and constants as specified above. The database D consists of unary relation R_Σ that encodes the alphabet Σ. The query Q is defined as follows:

- If $\imath(s) = \vee$ then for each $s' \in \text{St}$ for each $a, a' \in \Sigma$ with $\delta(s, a) = (s', a', \mu)$, for some $\mu \in \{L, R\}$, we let $\ell = -1$ if $\mu = L$, and $\ell = 1$ otherwise. For each $i \in [0, g(|\vec{x}|)]$, we add the following rule:

$$\begin{aligned} &\leftarrow \Pi_{s',i+\ell}(z_1, \ldots, z_{i-1}, w_{a'}, z_{i+1}, \ldots, z_{g(|\vec{x}|)}). \\ \Pi_{s,i}(z_1, \ldots, z_{n^k}, z_{n^k+1}, \ldots, z_{i-1}, w_a, z_{i+1}, \ldots, z_{g(|\vec{x}|)}) \end{aligned}$$

- If $\imath(s) = \wedge$, then for each $a \in \Sigma$, we construct the set $Q_{s,a} = \{(s_1, a_1), \ldots, (s_{k(a)}, a_{k(a)})\}$ consisting of all pairs (s_j, a_j) such that $\delta(s, a) = (s_j, a_j, \mu_j)$ for some $\mu_j \in \{L, R\}$. As before, we set $\ell_j = -1$ if $\mu_j = L$ and $\ell_j = 1$ otherwise. For each $i \in [0, g(|\vec{x}|)]$, we add the following rule:

$$\leftarrow \bigwedge_{j=1}^{k(a)} \Pi_{s_j,i+\ell_j}(z_1, \ldots, z_{i-1}, w_{a_j}, z_{i+1}, \ldots, z_{g(|\vec{x}|)}).$$
$$\Pi_{s,i}(z_1, \ldots, z_{i-1}, w_a, z_{i+1}, \ldots, z_{g(|\vec{x}|)})$$

- If $\imath(s) = \mathsf{acc}$ is an accepting state then for each $i \in [0, g(|\vec{x}|)]$, we add the following rule:

$$\Pi_{s,i}(z_1, \ldots, z_{g(|\vec{x}|)}) \leftarrow \bigwedge_{j=1}^{g(|\vec{x}|)} R_\Sigma(z_i),$$

where R_Σ denotes the unary instance consisting of all alphabet symbols.

- If $\imath(s) = \mathsf{rej}$ is a rejecting state then for each $i \in [0, g(|\vec{x}|)]$, we add the following rule:

$$\Pi_{s,i}(z_1, \ldots, z_{g(|\vec{x}|)}) \leftarrow \emptyset.$$

- If $s = s_0$ then we add

$$Q(y_1, \ldots, y_{n^k}) \leftarrow \Pi_{s_0,0}(y_1, \ldots, y_{n^k}, w_{x_1}, \ldots, w_{x_n}, w_\sqcup, \ldots, w_\sqcup).$$

Clearly, $\vec{y} \in Q(D)$ iff $(\vec{x}, \vec{y})$ is accepted by $\mathcal{M}$. Here, $\vec{y} = (y_1, \ldots, y_{n^k})$ is the tuple encoding $\vec{y}$. Furthermore, $|Q(D)| = |\{\vec{y} \mid (\vec{x}, \vec{y}) \text{ is accepted by } \mathcal{M}\}| = h(\vec{x})$.

Upper bound. As shown by Theorem 5.2, the witness function for CPP(DATALOG) is in $\mathsf{EXPTIME}$. Hence it suffices to observe that the size of encodings of recommended packages is polynomially bounded by the size of encoding of the input. Indeed, this readily follows from the fact that the encoding of single tuples is bounded, and packages consist of a number of tuples that is polynomial in the size of the input. Therefore, CPP(DATALOG) is in $\#\cdot\mathsf{EXPTIME}$.

(2) Data complexity. We next verify the data complexity bounds.

We show that $\mathsf{CPP}(\mathcal{L}_Q)$ is $\#\cdot\mathsf{P}$-complete for all the query languages considered. It suffices to show that CPP(CQ) is $\#\cdot\mathsf{P}$-hard and CPP(FO) and CPP(DATALOG) are in $\#\cdot\mathsf{P}$.

Lower bound. We show $\#\cdot\mathsf{P}$-hardness by a parsimonious reduction from #SAT, which is known to be $\#\cdot\mathsf{P}$-complete (recall that $\#\mathsf{P} = \#\cdot\mathsf{P}$). An instance of #SAT is an instance $\varphi(X) = C_1 \wedge \cdots \wedge C_r$ of 3CNF over $X = \{x_1, \ldots, x_m\}$. It is to count the number of truth assignments of X that make φ true.

Given φ, we define a database D, an identity query Q, empty compatibility constraints Q_c, functions $\mathsf{cost}()$, $\mathsf{val}()$ and constant $C = 1$. These are the same as their counterparts given in the proof of Lemma 4.4. Furthermore, we set $B = r$. From that proof, we know that for a package $N \subseteq Q(D)$, $\mathsf{cost}(N) \leq C$ and $\mathsf{val}(N) \geq B$ iff N encodes a truth assignment for X variables which make φ true. Hence, $|\{N \mid N \text{ is a valid package for } (Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)\}|$ is equal to the number of truth assignments of X that satisfy φ.

Upper bound. Given D, Q, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C and B and a package N, verifying whether N is a valid package is in PTIME. Furthermore, since N is polynomially bounded by $|D|$ and N consists of values from the active domains of D and Q, the size of encoding of packages is polynomially bounded by the size of encoding of the input. In other words, $\mathsf{CPP}(\mathcal{L}_Q)$ is in $\#\cdot\mathsf{P}$ for all languages considered.

(3) In the absence of compatibility constraints. We next verify the complexity of CPP in the absence of compatibility constraints.

Combined complexity. When $\mathcal{L}_Q$ is $\text{DATALOG}_{\mathsf{nr}}$, FO or DATALOG, we show that the absence of Q_c makes no difference when combined complexity is concerned. In contrast, the absence of Q_c does have an effect when $\mathcal{L}_Q$ is CQ, UCQ or $\exists\text{FO}^+$.

When $\mathcal{L}_Q$ is CQ, UCQ or $\exists\mathbf{FO}^+$. It suffices to show that CPP(CQ) is $\#\cdot\mathsf{NP}$-hard and $\mathsf{CPP}(\exists\text{FO}^+)$ is in $\#\cdot\mathsf{NP}$.

Lower bound. We show that $\mathsf{CPP}(\mathrm{CQ})$ is $\#\cdot\mathsf{NP}$-hard by a parsimonious reduction from $\#\Sigma_1\mathrm{SAT}$, which is known to be $\#\cdot\mathsf{NP}$-complete [12]. An instance of $\#\Sigma_1\mathrm{SAT}$ consists of an existentially quantified Boolean formula of the form $\varphi(X,Y) = \exists X\ (C_1 \wedge \cdots \wedge C_r)$, where C_i are disjunctions of variables or negated variables taken from $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$; $\#\Sigma_1\mathrm{SAT}$ is to count the number of truth assignments of Y that make φ true.

Given an instance φ of $\#\Sigma_1\mathrm{SAT}$, we define a database D, a CQ query Q, empty compatibility constraints Q_c, functions $\mathsf{cost}()$, $\mathsf{val}()$, and constants C and B. We show that the number of valid packages for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$ is equal to the number of truth assignments of Y that make φ true. In particular, we let $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$, $\mathsf{cost}(\emptyset) = \infty$ and we set $C = 1$. In addition, $\mathsf{val}()$ is a constant function assigning a value b to all packages. We let $B = b$. Note that valid packages consist of a single tuple only.

(1) The database consists of four relations specified by schemas $R_{01}(X)$, $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$, and $R_\neg(A, \vec{A})$ given in the proof of Theorem 4.1. The corresponding instances are shown in Figure 4.1.

(2) The query Q is then given by:

$$Q(\vec{y}) = \exists \vec{x}\Big(\bigwedge_{i\in[1,n]} R_{01}(y_i) \wedge \bigwedge_{i\in[1,m]} R_{01}(x_i) \wedge \bigwedge_{i\in[1,r]} Q_i(\vec{x}, \vec{y})\Big),$$

where $\vec{x} = (x_1, \ldots, x_m)$, $\vec{y} = (y_1, \ldots, y_n)$, and Q_i leverages R_{01}, $R_\vee$ and $R_\neg$ to encode the disjunctions in the clause C_i. The semantics of Q_i is that for a given truth assignment μ_X of X and μ_Y for Y, $Q_i(\mu_X, \mu_Y)$ evaluates to true if C_i holds for μ_X and μ_Y; and $Q_i(\mu_X, \mu_Y)$ returns false otherwise.

To see that this is a reduction, observe that a package N can be recommended iff N consists of a single tuple $s \in Q(D)$. Note that $s \in Q(D)$ iff s represents a truth assignment for Y in φ that makes φ true. Furthermore, since the condition $\mathsf{val}(\{s\}) \geq B$ does not remove any tuples we have that $|\{N \mid N \text{ is a valid package for } (Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)\ \}|$ equal to the number of truth assignments of Y that make φ true.

Upper bound. It is readily verified that $\mathsf{CPP}(\exists\mathrm{FO}^+)$ is in $\#\cdot\mathsf{NP}$, simply because verifying whether a given package is valid is in NP in the absence of compatibility constraints.

When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$, FO or DATALOG. We show that the absence of Q_c does not affect the combined complexity. For the lower bounds, it suffices to observe that the proofs of the $\#\cdot\mathsf{PSPACE}$ and $\#\cdot\mathsf{EXPTIME}$ lower bounds of $\mathsf{CPP}(\mathcal{L}_Q)$ for DATALOG$_{\mathsf{nr}}$, FO and DATALOG given earlier do no use compatibility constraints. Together with the upper bounds given there, we conclude that $\mathsf{CPP}(\mathcal{L}_Q)$ is $\#\cdot\mathsf{PSPACE}$-complete when $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO, and $\mathsf{CPP}(\mathcal{L}_Q)$ is $\#\cdot\mathsf{EXPTIME}$-complete when $\mathcal{L}_Q$ is DATALOG.

Data complexity. For data complexity, we observe that in the $\#\cdot\mathsf{P}$-hardness proof given above, no compatibility constraints were used. Since the data complexity of CPP is in $\#\cdot\mathsf{P}$ even in the presence of compatibility constraints, we can conclude that the data complexity of CPP is $\#\cdot\mathsf{P}$-complete for all the languages considered.

This completes the proof of Theorem 5.3. □

6. 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 6.1. *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 4.1, 5.1, 5.2 and 5.3, 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. **(1) Combined complexity**. The lower bounds of RPP, FRP, MBP and CPP hold here, since their proofs given in Theorems 4.1, 5.1, 5.2 and 5.3, respectively, use only top-1 package with one item. For the upper bounds, the algorithms given there obviously remain intact in the special case for packages with a constant bound B_p.

(2) Data complexity. It suffices to show that, for fixed DATALOG and FO queries Q and Q_c, and packages with a constant bound B_p, RPP, FRP, MBP and CPP are in PTIME, FP, PTIME and FP, respectively.

(a) RPP($\mathcal{L}_Q$). Consider the algorithm given in the proof of Theorem 4.1 for RPP (DATALOG). We revise the algorithm such that in step 3, it only enumerates all subsets of $Q(D)$ consisting of B_q tuples at most, and do step 3(a) and 3(b) of the algorithm for each of these subsets. Obviously, the revised algorithm works here. We next show that it is a PTIME algorithm. Obviously, step 1 and 2 is in PTIME for fixed queries Q and Q_c. Furthermore, there are polynomial many subsets of $Q(D)$ consisting of B_q tuples. So step 3 is also in PTIME. Thus the algorithm is in PTIME. Hence the problem is in PTIME.

(b) FRP($\mathcal{L}_Q$). Observe that the algorithm given in the proof of Theorem 5.1 for FRP ($\exists$FO$^+$) works here, and moreover, it is easy to see that the oracle used in the algorithm reduces to an PTIME oracle. Indeed, since considering packages only with constant bound B_p here, the oracle only need to make polynomial many guesses of subsets of $Q(D)$. Moreover, when data complexity is considered, for each package N

guessed, it is PTIME to check whether (i) it is valid, (ii) has a certain rating value, and (iii) is distinct from a number of other packages. Thus the algorithm is in FP. Hence the problem is in FP.

(c) MBP($\mathcal{L}_Q$). To verify that MBP (FO) and MBP (DATALOG) are both in PTIME, we use the algorithm given in the proof of Theorem 5.2 for MBP (DATALOG) (combined complexity), except that in step 2, the algorithm enumerates all subsets of $Q(D)$ consisting of B_q tuples at most. Then there are only polynomial many such subsets. Obviously, the algorithm can carry over here, and moreover, it is easy to see that the algorithm is in PTIME for data complexity.

(d) CPP($\mathcal{L}_Q$) We give an FP algorithm, given D, Q, Q_c, cost(), val(), C, B, to count the number of packages that are valid for (Q, D, Q_c, cost(), val(), C, B). It works as follows:

1. Denote by n the number of packages that are valid for (Q, D, Q_c, cost(), val(), C, B). Initially, let $n = 0$.
2. Compute $Q(D)$.
3. Enumerates all subsets of $Q(D)$ consisting of B_q tuples at most.
4. For each such subset N, check if N is valid for (Q, D, Q_c, cost(), val(), C, B). If so, let $n = n + 1$; otherwise, continue.
5. Return n after all the subsets of $Q(D)$ are inspected.

Step 1 is in PTIME for fixed Q. Furthermore, there are polynomial many subsets enumerated in step 2, and moreover, it readily verify that step 4 is also in PTIME for fixed Q and Q_c. Thus the algorithm is in FP.

This completes the proof of Corollary 6.1 $\Box$

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 6.2. *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. We first show the complexity results of RPP, FRP, MBP and CPP, respective, for packages of variable sizes, and then for packages with a constant bound.

(1) For packages of variable sizes. It suffices to show that RPP($\mathcal{L}_Q$), FRP($\mathcal{L}_Q$),

$\mathsf{MBP}(\mathcal{L}_Q)$ and $\mathsf{CPP}(\mathcal{L}_Q)$ are coNP-hard, $\mathsf{FP}^{\mathsf{NP}}$-hard, DP-hard and $\# \cdot \mathsf{P}$-hard for fixed SP queries, respectively, when Q_c is absent, and are in coNP, $\mathsf{FP}^{\mathsf{NP}}$, DP and $\# \cdot \mathsf{P}$, respectively, for varied SP queries, when Q_c is present. Observe that the lower bounds of $\mathsf{RPP}(\mathcal{L}_Q)$, $\mathsf{FRP}(\mathcal{L}_Q)$, $\mathsf{MBP}(\mathcal{L}_Q)$ and $\mathsf{CPP}(\mathcal{L}_Q)$ for data complexity, in the absence of Q_c, given in Theorem 4.5, 5.1, 5.2 and 5.3, respectively, are established by taking Q as a identity query, which is in SP. As a result, these lower bounds hold here. For the upper bound, obviously, the algorithms for $\mathsf{RPP}(\mathrm{FO})$, $\mathsf{FRP}(\exists\mathrm{FO}^+)$, $\mathsf{MBP}(\mathrm{FO})$, given in Theorem 4.1, 5.1 and 5.2, respectively, can carry over here. Furthermore, since the combined complexity of membership problem of SP queries is in PTIME, one can readily verify that these algorithms are in coNP, $\mathsf{FP}^{\mathsf{NP}}$ and DP, respectively. For $\mathsf{CPP}(\mathcal{L}_Q)$, it is easy to see that it is in PTIME to check if a given set N is valid for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B)$. Thus the problem is in $\# \cdot \mathsf{P}$.

(2) For packages of a constant size. It suffices to show that $\mathsf{RPP}(\mathcal{L}_Q)$, $\mathsf{FRP}(\mathcal{L}_Q)$, $\mathsf{MBP}(\mathcal{L}_Q)$ and $\mathsf{CPP}(\mathcal{L}_Q)$ are in PTIME, FP, PTIME and FP, respectively, for varied queries Q and Q_c in SP. Since the combined complexity of membership problem of SP queries is in PTIME, obviously, the algorithms for fixed Q and Q_c given in Corollary 6.1 can carry over here.

This completes the proof of Corollary 6.2. □

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 6.3. *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 4.5, 5.1, 5.2 and 5.3, respectively, for all the languages of Section 2.*

Proof. The lower bounds of RPP, FRP, MBP and CPP in the absence of Q_c, given in Theorems 4.5, 5.1, 5.2 and 5.3, respectively, 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 4.5, 5.1, 5.2 and 5.3 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 6.4. *For items,* RPP*,* FRP*,* MBP *and* CPP *have*

- *the same combined complexity as their counterparts in the absence of Q_c (Theorems 4.5, 5.1, 5.2, 5.3), and*
- *the same data complexity as their counterparts for packages with a constant bound (Corollary 6.1),*

for all the query languages given in Section 2.

Proof. We first verify the combined complexity results of RPP, FRP, MBP and CPP for items, and then show their data complexity.

(1) Combined complexity. The upper bounds of RPP, FRP, MBP and CPP, in absence of compatibility constraints, given in Theorem 4.5, 5.1, 5.2 and 5.3, respectively, obviously remain intact here. We next show the lower bounds. Observe that the lower bounds proofs of RPP and CPP in the absence of compatibility constraints, given in 4.5 and 5.3, respectively, use only top-1 package. Thus these lower bounds are still valid here. We next consider the lower bounds of FRP and MBP. Note that the lower bounds of FRP and MBP, given in Theorem 5.1 and 5.2, for DATALOG$_{\mathsf{nr}}$, FO and DATALOG, respectively, are established by using top-1 packages with one item. So these lower bounds remain intact. Now we only need to show that for items, (a) FRP (CQ) is $\mathsf{FP}^{\mathsf{NP}}$-hard and (b) MBP (CQ) is DP-hard.

(a) FRP *(CQ)*. We show that FRP (CQ) is $\mathsf{FP}^{\mathsf{NP}}$-hard by reduction from MAX-WEIGHT SAT (see the proof of Theorem 5.1 for the statement of SAT-UNSAT). Given an instance $(\mathcal{C}, \{w_1, \ldots, w_r\})$ of MAX-WEIGHT SAT, where $\mathcal{C}$ is a set of clauses $\{C_1, \ldots, C_r\}$ which are defined over variables in set $X = \{x_1, \ldots, x_m\}$, and for each $i \in [1, r]$, w_i is an integer weight associated with clause C_i, we define a database D to consist of one single relation I_{01} as shown in Figure 4.1, specified by schema $R_{01}(X)$, a query Q as a cartesian product of relation R_{01} to generate all truth assignments of X variables, and set $k = 1$. Furthermore, for each tuple t in $Q(D)$, we define the utility $f(t)$ as the sum of weights of clauses in $\mathcal{C}$ that are true under the truth assignment encoded by t.

By the definition of $f()$, one can readily verify that for any tuple $t \in Q(D)$, $\{t\}$ is a top-1 item selection for (Q, D, f) iff t encodes a truth assignment of X variables that satisfies a set of clauses with the most total weight. Thus it is a reduction.

(b) MBP *(CQ)*. We show that MBP (CQ) is DP-hard by reduction from SAT-UNSAT (see the proof of Theorem 4.5) (CQ case) for the statement of SAT-UNSAT). Given an instance (φ_1, φ_2) defined over $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$, respectively, we define a database D to consist of a single relation I_{01} as shown in Figure 4.1, specified by schema $R_{01}(X)$, a CQ query Q as a cartesian product of relation R_{01} to generate all truth assignments of $X \cup Y$ variables, and take $k = 1$. Furthermore, for any tuple $t \in Q(D)$, we define (i) $f(t) = 1$ if the truth assignment μ_X of X variables encoded by t makes φ_1 true, while the truth assignment μ_Y of Y variables also encoded by t make φ_2 false; and (ii) for any other tuple $t' \in Q(D)$, we define $f(t) = 2$. Finally, we set $B = 1$.

We next show that φ_1 is satisfiable and φ_2 is not satisfiable iff B is the maximum bound for $(Q, D, f, k = 1)$. By the definition of $f()$, φ_1 is satisfiable and φ_2 is not satisfiable iff there exists a tuple $t \in Q(D)$ such that $f(t) = 1$, and moreover, there exists no tuple $t' \in Q(D)$ such that $f(t') > 1$. Obviously, the latter holds iff $B = 1$ is the maximum bound for $(Q, D, f, k = 1)$.

(2) Data complexity. Obviously, the algorithms developed for Corollary 6.1 suffice for item selections when Q is fixed. As a result, the upper bounds give there hold here.

This completes the proof of Theorem 6.4. □

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 [36]. (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 [27]. (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.

7. 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 7.1). We then identify two query relaxation recommendation problems, and establish their complexity bounds (Section 7.2).

7.1. 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, 19] and preference queries [30]. 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 $\psi_u(\vec{u})$. Furthermore, 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 7.1. *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:*

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

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 1/1/2012, where the distance function for dates is $\mathsf{dist}_d()$*:*

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

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.

7.2. 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) $\mathsf{gap}(Q_\Gamma) \leq g$.

$\mathsf{QRPP}(\mathcal{L}_Q)$:	*The query relaxation recommendation 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 $k \geq 1$, 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 $\mathsf{QRPP}(\mathcal{L}_Q)$ 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 7.2. *For* $\mathsf{QRPP}(\mathcal{L}_Q)$*, 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. We first show the complexity results of $\mathsf{QRPP}(\mathcal{L}_Q)$ for combined complexity, and then for data complexity. Finally, we provide the complexity results for absent compatibility constraints.

(1) Combined complexity. We first establish the combined complexity bounds.

When $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$. It suffices to show that QRPP(CQ) is Σ_2^p-hard and $\mathsf{QRPP}(\exists$FO$^+)$ is in Σ_2^p.

Lower bound. We show that QRPP(CQ) is Σ_2^p-hard by reduction from the $\exists^*\forall^*$3DNF problem (see the proof of Lemma 4.2 for the statement of the problem). Given an instance $\varphi = \exists X \forall Y\, \psi(X, Y)$ of the $\exists^*\forall^*$3DNF problem, we define a database D, queries Q and Q_c in CQ, a collection Γ of distance functions, functions $\mathsf{cost}()$ and $\mathsf{val}()$, and constants C, B, k and g. We show that φ is true iff there exists a query relaxation Q_Γ of Q for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$, when $k = 1$. Assume $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$. The reduction is an extension of its counterpart given in the proof of Lemma 4.2.

(1) The database D consists of four relations I_{01}, $I_{\vee}$, $I_{\wedge}$ and $I_{\neg}$ given in Figure 4.1.

(2) We define a CQ query Q as follows:

$$Q(\vec{x}, c) \;=\; \big((R_{01}(x_1) \wedge \cdots \wedge R_{01}(x_m) \;\wedge\; R_{01}(c) \wedge c = 0)\big).$$

Here $\vec{x} = (x_1, \ldots, x_m)$, and the query Q generates all truth assignments of X variables by means of Cartesian products of R_{01}. We let $E = \{0\}$, and $Z = \emptyset$. That is, we only allow the Boolean value of 0 to be extended.

(3) We use a minor variation of the CQ query Q_c defined in the proof of Lemma 4.2:

$$Q_c(b) \;=\; \exists \vec{x}, \vec{y}, c \;\big(R_Q(\vec{x}, c) \;\wedge\; Q_Y(\vec{y}) \;\wedge\; Q_{\psi}(\vec{x}, \vec{y}, b) \;\wedge\; b = 0\big).$$

Here R_Q is the schema of the query answer $Q(D)$, and $Q_Y(\vec{y})$ generates all truth assignments of Y variables by means of Cartesian products of R_{01}. Query Q_{ψ} encodes the truth value of $\psi(X, Y)$ for given truth assignments μ_X and μ_Y. It returns $b = 1$ if $\psi(X, Y)$ is satisfied by μ_X and μ_Y, and $b = 0$ otherwise. Thus the query $Q_c(b)$ returns a nonempty set iff for a given set $N \subseteq Q(D)$ that encodes a valid truth assignment μ_X for X, there exists a truth assignment of Y that makes $\psi(X, Y)$ false.

(4) We define $B = 1$, $k = 1$, $C = 1$ and $g = 1$. Let Γ consist of a single distance function $\mathsf{dist}()$ defined on Boolean values: $\mathsf{dist}(1, 0) = \mathsf{dist}(0, 1) = 1$, and $\mathsf{dist}(0, 0) = \mathsf{dist}(1, 1) = 0$. In addition, we define $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$, and $\mathsf{cost}(\emptyset) = \infty$. These assure that each valid package consists of a single tuple from the query answer. For $N = \{s\}$, where $s = (x_1, \ldots, x_m, c)$, we define $\mathsf{val}(N) = 1$ if $c = 1$, and $\mathsf{val}(N) = -\infty$ otherwise. We also let $\mathsf{val}(\emptyset) = -\infty$.

Observe that there exists no package $N \subseteq Q(D)$ such that $\mathsf{val}(N) \geq B$ by the definitions of $Q, \mathsf{val}()$ and B.

We now verify that φ is true iff there exists a query relaxation Q_{Γ} of Q for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$.

$\boxed{\Rightarrow}$ First assume that φ is true. Then there exists a truth assignment μ_X^0 for X such that for all truth assignments μ_Y for Y, ψ is true. Define a relaxed query Q_{Γ}:

$$Q_{\Gamma}(\vec{x}, c) = \exists w_c \big(R_{01}(x_1) \wedge \cdots \wedge R_{01}(x_m) \;\wedge\; R_{01}(c) \wedge\; c = w_c \;\wedge\; \mathsf{dist}(w_c, 0) \leq 1\big).$$

Then Q_{Γ} returns $(\mu_X^0, c = 1)$ when Q_X generates μ_X^0, since in this case $w_c = 1$ and $\mathsf{dist}(w_c, 0) \leq 1$. Let N consist of the tuple representing $(\mu_X^0, c = 1)$. Then Q_{ψ} does not return $b = 0$ for μ_X^0 and hence, $Q_c(N, D)$ is empty. In addition, $\mathsf{gap}(Q_{\Gamma}) \leq g$, $\mathsf{val}(N) \geq B$ and $\mathsf{cost}(N) \leq C$, $N \subseteq Q_{\Gamma}(D)$. Therefore, Q_{Γ} is a relaxed query for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$.

$\boxed{\Leftarrow}$ Conversely, assume that φ is false. Then for all truth assignment μ_X for X, there exists a truth assignment μ_Y for Y such that ψ is not satisfied by μ_X and μ_Y. As shown in the proof of Lemma 4.2, no matter how we select N, as long as N consists of a truth assignment of X, Q_{ψ} returns $b = 0$ and hence, $Q_c(N, D)$ is nonempty. As a result, there exists no relaxed query for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$.

Upper bound. We show that $\mathsf{QRPP}(\exists\mathrm{FO}^+)$ is in Σ_2^p, by giving the following algorithm:

1. Guess (a) a relaxed query Q_{Γ} of Q based on the active domain of D, (b) k sets of CQ queries from Q_{Γ}, each of a polynomial cardinality, and (c) a tableau from D for each of these CQ queries. These yield a set $\mathcal{N} = \{N_i \mid i \in [1, k]\}$ such that $N_i \subseteq Q_{\Gamma}(D)$ for all $i \in [1, k]$.

2. For each $N_i \in \mathcal{N}$, check whether $Q_c(N_i, D) = \emptyset$. If so, continue; otherwise reject the guess and go back to step 1.
3. Check whether (a) $\mathsf{gap}(Q_\Gamma) \leq g$. Moreover, for each $N_i \in \mathcal{N}$, (b) $\mathsf{cost}(N_i) \leq C$ and (c) $\mathsf{val}(N_i) \geq B$. In addition, check whether $N_i \neq N_j$ for $i, j \in [1, k]$ and $i \neq j$. If all these conditions are satisfied, return "yes", and otherwise reject the guess and go back to step 1.

To give details for step 1, we use the following notion. Given a query Q and a set Γ of distance functions, we say that two relaxed queries Q_Γ and Q'_Γ of Q by Γ are *D-equivalent* if $Q_\Gamma(D) = Q'_\Gamma(D)$, where D is a database. When checking whether there exists a relaxation of Q for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$ it suffices to consider those relaxed queries that are not D-equivalent.

Recall the definition of relaxed queries Q_Γ of Q from Section 7. Given a predicate of the form $\mathsf{dist}(w_c, c) \leq d$ (resp. $\mathsf{dist}(u_x, x) \leq d$), where the domain of w_c (resp. u_x) is $\mathsf{dom}(R.A)$, the bound d is constrained by the active domain of $R.A$. That is, $d \leq l$, where l is the maximum distance between any two values in the active domain of $R.A$. In other words, for any $d > l, d' > l$, $\mathsf{dist}(w_c, c) \leq d$ (resp. $\mathsf{dist}(u_x, x) \leq d$) and $\mathsf{dist}(w_c, c) \leq d'$ (resp. $\mathsf{dist}(u_x, x) \leq d'$) are D-equivalent.

In light of this, step 1 is as follows: for predicate $\mathsf{dist}(w_c, c) \leq d$ (resp. $\mathsf{dist}(u_x, x) \leq d$), we guess two values from the active domain of $R.A$, and let d be the difference between the two values, in NP. In addition, note that step 2 is in coNP, while step 3 is in PTIME. Therefore, the algorithm is in Σ_2^p.

When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO. We show that QRPP is PSPACE-complete for DATALOG$_{\mathsf{nr}}$ and FO.

Lower bound. We show that QRPP is PSPACE-hard for DATALOG$_{\mathsf{nr}}$ by reduction from Q3SAT (see the proof of Theorem 4.1 for the statement of Q3SAT). Given an instance φ of Q3SAT, we define D, Q with Z and E, empty compatibility constraint Q_c, functions $\mathsf{cost}()$, $\mathsf{val}()$, Γ, C, B and g. We show that φ is true iff there exists a relaxation Q_Γ of Q for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$. In particular, we define $\mathsf{cost}(N) = |N|$ if $N \neq \emptyset$, $\mathsf{cost}(\emptyset) = \infty$ and set $C = 1$.That is, only single tuples constitute packages.

(1) The database D consists of a single relation, namely, I_{01} given in Figure 4.1 which is specified by schema $R_{01}(X)$.

(2) We define $Q(c) :- p(),\ R_{01}(c), c = 0$. Here $p()$ is the query given in the proof of Theorem 4.1 for DATALOG$_{\mathsf{nr}}$. We let $E = \{0\}$, and $Z = \emptyset$. That is, we only allow the Boolean value of 0 to be extended.

(3) We use the same val, B, k, Γ and g as given for the CQ case above.

From the proof of Theorem 4.1 for DATALOG$_{\mathsf{nr}}$, we know that the answer to $p()$ in D is nonempty iff φ is true. Then along the same lines as the proof for the CQ case given above, one can easily verify that φ is true iff there exists a relaxation Q_Γ of Q for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$.

Similarly, we show that QRPP is PSPACE-hard for FO by reduction from the membership problem for FO (see the proof of Theorem 4.1 for the statement of the problem). Given an instance (Q, D, t) of the membership problem for FO, we define query Q_1 to be $Q_1(c) = Q'(\vec{x}\,) \wedge R_{01}(c) \wedge c = 0$. Here Q' is the query given in the proof of Theorem 4.1 for FO. Let $E = \{0\}$ and $Z = \emptyset$. Then using the same D, empty compatibility constraint Q_c, functions $\mathsf{cost}()$, $\mathsf{val}()$, Γ, C, B and g defined above for DATALOG$_{\mathsf{nr}}$, one can easily verify that $t \in Q(D)$ iff there exists a relaxation Q_Γ for

$(Q_1, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$.

Upper bound. We give an NPSPACE algorithm for determining $\mathsf{QRPP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ is $\text{DATALOG}_{\mathsf{nr}}$ or FO, as follows:

1. Guess a relaxed query Q_Γ of Q based on the active domain of D, and a set $\mathcal{N} = \{N_i \mid i \in [1, k]\}$ such that each N_i has polynomially many items and $N_i \neq N_j$ when $i \neq j$.
2. For each $N_i \in \mathcal{N}$, check whether $Q_c(N_i, D) = \emptyset$. If so, continue, otherwise reject the guess and go back to step 1.
3. For each $N_i \in \mathcal{N}$, check whether $N_i \subseteq Q_\Gamma(D)$ and moreover, (a) $\mathsf{gap}(Q_\Gamma) \leq g$, (b) $\mathsf{cost}(N_i) \leq C$, and (c) $\mathsf{val}(N_i) \geq B$. If so, return "yes", and otherwise reject the guess and go back to step 1.

Step 1 involves guessing relaxed queries (as in the algorithm for $\mathsf{QRPP}(\exists\text{FO}^+)$ given above). Note that steps 2 and 3 are both in PSPACE. Hence the algorithm is in NPSPACE = PSPACE.

When $\mathcal{L}_Q$ is DATALOG. We show that QRPP(DATALOG) is EXPTIME-complete.

Lower bound. We show that QRPP(DATALOG) is EXPTIME-hard by reduction from the membership problem for DATALOG (see the proof of Theorem 4.1 for the statement of the problem). The reduction is the same as the reduction for the FO case given above, except that here the query $p()$ is the one given in the proof of Theorem 4.1 for DATALOG.

Upper bound. We give an EXPTIME algorithm for deciding QRPP(DATALOG) as follows.

1. Enumerate all relaxed queries of Q up to D-equivalence.
2. For each such relaxed query Q_Γ, if $\mathsf{gap}(Q_\Gamma) \leq g$, then do the following.
   (a) Enumerate all subsets of $Q_\Gamma(D)$ consisting of polynomially many tuples.
   (b) For each $\mathcal{N}$ consisting of k such pairwise distinct subsets, and for each set N_i in $\mathcal{N}$, check: (i) whether $Q_c(N_i, D) = \emptyset$, and (ii) $\mathsf{cost}(N_i) \leq C$; and (iii) whether $\mathsf{val}(N_i) \geq B$. If all these conditions are satisfied, return "yes".
3. Return "no" after all Q_Γ up to D-equivalence and all $\mathcal{N}$ are inspected, if none satisfies the conditions above.

We show that step 1 is in EXPTIME. Indeed, following the same argument as given earlier for the algorithm for $\mathsf{QRPP}(\exists\text{FO}^+)$, it suffices to consider relaxed queries that are not D-equivalent. Given the set Z of variables and the set E of constants in Q, there exist at most $|D|^{|E|+|Z|}$ many relaxed queries of Q up to D-equivalence. Indeed, as argued above, for a predicate of the form $\mathsf{dist}(w_c, c) \leq d$ (resp. $\mathsf{dist}(u_x, x) \leq d$), where the domain of w_c (resp. u_x) is $\mathsf{dom}(R.A)$, the bound d is no larger than the maximum distance l between any two values in the active domain of $R.A$. Hence there exist at most l distinct relaxations of the predicate up to D-equivalence. From this follows the bound $|D|^{|E|+|Z|}$. Hence the algorithm is in EXPTIME. Step 2 is iterated exponentially many times, and each iteration takes EXPTIME [33]. Hence the algorithm is in EXPTIME.

(2) Data complexity. We now study the data complexity of $\mathsf{QRPP}(\mathcal{L}_Q)$.

Lower bound. We show that QRPP(CQ) is already NP-hard even in the absence of compatibility constraints Q_c. We verify this by reduction from 3SAT. Given an instance φ of 3SAT, we define Q, D, Γ, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C, B, k and g. We show that

φ is satisfiable iff there exists a query relaxation Q_Γ of Q for (Q, D, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C, B, k, g), when $k = 1$. Assume $\varphi = C_1 \wedge \cdots \wedge C_r$, and $X = \{ x_1, \ldots, x_m \}$

(1) Database D. The database is defined over a single relation $R_C(\mathsf{cid}, L_1, V_1, L_2, V_2, L_3, V_3, V)$. Its corresponding instance I_C consists of the following set of tuples. We know that for $i \in [1, r]$, $C_i = \ell_1^i \vee \ell_2^i \vee \ell_3^i$, where the ℓ_j^i's are variables or negation of variables in $X = \{x_1, \ldots, x_m\}$. For any possible truth assignment μ_i of variables in the literals in C_i that make C_i true, we add a tuple $(i, x_k, v_k, x_l, v_l, x_m, v_m, 1)$, where $x_k = \ell_1^i$ in case $\ell_1^i \in X$ and $x_k = \vec{\ell_1^i}$ in case $\ell_1^i = \vec{x}_k$. We set $v_k = \mu_i(x_k)$; similarly for x_l, x_m and v_l and v_m.

(2) Queries Q and Q_c. We define the query Q as follows:

$$Q(c, l_1, v_1, l_2, v_2, l_3, v_3, v) = \big(R_C(c, l_1, v_1, l_2, v_2, l_3, v_3, v) \wedge v = 0\big),$$

which simply selects tuples from I_C with their V-attribute set to 0. We let $E = \{0\}$ and $Z = \emptyset$. That is, we only allow the value 0 to be extended. Observe that $Q(D) = \emptyset$ since D only carries tuples with V-attribute equal to 1. In addition, we define Q_c to be the empty query.

(3) Rating function. We define $\mathsf{val}(N) = |N|$ and set $B = 1$.

(4) Cost function. We define $\mathsf{cost}(N) = 2$ in case that N contains two distinct tuples with the same cid value; or N contains two tuples s and t that contain the same variable x_i but s assigns it value 0 whereas t assigns it value 1, or when not all variables in X appear in N, or finally, when N does not contain a tuple for every cid value. Furthermore, for any other N, we define $\mathsf{cost}(N) = 1$. We set $C = 1$. Let Γ consist of a single distance function $\mathsf{dist}()$ defined on Boolean values: $\mathsf{dist}(1, 0) = \mathsf{dist}(0, 1) = 1$, and $\mathsf{dist}(0, 0) = \mathsf{dist}(1, 1) = 0$.

Note that there exists no package $N \subseteq Q(D)$ such that $\mathsf{val}(N) \geq B$ given that $Q(D) = \emptyset$ and $\mathsf{val}(\emptyset) < B$.

We now verify that φ is true iff there exists a query relaxation Q_Γ of Q for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$.

$\boxed{\Rightarrow}$ First assume that φ is satisfiable. Then there exists a truth assignment μ_X^0 for X that satisfies φ, *i.e.,* every clause C_j of φ is true with μ_{X0}. Define a relaxed query Q_Γ:

$$Q_\Gamma(c, l_1, v_1, l_2, v_2, l_3, v_3, w_c) = \big(R_C(c, l_1, v_1, l_2, v_2, l_3, v_3, w_c) \wedge \ \mathsf{dist}(w_c, 0) \leq 1\big).$$

Then $Q_\Gamma(D) = I_C$ when $w_c = 1$ (observe that if $w_c = 1$ then $\mathsf{dist}(w_c, 0) \leq 1$). Since μ_X^0 makes φ true, there exist r tuples in I_C, one for each clause in φ, such that the values of the variables in these tuples agree with μ_X^0. Let N consist of these r tuples. Then $\mathsf{val}(N) = r \geq B$ and $\mathsf{cost}(N) = 1 \leq C$. Therefore, Q_Γ is a relaxed query for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$.

$\boxed{\Leftarrow}$ Conversely, assume that φ is not satisfiable. Suppose by contradiction that there exists a relaxation Q_Γ. Let N be a valid package. Then, this would imply that I_C contains r tuples, one for each clause of φ, that together define a truth assignment μ_N for X which makes φ true. This contradicts the assumption that φ is not satisfiable.

Upper bound. To determine $\mathsf{QRPP}(\mathcal{L}_Q)$ when Q and Q_c are fixed, we use the same algorithm given above for $\mathsf{QRPP}(\mathrm{FO})$. Since Q and Q_c are fixed, steps 2 and 3 of that algorithm are in PTIME, when $\mathcal{L}_Q$ ranges over all the languages considered here.

Hence the algorithm is in NP, and so is $\mathsf{QRPP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ ranges over CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$, FO and DATALOG.

(3) Special case: In the absence of compatibility constraints. We next study $\mathsf{QRPP}(\mathcal{L}_Q)$ when Q_c is absent.

Combined complexity. When $\mathcal{L}_Q$ is CQ, observe the following. (a) We have shown above that QRPP(CQ) is NP-hard even when Q is fixed and Q_c is absent. (b) Recall the algorithm for $\exists$FO$^+$given above. When Q_c is absent, step 2 of the algorithm is not needed, and the algorithm is in NP. Putting these together, $\mathsf{QRPP}(\mathcal{L}_Q)$ is NP-complete when Q_c is absent, when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$.

When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$, FO or DATALOG, observe the following. (a) The upper bounds of $\mathsf{QRPP}(\mathcal{L}_Q)$ remain intact in the absence of Q_c. (b) The lower bound proofs of $\mathsf{QRPP}(\mathcal{L}_Q)$ given above do not use Q_c. Thus $\mathsf{QRPP}(\mathcal{L}_Q)$ has the same combined complexity irregardless of whether Q_c is present, for DATALOG$_{\mathsf{nr}}$, FO and DATALOG.

Data complexity. As shown above, QRPP(CQ) is NP-hard when Q is fixed and Q_c is absent. In addition, as argued above, $\mathsf{QRPP}(\mathcal{L}_Q)$ is in NP for fixed Q even when a fixed Q_c is present. Hence the data complexity of $\mathsf{QRPP}(\mathcal{L}_Q)$ is NP-complete when $\mathcal{L}_Q$ ranges over CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$, FO and DATALOG.

This completes the proof of Theorem 7.2. □

The query relaxation problem for items. We also study a special case of QRPP, for item selections. Given a database D, a query Q in which a set X of variables and a set E of constants are parameters that can be modified, a collection Γ of distance functions, a utility function $f()$ and constants C, B, g and $k \geq 1$, we define Q_Γ as a relaxation of Q for (Q, D, $f()$, C, B, k, g) is 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. The QRPP *for items* is to decide whether there exist a relaxation Q_Γ of Q for $(Q, D, Q_c, f(), C, B, k, g)$.

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

Corollary 7.3. *For all the query languages $\mathcal{L}_Q$ given in Section 2,* $\mathsf{QRPP}(\mathcal{L}_Q)$ *for items (1) has the same combined complexity as* $\mathsf{QRPP}(\mathcal{L}_Q)$ *in the absence of compatibility constraints; and (2) its data complexity is in* PTIME.

Proof. For $\mathcal{L}_Q$ ranging over CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$, FO and DATALOG, we first prove the combined complexity of $\mathsf{QRPP}(\mathcal{L}_Q)$ for items, and then show its data complexity.

(1) Combined complexity. We first verify the combined complexity bounds.

When $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$. It suffices to show that for items, QRPP(CQ) is NP-hard and QRPP($\exists$FO$^+$) is in NP.

Lower bound. We show that QRPP(CQ) is NP-hard when Q_c is absent by reduction from 3SAT (see the proof of RPP(DATALOG$_{\mathsf{nr}}$) given in Theorem 4.1 for the statement of 3SAT). Given an instance φ of 3SAT, we define a database D, a query Q in which a set X of variables and a set E of constants are parameters that can be modified, a collection Γ of distance functions, a utility function $f()$ and constants C, B, g and $k \geq 1$, We show that φ is satisfiable iff there exists a relaxation Q_Γ of Q for $(Q, D, f(), B, k, g)$. We assume that $\varphi = C_1 \wedge \cdots \wedge C_r$, with variables in

$X = \{x_1, \ldots, x_m\}$.

(1) The database D consists of relations I_{01}, $I_\vee$, $I_\wedge$ and $I_\neg$ given in Figure 4.1, specified over schemas $R_{01}(X)$, $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$ and $R_\neg(A, \vec{A})$.

(2) The query Q is defined as follows:

$$Q(c) = \exists \vec{x}\ \big(Q_X(\vec{x}\,) \wedge Q_\varphi(\vec{x}, c)\ \wedge\ c = 0\big),$$

where $\vec{x} = (x_1, \ldots, x_m)$, and $Q_X(\vec{x}\,)$ generates all truth assignments of X variables by means of Cartesian products of R_{01}. Query $Q_\varphi(\vec{x}, c)$ encodes the truth value of φ for a given truth assignment μ_X such that $c = 1$ if μ_X satisfies φ, and $c = 0$ otherwise. We let E consist of only 0, and Z be empty. That is, we only allow the Boolean value of 0 to be extended.

(3) We define the utility function $f()$ such that $f(c) = 0$ if $c = 0$, and $f(c) = 1$ otherwise. We also define $B = 1$ and $k = 1$.

(4) We define Γ consisting of a single distance function $\mathsf{dist}()$ defined on Boolean values: $\mathsf{dist}(1, 0) = \mathsf{dist}(0, 1) = 1$, and $\mathsf{dist}(0, 0) = \mathsf{dist}(1, 1) = 0$. Let $g = 1$.

Observe that there exists no item s from $Q(D)$ such that $f(s) \geq B$, since $Q(D)$ is either empty or a singleton set $\{(0)\}$.

We now verify that this is indeed a reduction.

$\boxed{\Rightarrow}$ First assume that φ is satisfiable. Then there is a truth assignment μ_X^0 for X that satisfies φ. Define a relaxed query:

$$Q_\Gamma(c) = \exists \vec{x}\ \exists w_c \big(Q_X(\vec{x}\,) \wedge Q_\varphi(\vec{x}, c)\ \wedge\ c = w_c\ \wedge\ \mathsf{dist}(w_c, 0) \leq 1\big).$$

Then $Q_\Gamma(c)$ returns $c = 1$ when Q_X generates μ_X^0, since in this case $w_c = 1$ and $\mathsf{dist}(w_c, 0) \leq 1$. Note that $\mathsf{gap}(Q_\Gamma) \leq g$ and $f(c) \geq B$. Hence Q_Γ is a relaxed query for (Q, D, f, B, k, g).

$\boxed{\Leftarrow}$ Conversely, assume that φ is not satisfiable. Then no matter how we extend the Boolean value c, we cannot find $Q_\Gamma(c)$ that returns $c = 1$. Hence there exists no relaxed query for (Q, D, f, B, k, g).

Upper bound. We show that $\mathsf{QRPP}(\exists\text{FO}^+)$ is in NP, by giving the following algorithm:

1. Guess (a) a relaxed query Q_Γ of Q based on the active domain of D, (b) k CQ queries from Q_Γ, and (c) a tableau from D for each of these k queries. These yield a set $S \subseteq Q_\Gamma(D)$.
2. Check whether (a) $\mathsf{gap}(Q_\Gamma) \leq g$, (b) for each $s \in S$, $f(s) \geq B$, and (c) S consists of k distinct items. If so, return "yes", and otherwise reject the guess and go back to step 1.

Recall from the algorithm for $\mathsf{QRPP}(\exists\text{FO}^+)$ that it suffices to consider those relaxed queries that are not D-equivalent. In light of this, step 1 is as follows: for predicate $\mathsf{dist}(w_c, c) \leq d$ (resp. $\mathsf{dist}(u_x, x) \leq d$), we guess two values from the active domain of $R.A$, and let d be the difference between these two values. Note that step 2 is in PTIME. Hence the algorithm is in NP.

When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$, FO or DATALOG. It suffices to observe that the lower bounds for $\mathsf{QRPP}(\mathcal{L}_Q)$ given in Theorem 7.2, for DATALOG$_{\mathsf{nr}}$, FO and DATALOG, respectively, are established by taking compatibility constraints Q_c as empty, and using top-1 package with one item. Then these lower bounds carry over to $\mathsf{QRPP}(\mathcal{L}_Q)$ for items. Moreover, obviously, the upper bounds given there also remain valid here. As a consequence, $\mathsf{QRPP}(\mathcal{L}_Q)$ is PSPACE-complete when $\mathcal{L}_Q$ is either DATALOG$_{\mathsf{nr}}$ or FO, and is $\mathsf{EXPTIME}$-complete when $\mathcal{L}_Q$ is DATALOG, for items.

(2) Data complexity. We show that when the query Q is fixed, $\mathsf{QRPP}(\mathcal{L}_Q)$ is in PTIME. Indeed, for a fixed Q there exist polynomially many relaxed queries up to D-equivalence, since $|E|$ and $|Z|$ are bounded. Hence we use the following algorithm:

1. Enumerate all relaxed queries of Q up to D-equivalence.
2. For each such relaxed query Q_Γ, if $\mathsf{gap}(Q_\Gamma) \leq g$, then do the following.
   (a) Compute $Q_\Gamma(D)$.
   (b) Find a set S of top-k items from $Q_\Gamma(D)$ if it exists. Check whether for all $s \in S$, $f(s) \geq B$. If so, return "yes".
3. Return "no" after all those relaxed queries of Q up to D-equivalence are inspected, if no one satisfies the condition above.

When Q is fixed, step 1 is in PTIME, as argued above. Step 2 is also in PTIME. Indeed, recall that relaxed queries have the form $Q_\Gamma(\vec{x}, \vec{w}, \vec{u}) = \exists \vec{w}\, \exists \vec{u}\, \big(Q'(\vec{x}, \vec{w}, \vec{u}) \wedge \psi_w(\vec{w}) \wedge \psi_u(\vec{u})\big)$. It takes PTIME to evaluate Q', ψ_w and ψ_u when Q is fixed. Hence the algorithm is in PTIME, when Q is a fixed query in CQ, UCQ, $\exists\mathrm{FO}^+$, $\mathrm{DATALOG}_{\mathsf{nr}}$, FO or DATALOG.

This completes the proof of Corollary 7.3. □

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 7.2 and Corollary 7.3 also tell us that for packages with a constant bound, $\mathsf{QRPP}(\mathcal{L}_Q)$ 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, $\mathsf{QRPP}(\mathcal{L}_Q)$ has the same combined and data complexity as its counterpart in the absence of Q_c. These are consistent with Corollaries 6.1 and 6.3.

8. 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 8.1). We then study adjustment recommendations problems (Section 8.2).

8.1. 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 $k \geq 1$, 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 $\mathsf{cost}(N) \leq C$.

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

8.2. Deciding Adjustment Recommendations. These suggest that we study the following problem.

The adjustment recommendation problem. Given a database D, a collection D' of items , queries Q and Q_c, functions $\mathsf{cost}()$ and $\mathsf{val}()$, and constants k and k', the *adjustment recommendation problem for packages*, ARPP, is to decide whether there is a package adjustment $\Delta(D, D')$ for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$.

ARPP($\mathcal{L}_Q$):	*The adjustment recommendation problem (packages)*
INPUT:	Database D and D', queries $Q, Q_c \in \mathcal{L}_Q$, two functions $\mathsf{cost}()$ and $\mathsf{val}()$, natural numbers C, B and $k, k' \geq 1$.
QUESTION:	Does there exist 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, ARPP($\mathcal{L}_Q$) has the same combined and data complexity as QRPP($\mathcal{L}_Q$), although their proofs are quite different.

THEOREM 8.1. *The combined complexity of* ARPP($\mathcal{L}_Q$) *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. Below we first give the combined complexity of ARPP($\mathcal{L}_Q$) when $\mathcal{L}_Q$ ranges over CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$, FO and DATALOG. We then verify its data complexity. Finally, we revisit ARPP($\mathcal{L}_Q$) in the absence of compatibility constraints Q_C.

(1) Combined complexity. We first verify the combined complexity bounds.

When $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$. It suffices to show that ARPP(CQ) is Σ_2^p-hard and ARPP($\exists$FO$^+$) is in Σ_2^p.

Lower bound. We show that ARPP(CQ) is Σ_2^p-hard by reduction from the $\exists^*\forall^*$3DNF problem (see the proof of Theorem 4.1 for the statement of the problem). Given an instance $\varphi = \exists X \forall Y\, \psi(X, Y)$ of the $\exists^*\forall^*$3DNF problem, we define a database D, a collection D' of items , queries Q and Q_c, functions $\mathsf{cost}()$ and $\mathsf{val}()$, and constants k and k'. We show that φ is true iff there exists a set $\Delta(D, D')$ of adjustments for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), B, C, k, k')$, when $k = 1$. We give the reduction as follows. Assume $X = \{ x_1, \ldots, x_m \}$ and $Y = \{ y_1, \ldots, y_n \}$.

(1) The database D consists of four relations $I_\vee$, $I_\wedge$ and $I_\neg$ as shown in Figure 4.1, and a unary relation $I_b = \emptyset$, which are specified by $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$, $R_\neg(A, \vec{A})$ and $R_{01}(X)$, respectively. We define D' to be the relation I_{01} given in Figure 4.1, consisting of Boolean values 0 and 1.

(2) We define a CQ query Q as follows:

$$Q(\vec{x}\,) \;=\; \exists z_1, z_0\, \big((R_{01}(z_1) \wedge z_1 = 1 \,\wedge\, R_{01}(z_0) \wedge z_0 = 0) \,\wedge\, (R_{01}(x_1) \wedge \cdots \wedge R_{01}(x_m))\big).$$

Here $\vec{x} = (x_1, \ldots, x_m)$, the sub-query $R_{01}(z_1) \wedge z_1 = 1 \;\wedge\; R_{01}(z_1) \wedge z_0 = 0$ assures that the updated relation I_{01} encodes the Boolean domain, and if so, $Q(\vec{x}\,)$ generates all truth assignments of X variables by means of Cartesian products of R_{01}.

(3) We use the same CQ query Q_c as defined in the proof of Lemma 4.2:

$$Q_c(b) \;=\; \exists\vec{x}\;\exists\vec{y}\;\big(R_Q(\vec{x}\,)\;\wedge\;Q_Y(\vec{y}\,)\;\wedge\;Q_\psi(\vec{x},\vec{y},b)\;\wedge\;b=0\big).$$

Here R_Q is the schema of the query answer $Q(D\oplus\Delta(D,D'))$, and $Q_Y(\vec{y}\,)$ generates all truth assignments of Y variables by means of Cartesian products of R_{01}. Query Q_ψ encodes the truth value of $\psi(X,Y)$ for given truth assignments μ_X and μ_Y. It returns $b=1$ if $\psi(X,Y)$ is satisfied by μ_X and μ_Y, and $b=0$ otherwise. The query $Q_c(b)$ returns a nonempty set iff for a given set $N\subseteq Q(D\oplus\Delta(D,D'))$ that encodes a valid truth assignment μ_X for X, there exists a truth assignment of Y that makes $\psi(X,Y)$ false.

(4) We define $\mathsf{cost}(N)=\mathsf{val}(N)=|N|$ if N is nonempty, and $\mathsf{cost}(N)=\infty$ and $\mathsf{val}(N)=-\infty$ otherwise. We define the cost budget $C=1$ and $B=1$. These assure that any package N selected has exactly one item. We also define $k=1$ and $k'=2$.

We next verify that φ is true iff there exists a set $\Delta(D,D')$ of adjustments for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),B,C,k,k')$.

$\boxed{\Rightarrow}$ First assume that φ is true. Then there exists a truth assignment μ_X^0 for X such that for all truth assignments μ_Y for Y, ψ is true. Let $\Delta(D,D')=I_{01}$, and N consist of the tuple representing μ_X^0. Then $|\Delta(D,D')|\le k'$, $Q_c(N,D\oplus\Delta(D,D'))$ is empty, $\mathsf{cost}(N)=1\le C$, and $\mathsf{val}(N)\ge 1=B$. Therefore, $\mathcal{N}=\{N\}$ is a top-1 package recommendation. In other words, this $\Delta(D,D')$ is indeed an adjustment for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C,B,k,k')$.

$\boxed{\Leftarrow}$ Conversely, assume that φ is false. Then for all truth assignment μ_X for X, there exists a truth assignment μ_Y for Y such that ψ is not satisfied by μ_X and μ_Y. As a result, no matter what $\Delta(D,D')$ we pick, either $D\oplus\Delta(D,D')$ does not encode the Boolean domain and hence $Q(D\oplus\Delta(D,D'))$ is empty; or for all N that satisfies $\mathsf{cost}(N)\le C$ and $\mathsf{val}(N)\ge B$, we have that $Q_c(N,D\oplus\Delta(D,D'))$ is nonempty. That is, there exists no $\Delta(D,D')$ that is an adjustment for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C,B,k,k')$.

Upper bound. We show that $\mathsf{ARPP}(\exists\mathrm{FO}^+)$ is in Σ_2^p, by giving the algorithm below:

1. Guess (a) a set $\Delta(D,D')$ of at most k' updates, (b) k sets of CQ queries from Q, each of a polynomial cardinality, and (c) a tableau from $D\oplus\Delta(D,D')$ for each of these CQ queries. These yield a set $\mathcal{N}=\{N_i\mid i\in[1,k]\}$ such that $N_i\subseteq Q(D\oplus\Delta(D,D'))$ for all $i\in[1,k]$.
2. For each $N_i\in\mathcal{N}$, check whether $Q_c(N_i,D\oplus\Delta(D,D'))=\emptyset$. If so, continue, and otherwise reject the guess and go back to step 1.
3. For each $N_i\in\mathcal{N}$, check whether (a) $\mathsf{cost}(N_i)\le C$ and (b) $\mathsf{val}(N_i)\ge B$. Furthermore, check whether $N_i\ne N_j$ for $i,j\in[1,k]$ and $i\ne j$. If so, return "yes", and otherwise reject the guess and go back to step 1.

Note that step 2 is in coNP, while step 3 is in PTIME. Hence the algorithm is in Σ_2^p.

When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$ or FO. We show that for DATALOG$_{\mathsf{nr}}$ and FO, ARPP is PSPACE-complete.

Lower bound. We show that ARPP is PSPACE-hard for DATALOG$_{\mathsf{nr}}$ by reduction from Q3SAT (see the proof of RPP(DATALOG$_{\mathsf{nr}}$) given in Theorem 4.1 for the statement of Q3SAT). Given an instance $\varphi=P_1x_1\ldots P_mx_m\;\psi(x_1,\ldots,x_m)$ of Q3SAT, we define Q, D, D', Q_c, C, $\mathsf{cost}()$, $\mathsf{val}()$, k,B and k'. We show that φ is true iff there exist adjustments $\Delta(D,D')$ for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C,B,k,k')$.

(1) The database D consists of a single unary relation $I_b = \emptyset$ which is defined over the schema $R_{01}(X)$, and $D' = \{(1), (0)\}$.

(2) The query Q is the same as its counterpart given in the proof of $\mathsf{RPP}(\text{DATALOG}_{\mathsf{nr}})$ given in Theorem 4.1

(3) We define $\mathsf{cost}(N) = |N|$ if N is nonempty and $\mathsf{cost}(\emptyset) = \infty$ otherwise. We define $k = 1$, $k' = 2$, $B = 1$ and $\mathsf{val}()$ as a constant function that returns 1 on any package.

Along the same lines as the proof of Theorem 4.1 for $\text{DATALOG}_{\mathsf{nr}}$, one can readily verify that φ is true iff there exist adjustments $\Delta(D, D')$ for (Q, D, Q_c, $\mathsf{cost}()$, $\mathsf{val}()$, C, B, k, k').

Similarly, we show that ARPP is PSPACE-hard for FO by reduction from the membership problem for FO (see the proof of Theorem 4.1 for the statement of the problem). Using the same D, D', Q_c, C, $\mathsf{cost}()$, $\mathsf{val}()$, k, B and k' defined above and the same query Q as given in the proof of Theorem 4.1 for FO, one can encode an instance (Q, D, t) of the membership problem for FO. One can easily verify that $t \in Q(D)$ iff there exist adjustments for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$.

Upper bound. We show that ARPP is in PSPACE for $\text{DATALOG}_{\mathsf{nr}}$ and FO, by presenting the following algorithm:

1. Guess $\Delta(D, D')$ of at most k' tuples from D and D', and a set $\mathcal{N} = \{N_i \mid i \in [1, k]\}$ such that each N_i has polynomially many items and $N_i \neq N_j$ when $i \neq j$.
2. For each $N_i \in \mathcal{N}$, check whether $N_i \subseteq Q(D \oplus \Delta(D, D'))$ and moreover, (a) $\mathsf{cost}(N_i) \leq C$, and (b) $\mathsf{val}(N_i) \geq B$. If so, continue, and otherwise reject the guess and go back to step 1.
3. For each $N_i \in \mathcal{N}$, check whether $Q_c(N_i, D \oplus \Delta(D, D')) = \emptyset$.
4. If so, return "yes', and otherwise reject the guess and go back to step 1.

Note that steps 2 and 3 are both in PSPACE. Hence the algorithm is in NPSPACE = PSPACE.

When $\mathcal{L}_Q$ is DATALOG. We show that ARPP(DATALOG) is EXPTIME-complete.

Lower bound. We show that ARPP(DATALOG) is EXPTIME-hard by reduction from the membership problem for DATALOG (see the proof of RPP(DATALOG) given in Theorem 4.1 for the statement of the problem). The reduction is the same as the one for the FO case given above, except that here the query Q is the one given in the proof of RPP(DATALOG) given in Theorem 4.1.

Upper bound. We show that ARPP(DATALOG) is in EXPTIME by giving the following algorithm.

1. Compute all $\Delta(D, D')$ of at most k' tuples from D and D'.
2. For each such $\Delta(D, D')$ do the following:
   (a) Enumerate all subsets of $Q(D \oplus \Delta(D, D'))$ consisting of polynomially many tuples.
   (b) For each $\mathcal{N}$ consisting of k such pairwise distinct subsets, and for each set N_i in $\mathcal{N}$, check: (a) whether $Q_c(N_i, D \oplus \Delta(D, D')) = \emptyset$, and (b) $\mathsf{cost}(N_i) \leq C$; and (c) whether $\mathsf{val}(N_i) \geq B$. If all these conditions are satisfied, return "yes".
3. Return "no" after all $\Delta(D, D')$ and all $\mathcal{N}$ are inspected, if none satisfies the conditions above.

Obviously, step 2 is executed exponentially many times in total, and each iteration takes EXPTIME [33]. Hence the algorithm is in EXPTIME.

(2) Data complexity. We show that when Q and Q_c are fixed, $\mathsf{ARPP}(\mathcal{L}_Q)$ is NP-complete for all the languages considered.

Lower bound. It suffices to show that $\mathsf{ARPP}(\mathrm{CQ})$ is already NP-hard by reduction from 3SAT. Given an instance $\varphi = C_1 \wedge \cdots \wedge C_r$ defined over a set $X = \{x_1, \ldots, x_m\}$ of variables, we define Q, D, D', Q_c, C, $\mathsf{cost}()$, $\mathsf{val}()$, k, B and k'. We show that φ is satisfiable iff there exists an adjustment $\Delta(D, D')$ for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$.

(1) The database D consists of three relations: (a) $I_X = \emptyset$ specified by $R_X = (X, V)$; (b) I_ψ specified by schema $R_\psi = (\mathsf{id}_C, P_x, X, V_x, w)$, where I_ψ encodes the clauses in ψ; for each $j \in [1, r]$, clause $C_j = \ell_1^j \vee \ell_2^j \vee \ell_3^j$ is encoded with six tuples in I_φ: $(j, i, x_{l_i}, v_i, w_i)$ for each $i \in [1, 3]$, where $x_{l_1}, x_{l_2}, x_{l_3}$ are variables in literals l_1^j, l_2^j, l_3^j, respectively, such that $w_i = 1$ if $v_i = 1$ and l_i^j is x_{l_i}, $w_i = 0$ if $v_i = 0$ and l_i^j is x_{l_i}, $w_i = 1$ if $v_i = 0$ and l_i^j is $\vec{x}_{l_i}$, and $w_i = 1$ if $v_i = 1$ and l_i^j is $\vec{x}_{l_i}$; and (c) relation $I_\vee$ given in Figure 4.1. The set D' is $\{\, (x_i, 0), (x_i, 1) \mid i \in [1, n]\}$, encoding truth values of X.

(2) We define the CQ query Q as follows:

$$
\begin{aligned}
Q(j, c, x, v, x', v') \;&=\; \exists j, x_1, x_2, x_3, v_1, v_2, v_3 \\
&\qquad \big(R_X(x_1, v_1) \wedge R_X(x_2, v_2) \wedge R_X(x_3, v_3) \;\wedge \\
&\qquad\quad Q_\varphi(j, x_1, x_2, x_3, v_1, v_2, v_3, c) \wedge \\
&\qquad\quad R_X(x, v) \wedge R_X(x', v')\big), \\
Q_\varphi(j, x_1, x_2, x_3, v_1, v_2, v_3, c) \;&=\; \exists w_1, w_2, w_3 \\
&\qquad \big(R_\psi(j, 1, x_1, v_1, w_1) \wedge R_\psi(j, 2, x_2, v_2, w_2) \wedge \\
&\qquad\quad R_\psi(j, 3, x_3, v_3, w_3) \;\wedge\; Q_\vee(w_1, w_2, w_3, c)\big).
\end{aligned}
$$

Here $Q_\vee$ computes $c = w_1 \vee w_2 \vee w_3$ by using the relation $I_\vee$. Intuitively, if $D \oplus \Delta(D, D')$ (*i.e.,* $\Delta(D, D')$ in this case) encodes a valid truth assignment μ_X for X, then query Q returns (j, c) for each clause C_j along with its truth value decided by μ_X. Moreover, it returns the Cartesian product of $\Delta(D, D')$. As will be seen shortly, this is to check whether $\Delta(D, D')$ encodes a valid truth assignment, *i.e.,* for every variable $x \in X$, there exists a unique truth value 0 or 1. This is enforced by using constants k, n, B, k' and function $\mathsf{val}()$ given below.

(3) We define Q_c to be the empty query. Let $\mathsf{cost}(N) = |N|$ if N is non-empty and $\mathsf{cost}(\emptyset) = \infty$ otherwise. We set $C = 1$. That is, packages consist of a single tuple only; in other words, we consider top-k item selections. We define $k = n * r$, where $n = |X|$ and r is the number of clauses in φ. We let $k' = n$ and $B = 1$. We define function $\mathsf{val}()$ such that $\mathsf{val}(\{(j, c, x, v, x', v')\}) = -1$ if (a) $c = 0$, or (b) $x \neq x'$, or (c) $x = x'$ but $v \neq v'$; we let $\mathsf{val}(\{(j, c, x, v, x', v')\}) = 1$ otherwise. Intuitively, this is to filter those tuples in $Q(D \oplus \Delta(D, D'))$ that do not denote a satisfied clause, or represent an invalid truth assignment to a variable in X.

We next show that this is indeed a reduction.

$\boxed{\Rightarrow}$ First assume that φ is satisfiable. Then there exists a truth assignment μ_X^0 for X that satisfies φ. Let $\Delta(D, D')$ include $(x_i, 1)$ if $\mu_X^0(x_i) = 1$, and $(x_i, 0)$ if $\mu_X^0(x_i) = 0$. Then for every clause C_j, Q returns $(j, 1, x, v, x', v')$. By the definition of $\mathsf{val}()$, only tuples of the form $(j, 1, x, v, x, v)$ are valid choices for all $x \in X$. Let N be the set of all such items. Obviously, $|\Delta(D, D')| \leq k'$, $|N| = k$, and for $\mathsf{val}(N) \geq B$. Hence there exists an adjustment for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$.

$\boxed{\Leftarrow}$ Conversely, assume that φ is not satisfiable. Then for any μ_X for X, there exists some C_j that is not satisfied by μ_X. By the definition of $\mathsf{val}()$, there exist no k distinct tuples in $Q(D \oplus \Delta(D, D'))$ with their $\mathsf{val}()$-ratings greater than B. Hence there exists no adjustment for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$.

Upper bound. For the upper bound, consider the same algorithm developed above for FO (combined complexity). We apply the algorithm to checking $\mathsf{ARPP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ is any language considered. When Q and Q_c are fixed, both step 2 and 3 are in PTIME. Hence the algorithm is in NP, for CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$, FO and DATALOG.

(3) Special case: In the absence of compatibility constraints. We now study $\mathsf{ARPP}(\mathcal{L}_Q)$ when Q_c is absent.

Combined complexity. When $\mathcal{L}_Q$ is CQ, observe the following. (a) As argued above, ARPP(CQ) is NP-hard even when Q is fixed and Q_c is absent by the proof of data complexity given earlier. (b) Consider the algorithm for $\exists$FO$^+$given above. When Q_c is absent, step 2 of the algorithm is not needed, and the algorithm is in NP. Hence when Q_c is absent, $\mathsf{ARPP}(\mathcal{L}_Q)$ is NP-complete for $\mathcal{L}_Q$ ranging over CQ, UCQ and $\exists$FO$^+$.

When $\mathcal{L}_Q$ is DATALOG$_{\mathsf{nr}}$, FO or DATALOG, observe the following. (a) The upper bounds of $\mathsf{ARPP}(\mathcal{L}_Q)$ remain intact in the absence of Q_c. (b) As remarked earlier, the lower bound proofs of $\mathsf{ARPP}(\mathcal{L}_Q)$ do not use compatibility constraints. Putting these together, we have that in the absence of Q_c, $\mathsf{ARPP}(\mathcal{L}_Q)$ has the same complexity as $\mathsf{ARPP}(\mathcal{L}_Q)$ in the presence of Q_c.

Data complexity. Recall that ARPP(CQ) is NP-hard when Q is fixed and Q_c is absent. In addition, the data complexity of $\mathsf{ARPP}(\mathcal{L}_Q)$ is in NP as shown above, for FO and DATALOG. Hence the data complexity of $\mathsf{ARPP}(\mathcal{L}_Q)$ is NP-complete for $\mathcal{L}_Q$ ranging over CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$, FO and DATALOG.

This completes the proof of Theorem 8.1. $\square$

The adjustment recommendation problem for items. Given a database D, a collection D' of items, a query Q, a utility function $f()$ and constants B, k and k', we say a set $\Delta(D, D')$ is an adjustment for $(Q, D, f(), C, B, k, k')$ if $\Delta(D, D')$ is an adjustment 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). Then ARPP *for items* is to decide whether there is an adjustment $\Delta(D, D')$ for $(Q, D, f(), C, B, k, k')$.

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 8.2. *For all the languages $\mathcal{L}_Q$ given in Section 2,* ARPP *for items has the same combined and data complexity as* ARPP *in the absence of compatibility constraints.*

Proof. Below we first give the combined complexity of ARPP for items, for CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$, FO and DATALOG. We then provide its data complexity.

Table 8.1

Combined complexity ($^{(\star)}$: *items (Th.6.1)*, $^{(\S)}$: *constant bound (Cor. 6.1)*, $^{(\dagger)}$: PTIME Q_c *(Cor.6.3))*

Problems	Languages	with Q_c	Without Q_c
RPP	CQ, UCQ, $\exists$FO$^+$ DATALOG$_{\mathsf{nr}}$, FO DATALOG	Π_2^p-complete$^{(\S)}$ PSPACE-complete$^{(\S)}$ EXPTIME-complete$^{(\S)}$ (Th. 4.1)	DP-complete$^{(\star,\dagger)}$ PSPACE-complete$^{(\star,\dagger)}$ EXPTIME-complete$^{(\star,\dagger)}$ (Th. 4.5)
FRP	CQ, UCQ, $\exists$FO$^+$ DATALOG$_{\mathsf{nr}}$, FO $^{(\star,\dagger)}$ DATALOG	FP$^{\Sigma_2^p}$-complete$^{(\S)}$ FPSPACE(poly)-complete$^{(\S)}$ FEXPTIME(poly)-complete$^{(\S)}$ (Th. 5.1)	FP$^{\mathsf{NP}}$-complete$^{(\star,\dagger)}$ FPSPACE(poly)-complete$^{(\star,\dagger)}$ FEXPTIME(poly)-complete$^{(\star,\dagger)}$ (Th. 5.1)
MBP	CQ, UCQ, $\exists$FO$^+$ DATALOG$_{\mathsf{nr}}$, FO DATALOG	D$_2^p$-complete$^{(\S)}$ PSPACE-complete$^{(\S)}$ EXPTIME-complete$^{(\S)}$ (Th. 5.2)	DP-complete$^{(\star,\dagger)}$ PSPACE-complete$^{(\star,\dagger)}$ EXPTIME-complete$^{(\star,\dagger)}$ (Th. 5.2)
CPP	CQ, UCQ, $\exists$FO$^+$ DATALOG$_{\mathsf{nr}}$, FO DATALOG	#·coNP-complete$^{(\S)}$ #·PSPACE-complete$^{(\S)}$ #·EXPTIME-complete$^{(\S)}$ (Th. 5.3)	#·NP-complete$^{(\star,\dagger)}$ #·PSPACE-complete$^{(\star,\dagger)}$ #·EXPTIME-complete$^{(\star,\dagger)}$ (Th. 5.3)
QRPP	CQ, UCQ, $\exists$FO$^+$ DATALOG$_{\mathsf{nr}}$, FO DATALOG	Σ_2^p-complete$^{(\S)}$ PSPACE-complete$^{(\S)}$ EXPTIME-complete$^{(\S)}$ (Th. 7.2)	NP-complete$^{(\star,\dagger)}$ PSPACE-complete$^{(\star,\dagger)}$ EXPTIME-complete$^{(\star,\dagger)}$ (Th. 7.2)
ARPP	CQ, UCQ, $\exists$FO$^+$ DATALOG$_{\mathsf{nr}}$, FO DATALOG	Σ_2^p-complete$^{(\S)}$ PSPACE-complete$^{(\S)}$ EXPTIME-complete$^{(\S)}$ (Th. 8.1)	NP-complete$^{(\star,\dagger)}$ PSPACE-complete$^{(\star,\dagger)}$ EXPTIME-complete$^{(\star,\dagger)}$ (Th. 8.1)

(1) Combined complexity. Recall that the lower bounds for ARPP($\mathcal{L}_Q$) given in Theorem 8.1, for CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$, FO and DATALOG, respectively, are established by using empty compatibility constraints and top-1 package with one item. Thus, these lower bounds are still valid here. For the upper bound, obviously, the upper bounds given there for ARPP($\mathcal{L}_Q$) can carry over here for items.

(2) Data complexity. We show that when the input query Q is fixed, ARPP($\mathcal{L}_Q$) for items is NP-complete. Indeed, as shown in the proof of Theorem 8.1 for data complexity, ARPP(CQ) for items is already NP-hard when Q is fixed by using top-k item selections. In addition, the algorithm given for ARPP(FO) in the proof of Theorem 8.1 can be used to check whether there exist item adjustments for (Q, D, B, f, k, k'), for all languages considered here. It is in NP when Q is fixed. Hence it shows that ARPI($\mathcal{L}_Q$) is in NP when $\mathcal{L}_Q$ is FO or DATALOG and when Q is fixed.

This completes the proof of Corollary 8.2. □

Remarks. One can find the following from the proofs of Theorem 8.1 and Corollary 8.2. (1) For packages with a constant bound, ARPP($\mathcal{L}_Q$) has the same combined complexity as ARPP($\mathcal{L}_Q$) for packages with variable sizes, and it has the same data complexity as ARPP($\mathcal{L}_Q$) for items. (3) When Q_c is in PTIME, ARPP($\mathcal{L}_Q$) has the same combined and data complexity as ARPP($\mathcal{L}_Q$) in the absence of Q_c.

9. Conclusions. We have studied a general model for recommendation systems, and investigated several fundamental problems in the model, from decision problems

TABLE 8.2
Data complexity ($^{(\star)}$*: items (Th. 6.1),* $^{(\S)}$*: constant bound (Cor. 6.1),* $^{(\dagger)}$*:* PTIME Q_c *(Cor. 6.3))*

Problems	Poly-bounded	Constant bound
RPP	coNP-complete$^{(\dagger)}$ (Th. 4.1)	PTIME $^{(\star)}$ (Cor. 6.1)
FRP	$\mathsf{FP}^{\mathsf{NP}}$-complete$^{(\dagger)}$ (Th. 5.1)	FP $^{(\star)}$ (Cor. 6.1)
MBP	DP-complete$^{(\dagger)}$ (Th. 5.2)	PTIME $^{(\star)}$ (Cor. 6.1)
CPP	#·P-complete$^{(\dagger)}$ (Th. 5.3)	FP $^{(\star)}$ (Cor. 6.1)
QRPP	NP-complete$^{(\dagger)}$ (Th. 7.2)	PTIME $^{(\star)}$ (Cor. 7.3)
ARPP	NP-complete$^{(\dagger)}$ (Th. 8.1)	NP-complete$^{(\star)}$ (Cor. 8.2)

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 8.2 and 8.2 for combined complexity and data complexity, respectively, annotated with their corresponding theorems (the results for SP (Corollary 6.2) 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 8.2. (2) The complexity bounds of these problems for CQ, UCQ and $\exists$FO$^+$*vary* when Q_c is present or not, and when packages have a constant bound or not. In contrast, the bounds for FO, 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 8.2 and 8.2).

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 cost(), 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.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[2] Gediminas Adomavicius and Alexander Tuzhilin. Multidimensional recommender systems: A

data warehousing approach. In *WELCOM*, 2001.

[3] Gediminas Adomavicius and Alexander 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] Sihem Amer-Yahia. Recommendation projects at Yahoo! *IEEE Data Eng. Bull.*, 34(2):69–77, 2011.

[5] Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawla, Gautam Das, and Cong Yu. Group recommendation: Semantics and efficiency. *PVLDB*, 2(1):754–765, 2009.

[6] Albert Angel, Surajit Chaudhuri, Gautam Das, and Nick Koudas. Ranking objects based on relationships and fixed associations. In *EDBT*, 2009.

[7] Alexander Brodsky, Sylvia Henshaw, and Jon 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] Surajit Chaudhuri and Moshe Y. Vardi. On the equivalence of recursive and nonrecursive Datalog programs. *JCSS*, 54(1):61–78, 1997.

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

[11] Ting Deng, Wenfei Fan, and Floris Geerts. The complexity of package recommendation problems. In *PODS*, pages 261–272, 2012.

[12] Arnaud Durand, Miki Hermann, and Phokion G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *TCS*, 340(3):496–513, 2005.

[13] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66(4):614–656, 2003.

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

[15] Konstantin Golenberg, Benny Kimelfeld, and Yehoshua Sagiv. Optimizing and parallelizing ranked enumeration. *PVLDB*, 4(11):1028–1039, 2011.

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

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

[18] Abhijit Kadlag, Amol V. Wanjari, Juliana Freire, and Jayant R. Haritsa. Supporting exploratory queries in databases. In *DASFAA*, 2004.

[19] Nick Koudas, Chen Li, Anthony K. H. Tung, and Rares Vernica. Relaxing join and selection queries. In *VLDB*, 2006.

[20] Georgia Koutrika, Benjamin Bercovitz, and Hector Garcia-Molina. FlexRecs: expressing and combining flexible recommendations. In *SIGMOD*, 2009.

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

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

[23] Theodoros Lappas, Kun Liu, and Evimaria Terzi. Finding a team of experts in social networks. In *KDD*, 2009.

[24] Chengkai Li, Mohamed A. Soliman, Kevin Chen-Chuan Chang, and Ihab F. Ilyas. RankSQL: supporting ranking queries in relational database management systems. In *VLDB*, 2005.

[25] Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. Supporting incremental join queries on ranked inputs. In *VLDB*, 2001.

[26] Christos H Papadimitriou. *Computational Complexity*. AW, 1994.

[27] Aditya G. Parameswaran, Hector Garcia-Molina, and Jeffrey D. Ullman. Evaluating, combining and generalizing recommendations with prerequisites. In *CIKM*, 2010.

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

[29] Karl Schnaitter and Neoklis Polyzotis. Evaluating rank joins with optimal cost. In *PODS*, 2008.

[30] Kostas Stefanidis, Georgia Koutrika, and Evaggelia Pitoura. A survey on representation, composition and application of preferences in database systems. *TODS*, 36(3), 2011.

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

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

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

[34] Sergei Vorobyov and Andrie Voronkov. Complexity of nonrecursive logic programs with complex values. In *PODS '98*, pages 244–253, 1998.

[35] Michael Wooldridge and Paul E. Dunne. On the computational complexity of qualitative

coalitional games. *Artif. Intell.*, 158(1):27–73, 2004.

[36] Min Xie, Laks V. S. Lakshmanan, and Peter T. Wood. Breaking out of the box of recommendations: from items to packages. In *RecSys*, 2010.