Information Systems 48 (2015) 64–88

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

On recommendation problems beyond points of interest

Ting Deng [a,c], Wenfei Fan [b,c], Floris Geerts [d,*]

[a] School of Computer Science and Engineering, Beihang University Beijing, No. 37 XueYuan Road, 100191 Beijing, China
[b] School of Informatics, University of Edinburgh, 10 Crichton Street, Edinburgh EH8 9AB, United Kingdom
[c] RCBD and SKLSDE Lab, Beihang University Beijing, No. 37 XueYuan Road, 100191 Beijing, China
[d] Department of Mathematics and Computer Science, University of Antwerp, Middelheimlaan 1, B-2020 Antwerpen, Belgium

ARTICLE INFO

Article history:
Received 27 February 2013
Received in revised form
27 August 2014
Accepted 31 August 2014
Recommended by: D. Suciu
Available online 16 September 2014

Keywords:
Recommendation problems
Query relaxation
Adjustment
Complexity

ABSTRACT

Recommendation systems aim to recommend items or packages of items that are likely to be of interest to users. Previous work on recommendation systems has mostly focused on recommending points of interest (POI), to identify and suggest top-k items or packages that meet selection criteria and satisfy compatibility constraints on items in a package, where the (packages of) items are ranked by their usefulness to the users. As opposed to prior work, this paper investigates two issues beyond POI recommendation that are also important to recommendation systems. When there exist no sufficiently many POI that can be recommended, we propose (1) *query relaxation recommendation* to help users revise their selection criteria, or (2) *adjustment recommendation* to guide recommendation systems to modify their item collections, such that the users' requirements can be satisfied.

We study two related problems, to decide (1) whether the query expressing the selection criteria can be relaxed to a limited extent, and (2) whether we can update a bounded number of items, such that the users can get desired recommendations. We establish the upper and lower bounds of these problems, *all matching*, for both combined and data complexity, when selection criteria and compatibility constraints are expressed in a variety of query languages, for both item recommendation and package recommendation. To understand where the complexity comes from, we also study the impact of variable sizes of packages, compatibility constraints and selection criteria on the analyses of these problems. Our results indicate that in most cases the complexity bounds of query relaxation and adjustment recommendation are comparable to their counterparts of the basic recommendation problem for testing whether a given set of (resp. packages of) items makes top-k items (resp. packages). In other words, extending recommendation systems with the query relaxation and adjustment recommendation functionalities typically does not incur extra overhead.

1. Introduction

Recommendation systems are also known as recommender systems, recommendation engines and platforms. Such systems are widely used to identify and suggest information items (e.g., movies, TV, news, books) or social elements (e.g., people, friends, groups or events in social networks) that are likely to be of interest to users. Traditional recommendation systems aim to find top-k items from a collection of items, e.g., books, events, Web sites and research papers [1], which satisfy certain selection criteria identified for a user, and are ranked by their

* Corresponding author.
E-mail addresses: dengting@act.buaa.edu.cn (T. Deng), wenfei@inf.ed.ac.uk (W. Fan), floris.geerts@uantwerpen.be (F. Geerts).

http://dx.doi.org/10.1016/j.is.2014.08.002
0306-4379/

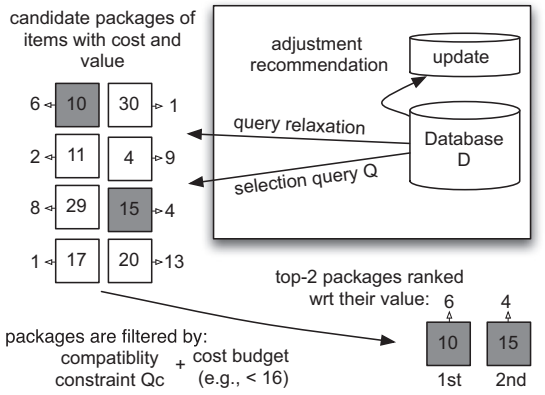

Fig. 1. Overview of package recommendation framework: Candidate packages get selected by query Q from database D; each package comes equipped with a cost (shown inside the package) and value; valid (gray shaded) packages are selected based on compatibility constraints Q_c and cost budget; and are finally ranked according to their value. In this paper, we add query relaxation and adjustment recommendation to the framework.

value of a utility function. More recently recommendation systems are often used to find top-k packages, i.e., *sets of items*, such as travel plans [2], teams of players [3] and various course combinations and prerequisites [4–6]. The items in a package are required not only to meet the selection criteria for each *individual* item, but also to satisfy compatibility constraints defined on all the items in a package *taken together*. Packages may have variable sizes subject to a cost budget, and are ranked by overall ratings of their items determined by a utility function [2]. Relational queries are often used to specify selection criteria and compatibility constraints [6–8,4,2], as illustrated below. An overview of the recommendation framework considered in this paper is depicted and illustrated in Fig. 1.

Example 1. Consider a recommendation system for travel plans, which maintains two relations specified by

flight(f#, From, To, DT, DD, AT, AD, Pr),

vista(name, city, type, ticket, time, dates).

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 vista tuple specifies a site name to visit in the city, its ticket price, type (e.g., museum, theater), the amount of time needed for the visit; there is an entry for each range of dates for which it is open to the public.

(1) Item recommendation. One wants to find top-3 flights from EDI (Edinburgh) to NYC (New York City) with at most one stop, departing on 1/1/2013, with lowest possible airfare and duration time. This can be stated as an item recommendation problem: (a) flights are items; (b) the selection criteria are expressed as a union $Q_1 \cup Q_2$ of conjunctive queries (CQ), where Q_1 and Q_2 select direct and one-stop flights from EDI to NYC on 1/1/2013, 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, where Dur can be derived from DT, DD, AT and AD, and $f()$ may associate different weights with Pr and Dur.

(2) Package recommendation. A user is planing a 5-day holiday, by taking a direct flight from EDI to NYC departing on 1/1/2013 and visiting as many places in NYC as possible. In addition, she does not want to have more than 2 museums in a package, which is a compatibility constraint [2]. Moreover, she wants the plans to have the lowest overall price.

This is an example of package recommendations: (a) the selection criteria are expressed as the following conjunctive query Q, which finds pairs of flight and vista tuples as items. That is, Q(f#, Pr, name, type, ticket, time, dates) is given by

$$\exists\, \mathsf{DT}, \mathsf{AT}, \mathsf{AD}, x_{\mathsf{To}} \big(\mathsf{flight}(\mathsf{f\#}, \mathsf{EDI}, x_{\mathsf{To}}, \mathsf{DT}, 1/1/2013, \mathsf{AT}, \mathsf{AD}, \mathsf{Pr}) \wedge \mathsf{vista}(\mathsf{name}, x_{\mathsf{To}}, \mathsf{type}, \mathsf{ticket}, \mathsf{time}, \mathsf{dates}) \wedge x_{\mathsf{To}} = \mathsf{NYC}\big);$$

(b) a package N consists of some items that have the same f# (and hence Pr); (c) the rating of N, denoted by 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 val(N) is; (d) the compatibility constraint requires that a package has no more than 2 museums, and can be expressed as another conjunctive query Q_c such that $Q_c(N) = \emptyset$, where Q_c is given by

$$Q_c() = \exists\, \mathsf{f\#}, \mathsf{Pr}, n_1, p_1, t_1, d_1, n_2, p_2, t_2, d_2, n_3, p_3, t_3, d_3 \big(R_Q(\mathsf{f\#}, \mathsf{Pr}, n_1, \mathit{museum}, p_1, t_1, d_1) \wedge R_Q(\mathsf{f\#}, \mathsf{Pr}, n_2, \mathit{museum}, p_2, t_2, d_2) \wedge R_Q(\mathsf{f\#}, \mathsf{Pr}, n_3, \mathit{museum}, p_3, t_3, d_3) \wedge (n_1 \neq n_2) \wedge (n_1 \neq n_3) \wedge (n_2 \neq n_3)\big).$$

Here R_Q denotes the schema of the query answer $Q(D)$; and (e) the cost of N, denoted by cost(N), is the total time taken for visiting all vista sites in N, which cannot exceed the time allocated for sightseeing in 5 days. Furthermore, cost(N) assigns $+\infty$ whenever the package contains vistas that are not open during the 5 day holiday, by using the dates information; such packages will thus not be recommended. Putting these together, the travel planning recommendation system is to find top-k such packages ranked by val(N), for a constant k chosen by the user. □

The need for studying recommendation systems is evident in Web services [2], Web search [9], social networks [9], education software [6] and commerce services [1]. There has been a host of work on recommendation problems, mostly focusing on algorithms for selecting and suggesting items or packages, known as *points of interest* (*POI*) [2], which meet selection criteria and satisfy compatibility constraints (see [1,9] for surveys). There has also been work on the complexity of computing POI recommendations [10,11,3,5,6,2].

There are other central issues beyond POI recommendation in connection with recommendation systems. In practice one often gets *no sensible recommendations*, i.e., the system fails to find items or packages that satisfy the user's needs. This may happen either when the selection criteria given by the user are too restrictive or when the

system does not have sufficiently many items from which recommendations can be made. When this happens, a recommendation system should be able to come up with recommendations for the users to *revise selection criteria*, or recommendations for the systems or vendors to *adjust their item collections*, as shown in Fig. 1.

Example 2. Consider again the user's request for recommending a 5-day holiday package, as described in Example 1.

(1) Query relaxation recommendations. One may not get any direct flight from EDI to NYC. Nevertheless, if we relax the query 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 help the user revise her selection criteria by recommending query relaxations.

(2) Adjustment recommendations. The relation vista in the recommendation system may consist of museums only, which users may not want to visit too many of (no more than two), as indicated by the compatibility constraint Q_c above. This motivates us to study adjustment recommendations, by recommending the system to include, e.g., theaters, in its vista collection. □

Recommendations for query relaxation and adjustments are as important as POI recommendations, and should logically be part of a practical recommendation system. However, no matter how important these issues are, no previous work has studied recommendations beyond POI. To support recommendations of query relaxation and adjustments, we need to settle the complexity of computing query relaxation and adjustments. Indeed, such additional functionalities should not come at too high an extra cost when compared to the complexity of the underlying system for recommending POI.

In our previous work [10,11] we have shown that the complexity of standard POI recommendation problems may depend on what query language we use to specify selection criteria and compatibility constraints; whether compatibility constraints are present or whether the packages of items are assumed to be of fixed size. In this paper we study the impact of these parameters on the complexity of query relaxation and adjustment recommendations. What is the complexity of query relaxation and adjustment 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? Do these complexity bounds differ from the counterparts of those POI recommendation problems studied in [10,11]? To the best of our knowledge, these questions have not been answered before.

Contributions. In this paper we study query relaxation and adjustment recommendations. Consider a recommendation system (see Section 2 for their formal definitions) in which (1) a database D collects items, (2) selection criteria and compatibility constraints are expressed as queries Q and Q_c, respectively, in a query language $\mathcal{L}_Q$, (3) a function cost() is defined on packages, and (4) a rating function val() is used to rank packages selected. A user request is stated as Q, Q_c, cost(), val(), a cost budget C and a positive integer k. It is to find k packages of items such that for each N of these packages, (1) N is a subset of $Q(D)$, the set of items selected by Q from D, (2) N satisfies the compatibility constraint Q_c, (3) cost(N) does not exceed the budget C, and moreover, (4) N is among the top-k of such packages ranked by val(). Now suppose that the system fails to find k such packages. We study the following problems.

(1) *Query relaxation recommendation* is to find a "minimum" relaxation Q_Γ of the query Q for selection criteria such that Q_Γ is able to find k packages from D that satisfy the user's needs.

(2) *Adjustment recommendation* is to find "minimum" updates $\Delta(D, D')$ to the item collection D, possibly by removing tuples from D and inserting into D some tuples taken from a collection D' of additional items, such that k packages required by the user can be found by query Q from $D \oplus \Delta(D, D')$, the database of items obtained by updating D with $\Delta(D, D')$.

We investigate the decision version of these problems, denoted by QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$), respectively. The study of these problems helps us understand whether we can get k packages that satisfy the user's requirements by minimally relaxing the selection criteria or minimally updating the item collection.

We parameterize each of these problems with various query languages $\mathcal{L}_Q$ in which selection criteria Q and compatibility constraints Q_c are expressed. We consider the following $\mathcal{L}_Q$, all with built-in predicates $=, \neq, <, \leq, >, \geq$:

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

We now illustrate the need for considering various $\mathcal{L}_Q$.

Example 3. In the package recommendation example given above, the selection criteria and compatibility constraints are expressed as conjunctive queries (CQ). However, suppose that the user does not impose a predefined bound on how many stops he can bear with during the flight, as long as the cost is minimal. Then we need to express the selection criteria in, e.g., DATALOG, which is more costly to evaluate than CQ.

As another example, if the user also requires that there are at least two theaters in a package N, we need to express the compatibility constraints Q_c in first-order logic (FO) as follows:

$$
\begin{aligned}
Q_c'() = \forall\ & \mathsf{f\#}, \mathsf{Pr}, n_1, p_1, t_1, d_1, n_2, p_2, t_2, d_2 \\
& (R_Q(\mathsf{f\#}, \mathsf{Pr}, n_1, \mathit{theater}, p_1, t_1, d_1) \\
& \wedge R_Q(\mathsf{f\#}, \mathsf{Pr}, n_2, \mathit{theater}, p_2, t_2, d_2) \\
& \quad \rightarrow (n_1 = n_2 \wedge\ p_1 = p_2 \wedge\ t_1 = t_2)),
\end{aligned}
$$

where as before, R_Q denotes the relation schema of the query result $Q(D)$. In other words, query languages beyond CQ naturally arise in application scenarios. □

Furthermore, apart from the queries languages $\mathcal{L}_Q$ given above for specifying selection criteria and compatibility constraints, we also study special cases of these problems in terms of the size of packages and more simpler queries than CQ. More specifically, we consider the following special cases:

(a) when packages have a fixed size B_p;
(b) when $\mathcal{L}_Q$ is SP (selection-projection queries) for which the membership problem is in PTIME;
(c) when compatibility constraints are simple PTIME functions or are absent (i.e., empty query); and
(d) when item recommendations are considered, where each package has a single item and compatibility constraints are absent.

(3) *Complexity results*. We establish upper and lower bounds for QRP and ARP, *all matching*, for their combined complexity and data complexity, for various $\mathcal{L}_Q$ and for the special cases mentioned earlier. The main complexity results are summarized in Table 1, and the complexity bounds of the special cases are shown in Table 2. These results indicate where the complexity of query relaxation and adjustment recommendations comes from. More specifically, we find the following.

(1) The complexity of query evaluation is the dominating factor for the combined complexity analyses of query relaxation and adjustment recommendations. That is, the higher the complexity of query evaluation is, the higher the combined complexity bounds of the corresponding query relaxation and adjustment recommendations are. More specifically, QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) are both NP-complete when $\mathcal{L}_Q$ is SP, Σ_2^p-complete when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$, PSPACE-complete when $\mathcal{L}_Q$ is DATALOG$_{\text{nr}}$ or FO, and EXPTIME-complete for DATALOG.

(2) In contrast, query languages have no impact on the data complexity analyses of query relaxation and adjustment recommendations. More specifically, QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) are NP-complete when $\mathcal{L}_Q$ is any of SP, CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\text{nr}}$, FO, or DATALOG. This is because when a query Q is fixed, the evaluation of Q in a database D is in PTIME in the size of D no matter whether Q is in SP, FO or DATALOG.

These combined and data complexity bounds are rather robust: the lower bounds of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) remain intact when $k=1$, i.e., for selecting top-1 packages, for $\mathcal{L}_Q$ ranging over all the query languages mentioned above.

(3) For combined complexity, variable package sizes do not make our lives harder. Indeed, when $\mathcal{L}_Q$ is SP, CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\text{nr}}$, FO, or DATALOG, the combined complexity bounds of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) remain

Table 1
Main complexity results.

Problems	Languages	Combined complexity	Data complexity
QRP	SP	NP-complete (Cor. 1)	NP-complete (Cor. 1)
	CQ, UCQ, $\exists$FO$^+$	Σ_2^p-complete (Th. 1)	NP-complete (Th. 2)
	DATALOG$_{\text{nr}}$, FO	PSPACE-complete (Th. 1)	NP-complete (Th. 2)
	DATALOG	EXPTIME-complete (Th. 1)	NP-complete (Th. 2)
ARP	SP	NP-complete (Cor. 1)	NP-complete (Cor. 1)
	CQ, UCQ, $\exists$FO$^+$	Σ_2^p-complete (Th. 3)	NP-complete (Th. 4)
	DATALOG$_{\text{nr}}$, FO	PSPACE-complete (Th. 3)	NP-complete (Th. 4)
	DATALOG	EXPTIME-complete (Th. 3)	NP-complete (Th. 4)

Table 2
Special cases.

Conditions	Languages	QRP($\mathcal{L}_Q$)		ARP($\mathcal{L}_Q$)	
		Combined complexity	Data complexity	Combined complexity	Data complexity
$\lvert N\rvert \le B_p$ (Th. 5)	SP	NP-complete	PTIME	NP-complete	NP-complete
	CQ, UCQ, $\exists$FO$^+$	Σ_2^p-complete	PTIME	Σ_2^p-complete	NP-complete
	DATALOG$_{\text{nr}}$, FO	PSPACE-complete	PTIME	PSPACE-complete	NP-complete
	DATALOG	EXPTIME-complete	PTIME	EXPTIME-complete	NP-complete
PTIME Q_c (Cor. 3)	SP	NP-complete	NP-complete	NP-complete	NP-complete
	CQ, UCQ, $\exists$FO$^+$	NP-complete	NP-complete	NP-complete	NP-complete
	DATALOG$_{\text{nr}}$, FO	PSPACE-complete	NP-complete	PSPACE-complete	NP-complete
	DATALOG	EXPTIME-complete	NP-complete	EXPTIME-complete	NP-complete
For items (Th. 6)	SP	NP-complete	PTIME	PTIME	PTIME
	CQ, UCQ, $\exists$FO$^+$	NP-complete	PTIME	NP-complete	NP-complete
	DATALOG$_{\text{nr}}$, FO	PSPACE-complete	PTIME	PSPACE-complete	NP-complete
	DATALOG	EXPTIME-complete	PTIME	EXPTIME-complete	NP-complete

unchanged no matter whether packages have a fixed size or variable sizes.

In contrast, fixing package size simplifies the data complexity analysis of QRP($\mathcal{L}_Q$): it is in PTIME as opposed to NP-complete for packages with variable sizes. However, ARP($\mathcal{L}_Q$) remains NP-complete when packages have a fixed size.

(4) The absence of compatibility constraints and PTIME compatibility constraints has the same impact on the analyses of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$). They make our lives easier when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$: both QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) become NP-complete rather than Σ_2^p-complete, for combined complexity. However, when $\mathcal{L}_Q$ is SP, DATALOG$_{nr}$, FO, or DATALOG, the combined complexity bounds of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) remain intact no matter whether compatibility constraints are present or absent. Furthermore, the absence of compatibility constraints has no impact on the data complexity of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) no matter what $\mathcal{L}_Q$ is, as expected.

(5) For item recommendation, i.e., when each package has a fixed size 1 and compatibility constraints are absent, QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) have the same combined and data complexity as their counterparts for package recommendation when $\mathcal{L}_Q$ is DATALOG$_{nr}$, FO or DATALOG. In contrast, for CQ, UCQ and $\exists$FO$^+$, QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) become NP-complete instead of Σ_2^p-complete for combined complexity, while their data complexity bounds are unchanged. When $\mathcal{L}_Q$ is SP, ARP is down to PTIME for combined and data complexity from NP-complete, while QRP remains NP-complete for combined complexity.

(6) In most cases QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) have the same combined and data complexity when $\mathcal{L}_Q$ ranges over all the query languages mentioned above. However, they differ from each other in the following three special cases. (a) When packages are bounded by a constant size, QRP($\mathcal{L}_Q$) is tractable while ARP($\mathcal{L}_Q$) is NP-complete for data complexity, for all these query languages; and (b) for item recommendations, QRP($\mathcal{L}_Q$) is NP-complete for combined complexity, but ARP($\mathcal{L}_Q$) is tractable, if $\mathcal{L}_Q$ is SP; and (c) for the data complexity of item recommendations, QRP($\mathcal{L}_Q$) is tractable but ARP($\mathcal{L}_Q$) is NP-complete, for CQ, UCQ, $\exists$FO$^+$, FO, DATALOG$_{nr}$ and DATALOG.

(7) Finally, the complexity bounds of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) are comparable to the counterparts of the basic POI recommendation problem. The latter problem, denoted by RPP($\mathcal{L}_Q$), is to decide whether a given set of packages is indeed a top-k package recommendation [10,11], in the same settings studied in this paper. This tells us that despite the costs introduced by the large search space of possible query relaxations and by updates to the underlying database, extending recommendation systems with query relaxation and adjustment recommendation does not incur significant overhead (see more details shortly).

The complexity bounds of these problems are not only of theoretical interest, but may also help practitioners when developing recommendation models and algorithms. Indeed, these results tell us where high complexity comes from, and thus help us decide what to support when developing practical recommendation systems. For instance, if users of the system are to use only a fixed set of DATALOG queries and limit the number of items in a package to a constant, then query relaxation recommendation is in PTIME (Theorem 5). In contrast, if the system allows users to issue arbitrary CQ queries, then one should be prepared for a higher cost (Σ_2^p-complete, Theorem 1). In the latter case, the developers of the system should go for heuristic algorithms for query relaxation rather than for exact algorithms, as suggested by the lower bounds of the problems.

To the best of our knowledge, this work is among the first efforts to study query relaxation recommendation and adjustment recommendations, beyond POI recommendation. A variety of techniques are used to prove the results, including a wide range of reductions and constructive proofs with algorithms. To focus on the main idea of recommendations beyond POI, we adopt a single general methodology for relaxations and adjustments, and investigate their complexity analyses in terms of different query languages. We defer to future work the study of different relaxation schemes and relaxation of compatibly constraints in addition to queries for expressing selection criteria.

Related work. This paper extends our prior work on recommendation beyond POI presented in [10], by including all detailed proofs and new results on various special cases of query relaxation and adjustment recommendations. All the results in Sections 5.1–5.3 are new, providing complexity bounds of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) 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 simple PTIME functions. Some results of Sections 5.4 on item recommendation are also new: we show that ARP(SP) is in PTIME, while all the results given in [10] for ARP($\mathcal{L}_Q$) are intractable. Moreover, we show that QRP(SP) is intractable in the same setting, which was not studied in [10].

To focus on recommendation analyses beyond POI, we do not include the analyses of POI recommendation problems in this paper, which were studied in [10,11]. More specifically, we consider the settings when there exists no top-k package that satisfies users' need while the users nevertheless want to find recommendations by relaxing the selection query or by adjusting the database. To extend recommendation systems with these functionalities, it is important to understand the complexity incurred by the extension. As remarked earlier, the support of query relaxation and adjustment recommendations does not come with a higher complexity. To see this, we summarize the differences between RPP($\mathcal{L}_Q$) and QRP($\mathcal{L}_Q$) or ARP($\mathcal{L}_Q$) in Table 3. From Table 3 we can see the following.

(1) Combined complexity. (a) When $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$, if RPP($\mathcal{L}_Q$) lies in complexity class C, then QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) are in coC, except for the special cases when PTIME compatibility constraints or item recommendations are considered (to be elaborated shortly). (b) When $\mathcal{L}_Q$ is FO, DATALOG$_{nr}$ or DATALOG, RPP($\mathcal{L}_Q$), QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) have the same complexity as RPP($\mathcal{L}_Q$), i.e., they are PSPACE-complete, PSPACE-complete and EXPTIME-complete, respectively.

(2) Data complexity. The only increase in complexity is when packages are constrained to be of bounded size. In this case RPP($\mathcal{L}_Q$) is in PTIME whereas ARP($\mathcal{L}_Q$) is NP-complete.

Table 3
Differences between RPP in [10,11], and QRP and ARP.

	Conditions	RPP	QRP\ARP
Combined complexity (CQ, UCQ, $\exists$FO$^+$)		Π_2^p-complete (Th. 4.1 [11])	Σ_2^p-complete (Th. 1, 3)
	SP	coNP-complete (Cor. 6.2 [11])	NP-complete (Cor. 1)
	$\|N\| \le B_p$	Π_2^p-complete (Cor. 6.1 [11])	Σ_2^p-complete (Th. 5)
	PTIME Q_c	DP-complete (Cor. 6.3 [11])	NP-complete (Cor. 3)
	Items	DP-complete (Th. 6.4 [11])	NP-complete (Th. 6)
Data complexity		coNP-complete (Th. 5.1 [11])	NP-complete (Th. 2, 4)
(CQ, UCQ, $\exists$FO$^+$, FO, DATALOG$_{nr}$, DATALOG)	SP	coNP-complete (Cor. 6.2 [11])	NP-complete (Cor. 1)
	$\|N\| \le B_p$	PTIME (Cor. 6.1 [11])	PTIME \NP-complete (Th. 5)
	PTIME Q_c	coNP-complete (Cor. 6.3 [11])	NP-complete (Cor. 3)
	Items	PTIME (Th. 6.4 [11])	PTIME \NP-complete (Th. 6)

(3) In particular, QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) have lower complexity bounds than RPP($\mathcal{L}_Q$) in the following cases (unless P=NP). When $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$, and for either (a) traditional item recommendation or (b) PTIME compatibility constraints, the combined complexity of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) is both NP-complete, as opposed to DP-complete for RPP($\mathcal{L}_Q$). This is because in these settings, QRP($\mathcal{L}_Q$) (resp. ARP($\mathcal{L}_Q$)) just needs to test whether there exists a relaxation (resp. adjustment) that yields k items (or packages) satisfying users' selection criteria; in contrast, RPP($\mathcal{L}_Q$) has to check whether a given item (or package) satisfies the users' criteria and moreover, is among top-k such items (or packages).

Taken together, these show that extending recommendation systems with query relaxation and adjustment recommendation comes, in general, with no extra cost compared to the complexity of the underlying POI recommendation framework.

This work is also related to previous work on recommender systems, top-k query answering, query result diversification and query relaxation, as discussed next.

Recommender systems. Traditional recommendation systems aim to find, for each user, items that maximize the user's utility (see, e.g., [1] for a survey). Selection criteria are decided by content-based, collaborative and hybrid approaches, which consider preferences of each user in isolation, or combined preferences of similar users [1]. Recently recommendation systems have been extended to find packages, which are presented to the user in a ranked order based on some rating function [12,3,5,6,2]. A number of algorithms have been developed for recommending packages of a fixed size [12,3] or variable sizes [5,6,2]. Compatibility constraints [3,5,6,2] and budget restrictions [2] on packages have also been studied. Several recommendation problems in connection with POI recommendations have been identified and investigated: (a) to decide whether a set of packages makes a top-k recommendation; (b) whether a given rating bound B is maximum for selecting top-k packages, i.e., when B is increased there exist no k packages such that val(N) $\ge B$ for each recommended package N; (c) to compute top-k packages if these exist; and (d) to find how many packages meet the user's criteria [10,11,3,5,6,2]. However, none of the prior work considers query relaxation or adjustment recommendation, which is the focus of this paper.

Top-k query answering. There has also been a host of work on top-k query answering. It aims to retrieve k-items (tuples) from the answers $Q(D)$ of a query Q in a database D such that the retrieved items are top-ranked by some scoring function [13]. This can be seen as recommending items taken from views of the data [7,8,4,14,2], where the views are expressed as relational queries, representing preferences or points of interest [7,8,4]. A number of top-k query evaluation algorithms have been developed (e.g., [14–16]; see [13] for a survey). A central issue there concerns how to combine different ratings of the same item based on multiple criteria.

This work differs from the previous work on top-k query answering in that we study query relaxation and adjustment recommendations, whereas top-k query answering is related to POI recommendations only. In addition, we focus on the complexity of relaxation and adjustment recommendation analyses, rather than the efficiency or optimization of query evaluation.

Query result diversification. Another line of work concerns query result diversification, which extends top-k query answering to a bi-criteria optimization problem, to recommend non-homogeneous (diverse) items [17–19]. Given a query Q, a database D, an objective function $F(\cdot)$ and a positive integer k, it is to find a set U of k tuples from $Q(D)$ such that the value $F(U)$ is maximum. Here $F(\cdot)$ characterizes max–sum diversification, max–min diversification or mono-objective formulation, in terms of two criteria: relevance and diversity [19]. In other words, we want the tuples in U to be as relevant as possible to query Q, and at the same time, as diverse as possible to each other. Query result diversification can be viewed as a special case of package recommendation that is to find a single set of items of a fixed size k, based on a particular objective function $F(\cdot)$. It differs from query relaxation and adjustment recommendations from problem statements to complexity bounds. For example, problem QRD($\mathcal{L}_Q, F(\cdot)$) studied in [17] is to decide, given a query $Q \in \mathcal{L}_Q$, a database D, an objective function $F(\cdot)$, a real number B and a positive integer k, whether there exists a subset $U \subseteq Q(D)$ such that $|U| = k$ and $F(U) \ge B$. It is shown in [17] that when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$, QRD($\mathcal{L}_Q, F(\cdot)$) is NP-complete when $F(\cdot)$ is max–sum or max–min diversification, while it is

PSPACE-complete when $F(\cdot)$ is mono-objective formulation. In contrast, for the same $\mathcal{L}_Q$, QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) are Σ_2^p-complete (see Table 1).

Query relaxation. Query relaxations have been studied in, e.g., [20–23]. Several query generalization rules are introduced in [20], assuming that query acceptance conditions are monotonic. Heuristic query relaxation algorithms are developed in [21,22]. The topic is also studied for top-k query answering [23]. Here we focus on the main idea of query relaxation recommendations, and borrow general query relaxation rules from [20]. 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 study more powerful languages such as FO and DATALOG. As illustrated earlier, DATALOG is needed when, e.g., users do not impose a bound on how many stops he can bear with during the flight, and FO needs to be used when users require at least two theaters in a package. Furthermore, the prior work focuses on the design of efficient relaxation algorithms, but does not study its computational complexity, which is the focus of this paper.

Organization. The remainder of the paper is organized as follows. Package and item recommendations are formally presented in Section 2. Section 3 formulates and studies query relaxation recommendation, followed by adjustment recommendation in Section 4. A variety of special cases of these two problems are studied in Section 5. Section 6 summarizes the main results of the paper and identifies open issues.

2. Modeling recommendations

In this section we specify recommendations of packages and items, and review the query languages considered in this work.

Item collections. Following [4–8,2], we assume a database D that collects items for users to select. The database is specified with a relational schema $\mathcal{R}$ composed 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 dom($R_i.A$).

Package recommendation. 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 [6], travel plans including multiple POI [2], and teams of experts [3]. Package recommendation is to find top-k packages such that the items in each package N (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 [6,2] 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. In Example 1, for instance, we use a conjunctive query Q to specify what flights and sites a user wants to find for her holiday plan.

Compatibility constraints. To specify the compatibility constraints for a package N, we use a query Q_c such that N satisfies Q_c if and only if $Q_c(N, D) = \varnothing$. That is, Q_c identifies inconsistencies among items in N. For instance, in Example 1, the conjunctive query Q_c is to assert the constraint "no more than 2 museums" in a travel package N [2].

As another example, for a course package N, we use a query Q_c to ensure that for each course in N, its prerequisites are also included in N [5]. Observe that 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 $\mathcal{L}_Q$. This does not lose generality, since if a system supports compatibility constraints in $\mathcal{L}_Q$, there is no reason for not supporting queries in the same $\mathcal{L}_Q$ for selecting items, and vice versa. We defer to future work the study in the setting when Q_c and Q are expressed in different languages. Queries in various query languages, e.g., CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\text{nr}}$, FO, DATALOG, are capable of expressing compatibility constraints commonly found in practice, including those studied in [2–6].

Aggregate constraints. Following [2], we define a cost function and a rating function over packages N: (1) cost(N) computes a real number in $\mathbb{R}$ as the cost of package N; and (2) val(N) computes a value in $\mathbb{R}$ as the overall rating of N. For instance, cost(N) in Example 1 is computed from the total time taken for visiting vista sites in N, which are open during the traveler's holiday, while val(N) is such defined that the higher the airfare and the total ticket prices for visiting vista sites in N, the lower the rating val(N) is.

To simplify the discussion, we assume that cost() and 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 cost(N) $\leq C$. For instance, the cost budget C in Example 1 is the total time allowed for visiting vista sites in 5 days, and cost(N) $\leq C$ imposes a bound on the number of vista sites that can be included in a travel package N.

Top-k package selections. For a database D, a query Q and compatibility constraints Q_c in $\mathcal{L}_Q$, a natural number $k \geq 1$, a cost budget C, and functions cost() and val(), a *top-k package selection* is a set $\mathcal{N} = \{N_i | i \in [1, k]\}$ of packages such that for each $i \in [1, k]$, the following conditions are satisfied:

(1) $N_i \subseteq Q(D)$, i.e., the items in N meet the criteria Q;
(2) $Q_c(N_i, D) = \varnothing$, i.e., the items in the package satisfy the compatibility constraints specified by query Q_c;
(3) cost(N_i) $\leq C$, i.e., its cost is below the budget;
(4) there exist two *predefined polynomials* p and p_s such that (a) the number $|N_i|$ of items in N_i is no larger than $p(|D|)$, where $|D|$ is the size of D, and (b) the arity of R_Q (i.e., the number of attributes in the tuples of N) does not exceed $p_s(|Q|, |\mathcal{R}|)$, where $|Q|$ is the size of query Q and $|\mathcal{R}|$ is the size of schema $\mathcal{R}$; indeed, it is not of much practical use to find items of exponentially large size or a package with exponentially many items;
(5) for all packages $N' \notin \mathcal{N}$ that satisfy conditions (1–4) given above, val(N') $\leq$ 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 $\text{cost}(N_i) \leq C$, $|N_i|$ does not exceed a predefined polynomial in $|D|$, and the arity of tuples in N_i is bounded by a predefined polynomial p_s in $|Q|$ and $|\mathcal{R}|$. As will be seen in Section 5, we shall also consider a constant size bound for $|N_i|$.

Given $D, Q, Q_c, k, C, \text{cost}()$ and $\text{val}()$, the *package recommendation problem* is to find a *top-k package selection* for $(Q, D, Q_c, \text{cost}(), \text{val}(), C)$, if there exists one. As we have seen in Example 1, users may want to find, e.g., a top-k travel-plan selection with the minimum price. The complexity of this problem has been settled in [10,11], and is not a topic of this work.

Item recommendation. To rank items, we use a *utility function* $f()$ to measure the usefulness of items selected by $Q(D)$ to a user [1]. 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' preferences [24], and may be *different for different users*.

Given a constant $k \geq 1$, a *top-k selection for* (Q, D, f) is a set $S = \{s_i | i \in [1, k]\}$ such that

(a) $S \subseteq Q(D)$, i.e., items in S satisfy the criteria given by Q;
(b) for all $s \in Q(D) \backslash S$ 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, where items are flights and the utility function $f()$ is defined in terms of the airfare and duration of each flight.

The item recommendation problem has also been studied in [10,11], and is not considered in this work.

The connection between item and package selections. Item selections are a special case of package selections. More specifically, a top-k selection $S = \{s_i | i \in [1, k]\}$ for (Q, D, f) is a top-k package selection $\mathcal{N}$ for $(Q, D, Q_c, \text{cost}(), \text{val}(), C)$, where $\mathcal{N} = \{N_i \mid i \in [1, k]\}$, such that for each $i \in [1, k]$,

(a) $N_i = \{s_i\}$;
(b) Q_c is a constant query that returns $\varnothing$ on any input, referred to as the empty query;
(c) $\text{cost}(N_i) = |N_i|$ if $N_i \neq \varnothing$, and $\text{cost}(\varnothing) = \infty$; that is, $\text{cost}(N_i)$ counts the number of items in N_i if $N_i \neq \varnothing$, and moreover, 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 $\text{cost}(N_i) \leq C$; and finally,
(e) $\text{val}(N_i) = f(s_i)$.

In the sequel, we refer to top-k selection for (Q, D, f) as top-k package selection for $(Q, D, Q_c, \text{cost}(), \text{val}(), C)$ instead, where $Q_c, \text{cost}(), \text{val}()$ and C are given 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., [25] 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 ($\exists\text{FO}^+$), built from atomic formulas by closing under $\wedge$, *disjunction* $\vee$ and $\exists$;

(d) nonrecursive datalog queries ($\text{DATALOG}_{\text{nr}}$), 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, and its dependency graph is acyclic; the *dependency graph* of a DATALOG query Q is a defined to be directed graph $G_Q = (V, E)$, where V includes all the predicates of Q, and (p', p) is an edge in E if and only if p' is a predicate that appears in a rule with p as its head [26];

(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 $\text{DATALOG}_{\text{nr}}$ with an inflational fixpoint operator.

3. Recommendation of query relaxations

In this section, we study query relaxation recommendation. As remarked earlier, in practice a query Q specifying selection criteria often finds no sensible packages to recommend. When this happens, the users naturally want the recommendation system to suggests how to revise their selection criteria by relaxing the query Q. This functionality should logically be part of recommendation systems. Unfortunately, we are not aware of any recommendation systems that support this functionality yet.

Below we first present query relaxations (Section 3.1). We then state the query relaxation recommendation problem, and establish its combined and data complexity (Section 3.2).

3.1. Query relaxations

Consider a query Q for specifying the *selection criteria*, in which a set Z 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 [20], we relax Q by replacing constants in E with variables, and replacing repeated variables in Z with distinct variables.

(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 $z \in Z$ that appears at least twice in atoms of Q, we introduce a new variable u_z and substitute u_z for one of the occurrences of z. This can be repeated until no variable has multiple occurrences. We use $\vec{u}$ to denote the tuple consisting of all such variables.

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.

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

To prevent relaxations that are too far from what users need, we constrain variables in $\vec{w}$ and $\vec{u}$ with certain ranges, by means of techniques developed for query relaxations [20,23] and preference queries [24]. To simplify the discussion, we assume that for each attribute A in a relation R, a distance function $\mathsf{dist}_{R.A}()$ is defined. Intuitively, given values a and b in $\mathsf{dom}(R.A)$, 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* (databases) can be generalized to *CS* (Computer Science) if dist(*DB*,*CS*) is small enough [20]. We denote by Γ the set of all distance functions defined on the domains of attributes in schema $\mathcal{R}$.

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

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

where Q' is obtained from Q by substituting w_c for constant c, and u_z for repeated occurrence of variable z. Here $\psi_w(\vec{w})$ is a conjunction of predicates of either the form (a) $\mathsf{dist}_{R.A}(w_c, c) \le d$, 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 $\mathsf{dist}_{R.A}(w_c, c) \le d$, and $\mathsf{gap}(\gamma) = 0$ if γ is $w_c = c$; similarly for a predicate in $\psi_u(\vec{u})$. Furthermore, we assume that the levels of relaxation of predicates in a relaxed query are additive; so *the level of relaxation of query* Q_Γ, denoted by $\mathsf{gap}(Q_\Gamma)$, is defined to be $\mathsf{sum}_{\gamma \in (\psi_w(\vec{w}) \cup \psi_u(\vec{u}))} \mathsf{gap}(\gamma)$.

Example 4. Recall the query Q defined on relations flight and vista in Example 1. Suppose that the query finds no items, as there is no direct flight from EDI to NYC. Furthermore, suppose that E has constants EDI, NYC, 1/1/2013 and $Z = \{x_{\mathsf{To}}\}$, 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 $Q$by making the selection criteria less restrictive as follows:

$$\begin{aligned} Q_1(&\mathsf{f\#}, \mathsf{Pr}, \mathsf{nm}, \mathsf{tp}, \mathsf{tkt}, \mathsf{tm}, \mathsf{da}) \\ &= \exists \mathsf{DT}, \mathsf{AT}, \mathsf{AD}, u_{\mathsf{To}}, w_{\mathsf{Edi}}, w_{\mathsf{NYC}}, w_{\mathsf{DD}} \\ &(\mathsf{flight}(\mathsf{f\#}, w_{\mathsf{Edi}}, x_{\mathsf{To}}, \mathsf{DT}, w_{\mathsf{DD}}, \mathsf{AT}, \mathsf{AD}, \mathsf{Pr}) \\ &\wedge\ x_{\mathsf{To}} = w_{\mathsf{NYC}} \wedge\ \mathsf{vista}(\mathsf{nm}, u_{\mathsf{To}}, \mathsf{tp}, \mathsf{tkt}, \mathsf{tm}, \mathsf{da}) \\ &\wedge\ w_{\mathsf{DD}} = 1/1/2013 \wedge\ \mathsf{dist}(w_{\mathsf{NYC}}, \mathsf{NYC}) \le 15 \\ &\wedge \mathsf{dist}(w_{\mathsf{Edi}}, \mathsf{EDI}) \le 15\ \wedge\ x_{\mathsf{To}} = u_{\mathsf{To}}). \end{aligned}$$

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

We may further relax Q_1 by allowing w_{DD} to be within 3 days of 1/1/2013, where the distance function for dates is $\mathsf{dist}_d()$:

$$\begin{aligned} Q_2(&\mathsf{f\#}, \mathsf{Pr}, \mathsf{nm}, \mathsf{tp}, \mathsf{tkt}, \mathsf{tm}, \mathsf{da}) \\ &= \exists \mathsf{DT}, \mathsf{AT}, \mathsf{AD}, u_{\mathsf{To}}, w_{\mathsf{Edi}}, w_{\mathsf{NYC}}, w_{\mathsf{DD}} \\ &(\mathsf{flight}(\mathsf{f\#}, w_{\mathsf{Edi}}, x_{\mathsf{To}}, \mathsf{DT}, w_{\mathsf{DD}}, \mathsf{AT}, \mathsf{AD}, \mathsf{Pr}) \\ &\wedge\ x_{\mathsf{To}} = w_{\mathsf{NYC}} \wedge\ \mathsf{vista}(\mathsf{nm}, u_{\mathsf{To}}, \mathsf{tp}, \mathsf{tkt}, \mathsf{tm}, \mathsf{da}) \\ &\wedge\ \mathsf{dist}(w_{\mathsf{Edi}}, \mathsf{EDI}) \le 15 \wedge\ \mathsf{dist}(w_{\mathsf{NYC}}, \mathsf{NYC}) \le 15 \\ &\wedge\ \mathsf{dist}_d(w_{\mathsf{DD}}, 1/1/2013) \le 3\ \wedge\ x_{\mathsf{To}} = u_{\mathsf{To}}). \end{aligned}$$

Query 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 between flight and vista to a Cartesian product. □

In this paper we consider simple query relaxation rules to illustrate the main idea of query relaxation recommendation. We defer a full treatment of query relaxation rules to future work.

3.2. Query relaxation recommendation

Consider a database D, queries Q and Q_c in $\mathcal{L}_Q$, functions cost() and val(), a cost budget C, a rating bound B, and a natural number $k \ge 1$. When there exists no top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$, we want to relax Q to find more packages for the users. More specifically, let Γ be a collection of distance functions, and Z 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 packages for $(Q_\Gamma, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ that are rated above B, i.e., for each $N \in \mathcal{N}$, we have that $N \subseteq Q_\Gamma(D)$, $Q_c(N, D) = \emptyset$, $\mathsf{cost}(N) \le C$, $\mathsf{val}(N) \ge B$, and moreover, $|N|$ is bounded by a predefined polynomial $p()$ in $|D|$.

We naturally want Q_Γ to *minimally differ* from the original Q. For a positive integer 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) $\mathsf{gap}(Q_\Gamma) \le g$, and (b) there exists a set $\mathcal{N}$ of k distinct packages for $(Q_\Gamma, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ that are rated above B.

We now state the query relaxation recommendation problem.

QRP($\mathcal{L}_Q$):	*Query relaxation recommendation problem*
INPUT:	A database D, a query $Q \in \mathcal{L}_Q$ with selected sets Z and E, a query $Q_c \in \mathcal{L}_Q$, two functions cost() and val(), natural numbers C, B, g and $k \ge 1$, and a collection Γ of distance functions, such that there exists no top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$.
QUESTION:	Does there exist a relaxation Q_Γ of Q for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$?

Combined complexity of QRP($\mathcal{L}_Q$). In the rest of the section we establish the combined and data complexity of QRP($\mathcal{L}_Q$). For the combined complexity, the query Q, compatibility constraint Q_c and database D may all vary. For the data complexity, only the database D varies, while Q and Q_c are predefined and fixed (see, e.g., [25] for

details). We study QRP($\mathcal{L}_Q$) for all the query languages $\mathcal{L}_Q$ given in Section 2.

We start with the combined complexity. No matter how important query relaxation recommendation is, QRP($\mathcal{L}_Q$) is nontrivial: it is Σ_2^p-complete when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$, PSPACE-complete when $\mathcal{L}_Q$ is DATALOG$_{\text{nr}}$ or FO, and EXPTIME-complete when $\mathcal{L}_Q$ is DATALOG. This tells us that query language $\mathcal{L}_Q$ dominates the combined complexity of QRP($\mathcal{L}_Q$). In addition, the complexity bounds are rather robust: all the lower bounds remain intact when $k=1$, i.e., they hold even for top-1 package recommendation.

Theorem 1. *The combined complexity for* QRP($\mathcal{L}_Q$) *is*

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

All the lower bounds remain unchanged when $k=1$. □

Before presenting the proof, we first sketch the general idea behind the lower bound proofs when the complexity of QRP($\mathcal{L}_Q$) coincides with the complexity of query evaluation of $\mathcal{L}_Q$, i.e., when $\mathcal{L}_Q$ is DATALOG$_{\text{nr}}$, FO, or DATALOG. We start by considering a selection query Q in $\mathcal{L}_Q$ that encodes the hardness of query evaluation, and add a predicate "$c=0$" to it. As will be seen shortly, the addition of this predicate ensures that Q does not select any packages. We then define distance functions Γ such that the only possible relaxation Q_Γ with gap$(Q_\Gamma)\le g$ is equal to the original query Q together with the relaxed predicate "$c=1$". The lower bounds then follow by showing that Q_Γ returns a package if and only if Q evaluates to non-empty. That is, in these cases the complexity of query evaluation determines the complexity of QRP. A similar trick is used when $\mathcal{L}_Q$ is CQ, UCQ and $\exists$FO$^+$. The higher complexity of QRP($\mathcal{L}_Q$) when compared to the query evaluation (which is in NP) is due to the presence of compatibility constraints and the fact that cost() and val() may involve aggregate computations.

We next prove Theorem 1 as follows.

Proof. We study QRP($\mathcal{L}_Q$) when $\mathcal{L}_Q$ ranges over CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\text{nr}}$, FO and DATALOG.

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

Lower bound. We show that QRP(CQ) is Σ_2^p-hard by reduction from the $\exists^*\forall^*$3DNF problem, which is Σ_2^p-complete [27]. The $\exists^*\forall^*$3DNF problem is to decide, given a sentence $\varphi=\exists X\forall Y\,\psi(X,Y)$, whether φ is true. Here $X=\{x_1,\ldots,x_m\}$, $Y=\{y_1,\ldots,y_n\}$, ψ is a disjunction $C_1\vee\cdots\vee C_r$, and C_i is a conjunction of three literals defined with variables in $X\cup Y$.

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 cost() and val(), and positive integers C, B and g. We show that φ is true if and only if there exists a query relaxation Q_Γ of Q for $(Q,D,Q_c,\text{cost}(),\text{val}(),C,B,k,g)$, when $k=1$.

(1) 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 $R_\neg(A,\bar{A})$.

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

Fig. 2. Relation instances used in the lower bound proof of Theorem 1.

Their instances are shown in Fig. 2. Intuitively, I_{01} encodes the Boolean domain, and $I_\vee$, $I_\wedge$ and $I_\neg$ encode disjunction, conjunction and negation, respectively. As will be seen shortly, ψ can be expressed in CQ in terms of these relations.

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

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

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

(3) We define a CQ query Q_c as compatibility constraint:

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

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. One can verify that Q_ψ can be expressed in CQ by leveraging relations $I_\vee$, $I_\wedge$ and $I_\neg$ given in Fig. 2. Intuitively, query $Q_c(b)$ returns a nonempty set if and only if 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 dist() defined on Boolean values: $\text{dist}(1,0)=\text{dist}(0,1)=1$, and $\text{dist}(0,0)=\text{dist}(1,1)=0$. In addition, we define $\text{cost}(N)=|N|$ if $N\neq\varnothing$, and $\text{cost}(\varnothing)=\infty$. These ensure that each valid package consists of a single tuple from the query answer. For $N=\{t\}$, where $t=(x_1,\ldots,x_m,c)$, we define $\text{val}(N)=1$ if $c=1$, and let $\text{val}(N)=-\infty$ otherwise. We also define $\text{val}(\varnothing)=-\infty$ and $\text{val}(N')=-\infty$ for any package N' consisting of more than one tuple.

One can easily verify that there exists no package $N\subseteq Q(D)$ such that $\text{val}(N)\ge B$ by the definitions of Q, val() and B.

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

($\Rightarrow$) First assume that φ is true. Then there must exist 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 as follows:

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

Then Q_Γ returns $(\mu_X^0,c=1)$ when Q_X generates μ_X^0, since in this case $w_c=1$ and $\text{dist}(w_c,0)\le 1$. 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, one can easily show that $\mathrm{gap}(Q_\Gamma) \le g$, $\mathrm{val}(N) \ge B$, $\mathrm{cost}(N) \le C$, and $N \subseteq Q_\Gamma(D)$. Therefore, Q_Γ is indeed a relaxation of Q for $(Q, D, Q_c, \mathrm{cost}(), \mathrm{val}(), C, B, k, g)$.

($\Leftarrow$) Conversely, assume that φ is false. Then for all truth assignments 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. Then 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. Furthermore, the empty package $N=\varnothing$ cannot be recommended because $\mathrm{cost}(\varnothing)=\infty > C$. As a result, there exists no relaxation of Q for $(Q, D, Q_c, \mathrm{cost}(), \mathrm{val}(), C, B, k, g)$.

Upper bound. We show that QRP($\exists$FO$^+$) is in Σ_2^p. It suffices to give 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, based on the predefined polynomial $p()$ (see Section 2), and (c) a tableau from D for each of these CQ queries (see [25] for tableau representation of CQ queries). These yield a set $\mathcal{N} = \{N_i \mid i \in [1,k]\}$ of packages such that $N_i \subseteq Q_\Gamma(D)$ for all $i \in [1,k]$.
2. For each $N_i \in \mathcal{N}$, check whether $Q_c(N_i, D) = \varnothing$. If so, continue; otherwise reject the guess and go back to step 1.
3. Check whether (a) $\mathrm{gap}(Q_\Gamma) \le g$. Moreover, for each $N_i \in \mathcal{N}$, whether (b) $\mathrm{cost}(N_i) \le C$ and (c) $\mathrm{val}(N_i) \ge B$. Furthermore, check whether $N_i \ne N_j$ for $i,j \in [1,k]$ and $i \ne j$. If all these conditions are satisfied, return "yes", and otherwise reject the guess and go back to step 1.

Observe that step 2 is in coNP, and step 3 is in PTIME. Thus the algorithm is in Σ_2^p as long as step (1) can effective construct a relaxed query of Q. Below we show that this is possible.

To see that a relaxed query can be effectively constructed, we use the following notion. Given a query Q, a database D 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 $Q_\Gamma(D) = Q'_\Gamma(D)$.

To check whether there exists a relaxation of Q for $(Q, D, Q_c, \mathrm{cost}(), \mathrm{val}(), C, B, k, g)$, it suffices to consider those relaxed queries that are not D-equivalent. Recall the definition of relaxed queries Q_Γ of Q from Section 3.1. Given a predicate of the form $\mathrm{dist}(w_c, c) \le d$ (resp. $\mathrm{dist}(u_z, z) \le d$), where the domain of w_c (resp. u_z) is $\mathrm{dom}(R.A)$, the bound d is constrained by the active domain of $R.A$. That is, $d \le l$, where l is the maximum distance between any two values in the active domain of $R.A$. In other words, for any $d > l$ and $d' > l$, we have that $\mathrm{dist}(w_c, c) \le d$ (resp. $\mathrm{dist}(u_z, z) \le d$) and $\mathrm{dist}(w_c, c) \le d'$ (resp. $\mathrm{dist}(u_z, z) \le d'$) are D-equivalent.

In light of this, to construct predicate $\mathrm{dist}(w_c, c) \le d$ (resp. $\mathrm{dist}(u_z, z) \le d$) in a relaxed query Q_Γ, we guess two values from the active domain of $R.A$, and let d be the difference between the two values. This allows us to nondeterministically inspect all relaxed queries Q_Γ of Q up to D-equivalence.

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

Lower bound. We first show that for DATALOG$_{\mathrm{nr}}$, QRP is PSPACE-hard by reduction from Q3SAT, which is PSPACE-complete (cf. [28]). Given a quantified sentence $\varphi = P_1 x_1 \ldots P_m x_m\ \psi(x_1, \ldots, x_m)$, Q3SAT is to decide whether φ is true, where P_i is either $\exists$ or $\forall$, and ψ is an instance of 3SAT, i.e., ψ is a formula $C_1 \wedge \cdot \wedge C_r$ in which each clause C_i is a disjunction of three variables or negations thereof taken from $\{x_1, \ldots, x_m\}$.

Given an instance φ of Q3SAT, we define a database D, a query Q with a set Z of variables and a set E of constants indicating parameters that are changeable, empty compatibility constraint Q_c, functions cost() and val(), the set Γ of distance functions, and positive integers C, B and g. We show that φ is true if and only if there exists a relaxation Q_Γ of Q for $(Q, D, Q_c, \mathrm{cost}(), \mathrm{val}(), C, B, k, g)$. In particular, we define $\mathrm{cost}(N) = |N|$ if $N \ne \varnothing$, $\mathrm{cost}(\varnothing) = \infty$ and set $C=1$. That is, only single tuples constitute packages. Moreover, we let $k=1$.

(1) Database D consists of a single relation, namely, I_{01} given in Fig. 2, which is specified by schema $R_{01}(X)$.

(2) We define query Q as follows:

$$Q(c) \text{:-} \ p(), \quad R_{01}(c), \quad c = 0,$$

where $p()$ is defined in stages as follows. Its body has an IDB:

$$p() \text{ :-} \ p_1(x_1, \ldots, x_m).$$

For each $i \in [1,m]$, $p_i(x)$ is defined as follows. If P_i is $\forall$, then

$$p_i(x_1, \ldots, x_m) \text{:-} \ p_{i+1}(x_1, x_{i-1}, 1, x_{i+1}, \ldots, x_m),$$
$$p_{i+1}(x_1, x_{i-1}, 0, x_{i+1}, \ldots, x_m),$$

i.e., it checks both $x_i=1$ and $x_i=0$. If P_i is $\exists$, then

$$p_i(x_1, \ldots, x_m) \text{ :-} \ R(x_i),\ p_{i+1}(x_1, \ldots, x_m),$$

i.e., either $x_i=1$ or $x_i=0$ will do. Finally, $p_{m+1}(x_1, \ldots, x_m)$ is an IDB that encodes $\psi(x_1, \ldots, x_m)$, by using inequality $\ne$ to encode the negation of variables and multiple datalog rules to encode disjunction. Obviously this is a nonrecursive datalog program. We let $E = \{0\}$, and $Z = \varnothing$ as in its CQ counterpart. That is, we only allow the Boolean value 0 to be relaxed.

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

Obviously, the answer to $p()$ in D is nonempty if and only if φ is true. Then along the same lines as the proof for the CQ case, one can verify that φ is true if and only if there exists a relaxation Q_Γ of Q for $(Q, D, Q_c, \mathrm{cost}(), \mathrm{val}(), C, B, k, g)$.

We next show that QRP is PSPACE-hard for FO by reduction from the membership problem for FO, i.e., the problem to determine, given an FO query Q, a database D and a tuple t, whether $t \in Q(D)$. This problem is known to be PSPACE-complete [29].

Given an instance (Q, D, t) of the membership problem for FO, we define query Q_1 in FO as follows:

$$Q_1(c) = \left(Q'(\vec{x}) \ \wedge \ R_{01}(c) \ \wedge \ c = 0\right),$$

where Q' is defined as

$$Q'(\vec{x}) = \left(Q(\vec{x}) \wedge \vec{x} = t\right).$$

Let $E = \{0\}$ and $Z = \emptyset$. Then by using the same D, empty compatibility constraint Q_c, functions cost(), val(), Γ, C, B and g defined above for DATALOG$_{\text{nr}}$, one can easily verify that $t \in Q(D)$ if and only if there exists a relaxation Q_Γ of Q_1 for $(Q_1, D, Q_c, \text{cost}(), \text{val}(), C, B, k, g)$.

Upper bound. We give an NPSPACE algorithm for determining QRP($\mathcal{L}_Q$) when $\mathcal{L}_Q$ is DATALOG$_{\text{nr}}$ 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 based on the predefined polynomial $p()$, 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 (a) $N_i \subseteq Q_\Gamma(D)$, (b) $\text{gap}(Q_\Gamma) \leq g$, (c) $\text{cost}(N_i) \leq C$, and (d) $\text{val}(N_i) \geq B$. If so, return "yes"; otherwise reject the guess and go to step 1.

As argued in the proof for QRP($\exists\text{FO}^+$) given above, step 1 can effectively guess relaxed queries. When $\mathcal{L}_Q$ is DATALOG$_{\text{nr}}$ or FO, steps 2 and 3 are in PSPACE [29]. Hence the algorithm is in NPSPACE=PSPACE; and so is QRP($\mathcal{L}_Q$) in this case.

When $\mathcal{L}_Q$ is DATALOG. Finally, we show that QRP(DATALOG) is EXPTIME-complete.

Lower bound. We show that QRP(DATALOG) is EXPTIME-hard by reduction from the membership problem for DATALOG. The latter 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 [29]. The reduction is the same as the reduction for the FO case given above, by defining Q_1 from Q, except that the query Q is in DATALOG rather than in FO. Along the same line as the proof of QRP(FO), one can prove that $t \in Q(D)$ if and only if there exists a relaxation Q_Γ of Q_1 for $(Q_1, D, Q_c, \text{cost}(), \text{val}(), C, B, k, g)$.

Upper bound. To prove the upper bound, we give an EXPTIME algorithm for deciding QRP(DATALOG) as follows.

1. Enumerate all relaxed queries of Q up to D-equivalence.
2. For each such relaxed query Q_Γ, if $\text{gap}(Q_\Gamma) \leq g$, then do the following.
   (a) Enumerate all subsets of $Q_\Gamma(D)$ with polynomially many tuples (using the predefined polynomial $p()$).
   (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) $\text{cost}(N_i) \leq C$; and (iii) whether $\text{val}(N_i) \geq B$. 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 next show that the algorithm is in EXPTIME. We first prove that step 1 is in EXPTIME. Indeed, following the same argument as given earlier for QRP($\exists\text{FO}^+$), 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 that are changeable parameters, 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 $\text{dist}(w_c, c) \leq d$ (resp. $\text{dist}(u_z, z) \leq d$), where the domain of w_c (resp. u_z) is 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 step 1 is in EXPTIME. Step 2 is iterated exponentially many times, and each iteration takes EXPTIME [29]. Hence step 2 is also in EXPTIME. Putting steps 1 and 2 together, we have that the algorithm is in EXPTIME.

This completes the proof of Theorem 1. Note that in all the lower bound proofs above, we assume $k = 1$. That is, the lower bounds remain intact when top-1 package is considered. □

Data complexity of QRP($\mathcal{L}_Q$). When it comes to the data complexity, we show that it makes our lives easier when selection criteria Q and compatibility constraints Q_c are both fixed. Moreover, different query languages have no impact on the data complexity: it is NP-complete for all query languages given in Section 2. Further, the lower bounds remain unchanged when $k = 1$, like their counterparts for the combined complexity.

Theorem 2. *The data complexity of* QRP($\mathcal{L}_Q$) *is* NP*-complete for all the languages given in Section 2, and remains* NP*-hard when* $k = 1$. □

Similar to the lower bound proofs in Theorem 1, we construct a query Q such that it has only one parameter (constant 0) that can be relaxed and such that the query returns no packages. We then ensure that the relaxed query Q_Γ returns packages as long as these exist. The existence of such packages is NP-hard due to the fact that cost() and val() may involve aggregate computations that, e.g., can ensure that packages correspond to valid truth assignments. We next give proof of Theorem 2 as follows.

Proof. It suffices to show that QRP(CQ) is NP-hard and that QRP($\mathcal{L}_Q$) is in NP for all query languages given in Section 2.

Lower bound. We show that QRP(CQ) is already NP-hard even in the absence of compatibility constraints Q_c. We verify this by reduction from 3SAT. An instance of 3SAT is a formula $\varphi = C_1 \wedge \cdots \wedge C_r$, where $X = \{x_1, \ldots, x_m\}$, as described in the proof of Theorem 1 for the FO case. It is to decide whether φ is satisfiable, i.e., there exists a truth assignment of variables in X that satisfies C_i for all $i \in [1, r]$.

Given an instance φ of 3SAT, we define a database D, a query Q in CQ, a set Γ of distance functions, empty Q_c, functions cost() and val(), and positive integers C, B, and g. We show that φ is satisfiable if and only if there exists a query relaxation Q_Γ of Q for $(Q, D, Q_c, \text{cost}(), \text{val}(), C, B, k, g)$, when $k = 1$.

(1) The database consists of a single relation R_C(cid, L_1, V_1, L_2, V_2, L_3, V_3, V). Its instance I_C consists of the following

tuples. Recall that for all $i \in [1, r]$, $C_i = \ell^i_1 \vee \ell^i_2 \vee \ell^i_3$, where the ℓ^i_j's are variables or negation of variables in X. For any possible truth assignment μ_i of variables in the literals in C_i that make C_i true, I_C includes a tuple $(i, x_k, v_k, x_l, v_l, x_m, v_m, 1)$, where $x_k = \ell^i_1$ if $\ell^i_1 \in X$, and $x_k = \overline{\ell}^i_1$ if $\ell^i_1 = \overline{x}_k$. We set $v_k = \mu_i(x_k)$; similarly for x_l, x_m and v_l and v_m.

(2) We define the query Q as follows:

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

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 relaxed. Observe that $Q(D) = \emptyset$ since D only carries tuples in which the V-attribute is equal to 1.

In addition, we define Q_c to be the empty query.

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

(4) We define $\mathsf{cost}(N) = 2$ if one of the following conditions is satisfied: (a) N contains two distinct tuples with the same cid value, (b) when N contains two tuples s and t that contain the same variable x_i but $s[V_i] = 0$ whereas $t[V_i] = 1$, (c) when not all variables in X appear in N, or (d) 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 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 if and only if there exists a query relaxation Q_Γ of Q for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$.

($\Rightarrow$) First assume that φ is satisfiable. Then there exists a truth assignment μ^0_X for X that satisfies φ, i.e., every clause C_j of φ is true with μ^0_X. Define a relaxed query Q_Γ of Q:

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

Then $Q_\Gamma(D) = I_C$ when $w_c = 1$, since $\mathsf{dist}(w_c, 0) \leq 1$ in this case. Given that μ^0_X 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^0_X. Let N consist of these r tuples. Then one can easily verify that $\mathsf{val}(N) = r \geq B$ and $\mathsf{cost}(N) = 1 \leq C$. Therefore, Q_Γ is a relaxation of Q for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, g)$.

($\Leftarrow$) Conversely, assume that φ is not satisfiable. Suppose by contradiction that there exists a relaxation Q_Γ of Q. Let N be a package of $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ that is rated above B. Then, this would imply that I_C contains r tuples, one for each clause of φ, such that put together they define a truth assignment μ_N for X that makes φ true. This contradicts the assumption that φ is not satisfiable.

Upper bound. To determine $\mathsf{QRP}(\mathcal{L}_Q)$ when Q and Q_c are fixed, we use the same algorithm given in the proof of Theorem 1 for QRP(FO). Since Q and Q_c are fixed, steps 2 and 3 of that algorithm are in PTIME, for all $\mathcal{L}_Q$ given in Section 2. Hence the algorithm is in NP, and so is $\mathsf{QRP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ ranges over CQ, UCQ, $\exists\mathrm{FO}^+$, $\mathrm{DATALOG}_{\mathrm{nr}}$, FO and DATALOG.

This completes the proof of Theorem 2. □

4. Adjustment recommendation

In this section we study adjustment recommendation. 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 managers 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 4.1). We then formulate and study the adjustment recommendation problem (Section 4.2).

4.1. Adjustments to item collections

Consider a database D of items that are currently provided by a system, and a collection D' of additional available 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 a query Q in $\mathcal{L}_Q$ for selecting items, a query Q_c in $\mathcal{L}_Q$ as compatibility constraint, functions cost() and 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 packages for $(Q, D \oplus \Delta(D, D'), Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ that are rated above B. That is, Q can find k packages in the adjusted database $D \oplus \Delta(D, D')$ such that for each such package N, $\mathsf{val}(N) \geq B$, and N satisfies 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 positive integer $k' \geq 1$, 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 packages for $(Q, D \oplus \Delta(D, D'), Q_c, \mathsf{cost}(), \mathsf{val}(), C)$ that are rated above B.

4.2. Deciding adjustment recommendation

These suggest that we study the following problem.

$\mathsf{ARP}(\mathcal{L}_Q)$:	*The adjustment recommendation problem*
INPUT:	Databases D and D', queries $Q, Q_c \in \mathcal{L}_Q$, two functions cost() and val(), natural numbers C, B and $k, k' \geq 1$, such that there exists no top-k package selection for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C)$.
QUESTION:	Is there a package adjustment $\Delta(D, D')$ for (Q, D, Q_c, cost(), val(), C, B, k, k')?

Combined complexity of $\mathsf{ARP}(\mathcal{L}_Q)$. The analysis of adjustment recommendation is no easier than its counterpart for query relaxation recommendation. Indeed, $\mathsf{ARP}(\mathcal{L}_Q)$ has the same combined and data complexity as $\mathsf{QRP}(\mathcal{L}_Q)$, although their proofs are quite different. Moreover, like their $\mathsf{QRP}(\mathcal{L}_Q)$ counterparts, all the lower bounds of $\mathsf{ARP}(\mathcal{L}_Q)$ also remain intact when $k = 1$, i.e., for top-1 package selection, when $\mathcal{L}_Q$ ranges over the query languages given in Section 2.

We first establish the combined complexity of $\mathsf{ARP}(\mathcal{L}_Q)$, when queries Q and Q_c and databases D and D' can all vary.

Theorem 3. *The combined complexity of* $\mathsf{ARP}(\mathcal{L}_Q)$ *is*

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

All the lower bounds remain unchanged when $k=1$. □

Below we outline the idea behind the lower-bound proofs when the complexity of $\mathsf{ARP}(\mathcal{L}_Q)$ coincides with the complexity of query evaluation of $\mathcal{L}_Q$, i.e., when $\mathcal{L}_Q$ is $\mathrm{DATALOG}_{\mathrm{nr}}$, FO, or DATALOG. We start by considering a selection query Q in $\mathcal{L}_Q$ that encodes the hardness of query evaluation and assume that D contains an empty special relation ensuring that Q does not return any packages. Adjustments to D that make that special relation non-empty will trigger a non-empty evaluation of Q. We then show that Q generates packages on the updated database if and only if Q evaluates to non-empty, from which the lower bounds then follow. Similarly, we verify the lower bound of $\mathsf{ARP}(\mathcal{L}_Q)$ when $\mathcal{L}_Q$ is CQ, UCQ or $\exists\mathrm{FO}^+$. For these languages, the higher complexity of $\mathsf{ARP}(\mathcal{L}_Q)$ when compared to the query evaluation (which is in NP) is due to the presence of compatibility constraints and the fact that cost() and val() may involve aggregate computations. Based on these ideas, we prove Theorem 3 as follows.

Proof. We verify the combined complexity of ARP for CQ, UCQ, $\exists\mathrm{FO}^+$, $\mathrm{DATALOG}_{\mathrm{nr}}$, FO and DATALOG.

When $\mathcal{L}_Q$ *is CQ, UCQ or* $\exists\mathrm{FO}^+$. It suffices to show that ARP is Σ_2^p-hard for CQ when $k=1$, and ARP is in Σ_2^p for $\exists\mathrm{FO}^+$.

Lower bound. We show that ARP(CQ) is Σ_2^p-hard by reduction from the $\exists^*\forall^*$3DNF problem (see the proof of Theorem 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 cost() and val(), and natural numbers k and k'. We show that φ is true if and only if there exists a package adjustment $\Delta(D,D')$ for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),B,C,k,k')$, when $k=1$. Assume $X=\{x_1,\ldots,x_m\}$ and $Y=\{y_1,\ldots,y_n\}$.

(1) Database D consists of four relations: (a) an empty relation $I_b=\varnothing$ of schema R_b which is the same as R_{01} given in the proof of Theorem 1, and (b) $I_\vee$, $I_\wedge$ and $I_\neg$ as shown in Fig. 2, which are specified by schemas $R_\vee(B,A_1,A_2)$, $R_\wedge(B,A_1,A_2)$ and $R_\neg(A,\overline{A})$ given in the proof of Theorem 1, respectively. We define D' to be the relation I_{01} given in Fig. 2, encoding the Boolean domain.

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

$$Q(\overrightarrow{x})=\exists z_1,z_0\,(Q_Z(z_1,z_0)\ \wedge\ (R_{01}(x_1)\wedge\cdots\wedge R_{01}(x_m))).$$

Here $\overrightarrow{x}=(x_1,\ldots,x_m)$, and sub-query $Q_Z(z_1,z_0)$ is defined as $R_b(z_1)\wedge(z_1=1)\ \wedge\ R_b(z_0)\wedge(z_0=0)$. As will be seen shortly, it ensures that the updated relation I_b encodes the Boolean domain I_{01}, and if so, $Q(\overrightarrow{x})$ generates all truth assignments of X variables by means of Cartesian products of R_{01}.

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

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

Here R_Q is the schema of the query answer $Q(D\oplus\Delta(D,D'))$, and $Q_Y(\overrightarrow{y})$ is to generate all truth assignments of Y variables by means of Cartesian products of the updated relation I_b, which is an instance of schema R_{01} and is the Boolean domain I_{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 answer $Q_c(b)$ is nonempty if and only if 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 ensure that any package N selected has exactly one item. We also define $k=1$ and $k'=2$.

One can easily see that $Q(D)=\varnothing$ since $I_b=\varnothing$. We next show that φ is true if and only if there exists a package adjustment $\Delta(D,D')$ for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C,B,k,k')$.

($\Rightarrow$) First assume that φ is true. Then there exists a truth assignment μ_X^0 for variables in 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, $\mathcal{N}=\{N\}$ makes a top-1 package recommendation. In other words, this $\Delta(D,D')$ is indeed a package adjustment for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C,B,k,k')$.

($\Leftarrow$) Conversely, assume that φ is false. Then for all truth assignments 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 packages N that satisfy $\mathsf{cost}(N)\le C$ and $\mathsf{val}(N)\ge B$, we have that $Q_c(N,D\oplus\Delta(D,D'))$ is nonempty. From these it follows that there exists no $\Delta(D,D')$ that is a package adjustment for $(Q,D,Q_c,\mathsf{cost}(),\mathsf{val}(),C,B,k,k')$.

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

1. Guess (a) a set $\Delta(D,D')$ consisting of at most k' updates, (b) k sets of CQ queries from Q, each of a polynomial cardinality based on the predefined polynomial $p()$, and (c) a tableau from $D\oplus\Delta(D,D')$ for each of these CQ queries. These yield a set $\mathcal{N}=\{N_i\mid i\in[1,k]\}$ of packages 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 $Q_c(N_i,D\oplus\Delta(D,D'))=\varnothing$. 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.

The algorithm is in Σ_2^p since step 2 is in coNP, and step 3 is in PTIME. From this it follows that $\mathsf{ARP}(\exists\mathrm{FO}^+)$ is in Σ_2^p.

When $\mathcal{L}_Q$ *is* $\mathrm{DATALOG}_{\mathrm{nr}}$ *or FO*. We next show that when $\mathcal{L}_Q$ is $\mathrm{DATALOG}_{\mathrm{nr}}$ or FO, $\mathsf{ARP}(\mathcal{L}_Q)$ is PSPACE-complete.

Lower bound. We show that ARP is PSPACE-hard for $\mathrm{DATALOG}_{\mathrm{nr}}$ by reduction from Q3SAT (recall the statement of Q3SAT from the proof for Theorem 1, for $\mathsf{QRP}(\mathrm{DATALOG}_{\mathrm{nr}})$).

Given an instance $\varphi = P_1 x_1 \ldots P_m x_m\ \psi(x_1, \ldots, x_m)$ of Q3SAT, we define a database D consisting of a single empty instance I_b of schema R_{01}, and $D' = I_{01}$. Moreover, we use the same query Q, empty query Q_c, function cost() as their counterparts given in the proof of Theorem 1 for QRP(DATALOG$_{nr}$). We define val() to be a constant function that returns 1 on any package, and let $C = B = k = 1$ and $k' = 2$. Along the same lines as the proof of Theorem 1 for DATALOG$_{nr}$, one can verify that φ is true if and only if there exists a package adjustment $\Delta(D, D')$ for (Q, D, Q_c, cost(), val(), C, B, k, k').

We next show that ARP is PSPACE-hard for FO by reduction from the membership problem for FO (see the proof of Theorem 1 for the statement of the membership problem). Using the same D, D', Q_c, C, cost(), val(), k, B and k' as defined above and the same query Q' as given in the proof of Theorem 1 for FO, one can encode an instance (Q, D, t) of the membership problem for FO. One can easily verify that $t \in Q(D)$ if and only if there exists a package adjustment for $(Q', D, Q_c, \text{cost}(), \text{val}(), C, B, k, k')$.

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

1. Guess a set $\Delta(D, D')$ with 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 in $\mathcal{N}$ has polynomially many items (based on the predefined polynomial $p()$) and $N_i \neq N_j$ when $i \neq j$.
2. For all $N_i \in \mathcal{N}$, check whether (a) $N_i \subseteq Q(D \oplus \Delta(D, D'))$, (b) $\text{cost}(N_i) \leq C$, and (c) $\text{val}(N_i) \geq B$. If so, continue, and otherwise reject the guess and go back to step 1.
3. For all $N_i \in \mathcal{N}$, check whether $Q_c(N_i, D \oplus \Delta(D, D')) = \varnothing$. If so, return "yes"; and otherwise reject the guess and go back to step 1.

The algorithm is in NPSPACE = PSPACE since steps 2 and 3 are in PSPACE; hence so is ARP when $\mathcal{L}_Q$ is FO or DATALOG$_{nr}$.

When $\mathcal{L}_Q$ is DATALOG. Finally, we show that ARP(DATALOG) is EXPTIME-complete.

Lower bound. We show that ARP(DATALOG) is EXPTIME-hard by reduction from the membership problem for DATALOG (recall the statement of the problem from the proof of Theorem 1 for QRP(DATALOG)). 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 Theorem 1 for QRP(DATALOG).

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

1. Compute all sets $\Delta(D, D')$ consisting of at most k' tuples taken 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, based on $p()$.
   (b) For each $\mathcal{N}$ consisting of k such pairwise distinct subsets, and for each set N_i in $\mathcal{N}$, check whether (a) $Q_c(N_i, D \oplus \Delta(D, D')) = \varnothing$, and (b) $\text{cost}(N_i) \leq C$; and (c) $\text{val}(N_i) \geq B$. 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 1 is in EXPTIME. Step 2 is also in EXPTIME: it is executed exponentially many times in total, and each iteration takes EXPTIME [29]. Hence the algorithm is in EXPTIME. Therefore, ARP(DATALOG) is also in EXPTIME.

This completes the proof of Theorem 3. □

Data complexity of ARP($\mathcal{L}_Q$). We next study the data complexity of ARP($\mathcal{L}_Q$), when only databases D and D' may vary, while Q and Q_c are fixed. The results below tell us that fixing queries Q and Q_c simplifies the analysis of ARP($\mathcal{L}_Q$), although ARP($\mathcal{L}_Q$) is still intractable. This is consistent with its counterpart for query relaxation recommendation (Theorem 2).

Theorem 4. *The data complexity of* ARP($\mathcal{L}_Q$) *is* NP*-complete for all the languages given in Section 2, and remains* NP*-hard when* $k = 1$. □

The lower-bound proofs follow an idea similar to their counterparts given for Theorem 3. More specifically, we construct a query Q such that Q evaluates to empty on D but returns packages on the updated $D \oplus \Delta(D, D')$. It is NP-hard to determine whether there exist packages selected by Q from $D \oplus \Delta(D, D')$ because cost() and val() may involve aggregate computation, which ensures that packages encode valid truth assignments. Based on this idea, we prove Theorem 4 as follows.

Proof. We show that when Q and Q_c are fixed, ARP($\mathcal{L}_Q$) is NP-complete for all the languages given in Section 2.

Lower bound. It suffices to show that ARP(CQ) is already NP-hard when $k = 1$, by reduction from 3SAT (recall the statement of 3SAT from the proof of Theorem 1).

Given an instance $\varphi = C_1 \wedge \cdots \wedge C_r$ of 3SAT defined over a set $X = \{x_1, \ldots, x_m\}$ of variables, we define a database D, queries Q and Q_c, a set D' of items, functions cost() and val(), and natural numbers C, B, $k \geq 1$ and $k' \geq 1$. We show that φ is satisfiable if and only if there exists a package adjustment $\Delta(D, D')$ for $(Q, D, Q_c, \text{cost}(), \text{val}(), C, B, k, k')$.

Before giving the reduction, we first construct an equivalent formula φ' from φ, such that φ is satisfiable if and only if φ' is satisfiable. Let z, e_1, e_2, e_3 be fresh variables that are not in X. We define $\varphi' = (\varphi \vee z) \wedge \overline{z} \wedge C_{r+1} = \bigwedge_{i=1}^{r} (C_i \vee z) \wedge \overline{z} \wedge C_{r+1}$, where $C_{r+1} = e_1 \vee e_2 \vee e_3$. It is easy to see that for all truth assignments μ_X of X variables, μ_X satisfies φ if and only if μ_X makes φ' true when extended with $z = 0$ and $e_j = 1$ for some $j \in [1, 3]$. Hence, φ is satisfiable if and only if φ' is satisfiable.

We next give the reduction. We set $k = k' = 1$.

(1) Database D contains a single relation $R_C(\text{cid}, L_1, V_1, L_2, V_2, L_3, V_3, Z, V_Z)$. Its instance I_C consists of the following tuples. Let $C'_i = \ell^i_1 \vee \ell^i_2 \vee \ell^i_3 \vee z$ be the ith clause of φ', where $i \in [1, r]$. For any possible truth assignment μ_i of variables in the literals of C'_i that satisfies C_i, we add a tuple $(i, x_k, v_k, x_l, v_l, x_m, v_m, z, 0)$ to I_C, such that $x_k = \ell^i_1$ when $\ell^i_1 \in X$, and $x_k = \overline{\ell}^i_1$ when $\ell^i_1 = \overline{x}_k$; similarly for x_l, v_l, x_m and v_m.

(2) We define D' to be the set consisting of a single tuple $(r+1, e_1, 1, e_2, 1, e_3, 1, z, 0)$, which encodes a truth assignment of variables in clause C_{r+1} and has $z=0$, satisfying $\overline{z} \wedge C_{r+1}$.

(3) We define the query Q to be the identity query on instances of R_C, and let Q_c be empty.

(4) We define $\text{cost}(N) = 2$ if one of the following conditions is satisfied: (a) package N contains two distinct tuples with the same cid value; (b) there exists two distinct tuples in N that have different values for a variable appearing in both of them, (c) not all variables in X appear in N, or (d) when N does not contain a tuple for every cid value. Furthermore, for any other N, we define $\text{cost}(N) = 1$. We set $C=1$.

(5) We define $\text{val}(N) = |N|$ if $N \neq \varnothing$ and $\text{val}(\varnothing) = -\infty$. Finally, we set the rating bound $B = r+1$.

One can easily verify that for any package N composed of tuples in $Q(D)$, $\text{val}(N) < B$, by the definition of function val(). Hence there exist no k packages satisfying the requirements.

We next show that this is indeed a reduction.

($\Rightarrow$) First assume that φ is satisfiable. Then there exists a truth assignment μ_X^0 for variables in X that satisfies φ. As argued earlier, μ_X^0 makes φ' true when we let $z=0$, and $e_j=1$ for $j \in [1,3]$. Let $\Delta(D, D') = D'$, i.e., $\Delta(D, D')$ consists of the single tuple $(r+1, e_1, 1, e_2, 1, e_3, 1, z, 0)$. Let N consist of r tuples in D that correspond to the truth assignment μ_X^0, one for each clause of φ', as well as the tuple in $\Delta(D, D')$. Then one can readily verify that $\text{cost}(N) \leq C$ and $\text{val}(N) \geq B$ by the definition of cost() and val(), respectively. That is, $\Delta(D, D')$ is a package adjustment for $(Q, D, Q_c, \text{cost}(), \text{val}(), C, B, k, k')$.

($\Leftarrow$) Conversely, assume that φ is not satisfiable. Recall that D' consists a single tuple $(r+1, e_1, 1, e_2, 1, e_3, 1, z, 0)$ and $k' = 1$. Then it is easy to see that there exists no package N consisting of tuples in $D \oplus \Delta(D, D')$ such that $\text{cost}(N) \leq C$ and $\text{val}(N) \geq B$, no matter how we define $\Delta(D, D')$. Indeed, it there exists such a package N, we can construct a truth assignment of X variables from N that makes φ true, by capitalizing on the definition of function cost() given above. Hence there exists no package adjustment for $(Q, D, Q_c, \text{cost}(), \text{val}(), C, B, k, k')$.

Upper bound. To see the upper bound, consider the algorithm developed in the proof of Theorem 3 for ARP(FO). This algorithm can also be used to check ARP($\mathcal{L}_Q$) when $\mathcal{L}_Q$ is any language given in Section 2. When Q and Q_c are fixed, both steps 2 and 3 of that algorithm are in PTIME. Hence the algorithm is in NP. Therefore, for fixed Q and Q_c in $\mathcal{L}_Q$, ARP($\mathcal{L}_Q$) is in NP when $\mathcal{L}_Q$ is CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\text{nr}}$, FO and DATALOG.

This completes the proof of Theorem 4. □

5. Special cases

The high complexity bounds of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) given in the previous sections suggest that we study special cases of these problems, to find out where the complexity comes from, and to identify tractable cases. In this section we study the cases when $\mathcal{L}_Q$ is a language for which the membership problem is in PTIME (Section 5.1), when packages are bounded by a constant instead of a polynomial (Section 5.2), and when compatibility constraints are simply PTIME functions or are absent (Section 5.3). We also study traditional item recommendations, for which each package has a single item, and compatibility constraints are absent (Section 5.4). We provide combined and data complexity of these cases, which demonstrate the impact of various factors on the analyses of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$).

5.1. SP queries

We have seen that QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) are Σ_2^p-hard when $\mathcal{L}_Q$ subsumes CQ. This motivates us to consider query languages simpler than CQ, for which the membership problem is in PTIME. To this end, we study SP, a fragment of CQ that supports projection and selection operators only. An SP query is of the form

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

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

We show that SP queries indeed simplify the combined complexity analyses of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$), to an extent: both problems become NP-complete as opposed to Σ_2^p-complete for CQ (Theorems 1 and 3), while their data complexity analyses remain NP-complete (Theorems 2 and 4). These are consistent with Theorems 1 and 3: query languages dominate the combined complexity of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$).

The study of SP is not only of theoretical interest. It is also useful in practice. Indeed, one often uses SP queries in recommendation systems, even *identity queries* [2], a special case of SP when $|\vec{y}| = 0$ and ψ is a tautology (see [25] for details). In fact the result below holds for all query languages with a PTIME membership problem, including but not limited to SP.

Corollary 1. *When $\mathcal{L}_Q$ is* SP, QRP($\mathcal{L}_Q$) *and* ARP($\mathcal{L}_Q$) *are* NP*-complete for both combined and data complexity.* □

Proof. We first study QRP(SP), and then investigate ARP(SP).

QRP*(SP)*. Recall that in the proof of Theorem 2, it is shown that QRP(CQ) is NP-hard by using a fixed SP query as Q and by letting Q_c be empty. Since the empty Q_c can be expressed in SP, the lower bound holds here, i.e., the proof of Theorem 2 shows that QRP(SP) is NP-hard even for data complexity.

For the upper bound, consider the algorithm given in the proof of Theorem 1 for the combined complexity of QRP(FO), which also works on SP queries. When Q and Q_c are in SP, steps 2 and 3 of the algorithm are both in PTIME. Thus the algorithm is in NP for SP. Putting these together, we have that QRP($\mathcal{L}_Q$) is NP-complete for combined and data complexity.

ARP*(SP)*. Recall the proof of Theorem 4, where it is shown that ARP(CQ) is NP-hard when Q is a fixed identity query and Q_c is empty. Thus ARP(SP) is already NP-hard for data complexity.

For the upper bound, the algorithm given in the proof of Theorem 3 for ARP(FO) works for SP. Observe that the

combined complexity of the algorithm is in NP since the membership problem for SP is in PTIME. Taken together, these show that ARP(SP) is NP-complete for combined and data complexity. □

5.2. Packages bounded by a constant size

One might be tempted to think that fixing package size would simplify the analyses of QRP and ARP. Below we study the impact of fixing package sizes, by considering packages N such that $|N| \le B_p$, where B_p is a predefined *constant* rather than a polynomial. This is practical since in real-life recommendation systems, one often assumes a constant bound on packages. Indeed, to form an NBA team, one needs 5 players only [3].

Below we show that fixing package sizes does not make our lives easier when the combined complexity analyses are concerned for both QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$), when $\mathcal{L}_Q$ ranges over all the languages considered here. In contrast, it indeed simplifies the data complexity analysis of QRP, but not that of ARP.

Observe the following. (1) In the results of Sections 3 and 4, QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) have the same combined and data complexity. However, given a constant bound B_p on packages, these problems behave quite differently: the data complexity analysis of QRP($\mathcal{L}_Q$) is tractable, while its ARP($\mathcal{L}_Q$) counterpart remains NP-complete. (2) The proofs of the intractability of the combined complexity analyses of QRP(SP) and ARP(SP) are quite different from their counterparts for QRP(CQ) and ARP(CQ) (Theorems 1–4). The reductions of the former make use of k, the number of packages, while the latter remain intact when $k=1$. That is, although SP is a fragment of CQ, the results for SP do not carry over to CQ when $k=1$.

Theorem 5. *For packages with a constant bound B_p, when $\mathcal{L}_Q$ is* CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$, FO *or* DATALOG,

- QRP($\mathcal{L}_Q$) *and* ARP($\mathcal{L}_Q$) *have the same combined complexity as their counterparts given in Theorems*1 *and*3, *respectively*;
- QRP($\mathcal{L}_Q$) *is in* PTIME *for data complexity*; *and*
- ARP($\mathcal{L}_Q$) *is* NP*-complete for data complexity.*

For SP,

- QRP *is* NP*-complete for combined complexity and is in PTIME for data complexity*; *and*
- ARP *is NP-complete for combined and data complexity.* □

Proof. We first study QRP, and then investigate ARP

(1) QRP($\mathcal{L}_Q$). We first investigate the combined complexity of QRP($\mathcal{L}_Q$), and then study its data complexity.

(1.1) *Combined complexity*.

(1.1.1) *When $\mathcal{L}_Q$ is CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$ or DATALOG*. Observe that the lower bounds of QRP given in Theorem 1 hold here since their proofs use only top-1 package consisting of one item. Moreover, all the algorithms given there still work on packages with a constant bound. Thus, for packages bounded by B_p, the combined complexity bounds of QRP($\mathcal{L}_Q$) remain unchanged for CQ, UCQ, FO, DATALOG$_{\mathsf{nr}}$ and DATALOG.

(1.1.2) *When $\mathcal{L}_Q$ is SP*. We show that QRP(SP) is NP-complete for packages with a constant bound B_p.

Lower bound. We show that QRP(SP) is NP-hard by reduction from the Knapsack problem. Given a finite set U, a size function $s()$ and a value function $v()$ such that for all $u \in U$, $s(u)$ and $v(u)$ are positive integers, and moreover, positive integers B_U and K_U, the Knapsack problem determines whether there exists a subset $U' \subseteq U$ such that $\mathsf{sum}_{u \in U'} s(u) \le B_U$ and $\mathsf{sum}_{u \in U'} v(u) \ge K_U$. It is known that this problem is NP-complete even when (a) $s(u) = v(u)$ for all $u \in U$, and (b) $K_U = B_U = (\mathsf{sum}_{u \in U} v(u))/2$ (cf. [30]). We consider this special case of Knapsack that is to determine whether there exists a subset $U' \subseteq U$ such that $\mathsf{sum}_{u \in U'} s(u) = (\mathsf{sum}_{u \in U} s(u))/2$.

Given such an instance of Knapsack, we define a database D, queries Q and Q_c in SP, a collection Γ of distance functions, functions cost() and val(), and natural numbers C, B and g. We show that there exists a subset U' of U satisfying $\mathsf{sum}_{u \in U'} s(u) \le B_U$ and $\mathsf{sum}_{u \in U'} s(u) \ge K_U$ if and only if there exists a relaxation Q_Γ of Q for (Q, D, Q_c, cost(), val(), C, B, k, g) when $B_p = 1$, i.e., for single-tuple packages.

(1) Database D consists of a single relation I_U specified by a $2m$-arity schema $R_U(U_1, P_1, \ldots, U_m, P_m)$, where $m = |U|$, and P_i has a Boolean domain with values 0 or 1. Assume that U consists of $u_1, \ldots, u_m$, sorted in an (arbitrary) order. Intuitively, attribute U_i in R_U corresponds to element u_i in U. The instance I_U consists of the following tuples. For each $u_i \in U$, there exist $s(u_i)$ many tuples in I_U, such that in each tuple t, $t(U_i)$ is a positive integer in $[1, s(u_i)]$ and $t(P_i) = 1$, and for all $j \in [1, m]$ with $j \ne i$, $t(U_j) = 0$ and $t(P_i) = 0$. Intuitively, we use these tuples to encode the size $s(u_i)$ of u_i.

(2) We define an SP query Q as follows.

$$Q(x_1, p_1, \ldots, x_m, p_m) = \big(R_U(x_1, p_1, \ldots, x_m, p_m) \wedge p_1 = 0 \wedge \cdots \wedge p_m = 0\big).$$

We let E, the set of changeable constants, be $\{0\}$, and we let the set Z of changeable variables be $\varnothing$. We define the compatibility constraint Q_c to be the empty query.

(3) We define $\mathsf{cost}(N) = \mathsf{val}(N) = |N|$ if $N \ne \varnothing$, and let $\mathsf{cost}(N) = +\infty$ and $\mathsf{val}(N) = 0$ otherwise. We set $C = 1$ such that any valid package N has a single tuple. We let $B = B_p = 1$.

(4) We let Γ consist of m distance functions $\mathsf{dist}_i()$ defined on P_i attributes: $\mathsf{dist}_i(1, 0) = s(u_i)/2$, and $\mathsf{dist}_i(0, 0) = 0$. We define $k = g = (\mathsf{sum}_{u \in U} s(u))/2$, i.e., $k = g = K_U = B_U$. Note that as opposed to the proofs given in the previous sections, the bound k on the number of packages is no longer constant 1.

One can easily verify that $Q(D) = \varnothing$ and hence, there exist no top-k packages $N \subseteq Q(D)$ that can be recommended.

We next show that this is indeed a reduction.

($\Rightarrow$) Assume that there exists a subset U' of U such that $\mathsf{sum}_{u \in U'} s(u) = K_U = B_U = (\mathsf{sum}_{u \in U} s(u))/2$. To simplify the discussion, assume *w.l.o.g.* that the subset U' consists of elements $u_1, \ldots, u_l$ for $l \in [1, m]$. Given this, we define the

relaxed SP query Q_Γ as follows:

$$Q_\Gamma(x_1, p_1, \ldots, x_m, k) = \exists\ w_1, \ldots, w_l \Big(R_U(x_1, w_1, \ldots, x_l, w_l, x_{l+1}, p_{l+1}, \ldots, x_m, p_m) \wedge \bigwedge_{i \in [1,l]} (p_i = w_i \wedge \mathsf{dist}(w_i, 0) \le s(u_i)/2) \wedge \bigwedge_{j \in [l+1,m]} p_j = 0 \Big).$$

Obviously, $\mathsf{gap}(Q_\Gamma) \le g = B_U$, and $Q_\Gamma(D)$ returns a set of K_U tuples t with $t[P_i] = 1$ for $i \in [1, l]$, corresponding to an element u_i in U'. That is, $p_i = 0$ in Q is "relaxed" in $Q_\Gamma(D)$. Let $\mathcal{N}$ consist of $k = K_U$ packages, such that each package N has a single tuple. Obviously, by the definition of Q_c, cost(), val(), B and C, $\mathcal{N}$ makes a top-k selection. Thus Q_Γ is indeed a relaxation for (Q, D, Q_c, cost(), val(), C, B, k, g).

($\Leftarrow$) Conversely, assume that there exists no subset U' of U such that $\mathsf{sum}_{u \in U'} s(u) = K_U = B_U = (\mathsf{sum}_{u \in U} s(u))/2$. Assume by contradiction that there exists a relaxation Q_Γ for (Q, D, Q_c, cost(), val(), C, B, k, g). Then there must exist a set $\mathcal{N}$ consisting of $k = K_U$ distinct packages N, such that $|N| = 1$, i.e., N consists of a single tuple $t \in Q_\Gamma(D)$, where $t[P_i] \neq 0$ for some $i \in [1, m]$ due to the relaxation of $p_i = 0$ in Q. Then we can readily derive U' from U based on $\mathcal{N}$, such that an element u_i is included in U' if and only if there exists a package $N \in \mathcal{N}$ such that N consists of a tuple t and $t[P_i] = 1$. By the definition of distance functions, the query Q and database D, one can easily verify that $\mathsf{sum}_{u \in U'} s(u) = \mathsf{sum}_{u \in U} s(u)$. This contradicts the assumption that such a subset U' does not exist.

Upper bound. We have shown that the combined complexity of QRP(SP) is NP-complete in Corollary 1. The upper bound obviously carries over here to packages with a constant bound.

(1.2) *Data complexity*. We show that when queries Q and Q_c are both fixed, QRP($\mathcal{L}_Q$) is in PTIME for packages with a constant bound, when $\mathcal{L}_Q$ is SP, CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$, FO or DATALOG. Indeed, for a fixed Q, there exist polynomially many relaxed queries up to D-equivalence, since $|E|$ and $|Z|$ are bounded (see the proof of Theorem 1 for the argument). Hence we can use the following algorithm.

1. Enumerate all relaxed queries of Q up to D-equivalence.
2. For each such relaxed query Q_Γ, if $\mathsf{gap}(Q_\Gamma) \le g$, then compute $Q_\Gamma(D)$ and do the following.
   (a) Enumerate all packages $N \subseteq Q_\Gamma(D)$ such that $|N| \le B_p$, $Q_c(N, D) = \varnothing$, $\mathsf{cost}(N) \le C$ and $\mathsf{val}(N) \ge B$. Let $\mathcal{N}$ consist of all such packages N.
   (b) Check if $|\mathcal{N}| \ge k$. If so, return "yes", and otherwise move to the next relaxed query.
3. Return "no" after all those relaxed queries of Q up to D-equivalence are checked, if none satisfies these conditions.

It is easy to verify the correctness of the algorithm. In particular, when $|\mathcal{N}| \ge k$ (step (3)), one can find top-k distinct packages from $\mathcal{N}$ such that the packages satisfy the selection criteria, compatibility constraints and aggregate constraints.

To see that the algorithm is in PTIME, observe the following. When Q is fixed, step 1 is in PTIME, as argued above. Step 2 is also in PTIME. Indeed, observe that relaxed queries have the form $Q_\Gamma(\vec{x}, \vec{w}, \vec{u}) = \exists \vec{w}, \vec{u}\ (Q'(\vec{x}, \vec{w}, \vec{u}) \wedge \psi_w(\vec{w}) \wedge \psi_u(\vec{u}))$. It takes PTIME to evaluate Q', ψ_w and ψ_u when Q is fixed. Moreover, step 2(a) is in PTIME since there are polynomially many packages N such that $|N| \le B_p$ when Q and B_p are both fixed; furthermore, for each such set N, it is in PTIME to check whether $Q_c(N, D) = \varnothing$ since Q_c is fixed. Putting these together, the algorithm is in PTIME for packages with a constant bound, when Q and Q_c are fixed queries in any of the query languages SP, CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$, FO or DATALOG.

(2) ARP($\mathcal{L}_Q$). We next settle the combined complexity and data complexity of ARP($\mathcal{L}_Q$).

(2.1) *Combined complexity*.

(2.1.1) *When $\mathcal{L}_Q$ is CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\mathsf{nr}}$, FO or DATALOG*. Observe that the lower bounds of ARP($\mathcal{L}_Q$) given in Theorem 3 remain intact here since their proofs use only top-1 packages consisting of a single item. In addition, all the algorithms developed there obviously also work on the special case when packages have a constant bound. Thus, for packages with a bounded size, the combined complexity bounds of ARP($\mathcal{L}_Q$) remain unchanged for these query languages.

(2.1.2) *When $\mathcal{L}_Q$ is SP*. We show that ARP(SP) is NP-complete.

Lower bound. We show that ARP(SP) is NP-hard, by reduction from 3SAT (see the proof of Theorem 1 for the statement of 3SAT). Given an instance $\varphi = C_1 \wedge \cdots \wedge C_r$ of 3SAT defined over a set $X = \{x_1, \ldots, x_m\}$ of variables, we define Q, D, D', Q_c, C, cost(), val(), k, B and k'. We then show that φ is satisfiable if and only if there exists an adjustment $\Delta(D, D')$ for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$. In contrast to the proofs of Theorems 3 and 4, the reduction here does not set $k = 1$.

We give the reduction as follows.

(1) Database D consists of a single relation $I_C = \varnothing$ specified by schema R_C(cid, L_1, V_1, L_2, V_2, L_3, V_3, V).

(2) We define D' to be the set consisting of the following tuples of R_C. Recall that for all $i \in [1, r]$, $C_i = \ell_1^i \vee \ell_2^i \vee \ell_3^i$, where ℓ_j^i's are variables or negation of variables in X. For any possible truth assignment μ_i of variables in the literals in C_i that makes C_i true, D' includes a tuple $(i, x_k, v_k, x_l, v_l, x_m, v_m, 1)$, where $x_k = \ell_1^i$ if $\ell_1^i \in X$, and $x_k = \overline{\ell}_1^i$ if $\ell_1^i = \overline{x}_k$. We set $v_k = \mu_i(x_k)$; similarly for x_l, x_m and v_l and v_m.

(3) We define the query Q as the identity query on instances of R_C, and let Q_c be the empty query.

(4) We define $\mathsf{cost}(N) = 2$ if either (a) N contains two distinct tuples with the same cid value; or (b) N contains two distinct tuples s and t that contain the same variable x_i with different values, i.e., $s[V_i] = 0$ while $t[V_i] = 1$. Furthermore, for any other N, we define $\mathsf{cost}(N) = 1$. We set the cost bound $C = 1$. In addition, we define $\mathsf{val}(N) = 1$ if $|N| = 2$, and for any other packages N, we let $\mathsf{val}(N) = 0$. We set the rating bound $B = 1$.

(5) We set $k = r \cdot (r-1)/2$ and $k' = r$. Note that neither k nor k' is set to 1, in contrast to the proofs in Sections 3 and 4.

(6) Finally, we define $B_p = 2$.

Given these, it is obvious that $Q(D)=\varnothing$ and hence, there exist no k packages that are rated above B.

We next show that this is indeed a reduction.

($\Rightarrow$) First assume that φ is satisfiable. Then there exists a truth assignment μ_X^0 for X that satisfies φ. Thus there exist r tuples t in D' that have the same truth assignments of X variables and are consistent with μ_X^0. Let $\Delta(D,D')$ consist of these r tuples. Note that $D\oplus\Delta(D,D')=\Delta(D,D')$ and Q is the identity query. Let $\mathcal{N}$ consist of $r\cdot(r-1)/2$ pairwise distinct packages such that each package N consists of two distinct tuples in $D\oplus\Delta(D,D')$. It is easy to see that for each package $N\in\mathcal{N}$, $\mathsf{cost}(N)\le C$ and $\mathsf{val}(N)\ge B$. Thus $\Delta(D,D')$ is an adjustment for $(Q, D, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$

($\Leftarrow$) Conversely, assume that φ is not satisfiable. Then there exists no truth assignment of X variables that makes φ true. Assume by contradiction that there exists an adjustment $\Delta(D,D')$ for $(Q, D, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$. Then there exists a set $\mathcal{N}$ consisting of $r\cdot(r-1)/2$ packages such that for each package N, $\mathsf{cost}(N)\le C$ and $\mathsf{val}(N)\ge B$. That is, each such package N consists of two tuples that have the same values for variables appearing in both of them. In other words, $\mathcal{N}$ encodes r clauses of φ in terms of r tuples in $\Delta(D,D')$, and each package in $\mathcal{N}$ checks whether any two of these clauses have the same truth values for variables appearing in both of them. Obviously, such r tuples encode a truth assignment of X variables that satisfies φ, which contradicts the assumption that φ is not satisfiable.

Upper bound. Consider the algorithm for $\mathsf{ARP}(\exists\mathrm{FO}^+)$ given in the proof of Theorem 3. The algorithm can be used to check $\mathsf{ARP}(\mathrm{SP})$ since $\exists\mathrm{FO}^+$ subsumes SP. We show that the algorithm is in NP for SP. Indeed, step 2 of that algorithm is in PTIME since Q_c is in SP and the combined complexity of the membership problem for SP is in PTIME. Thus the algorithm is in NP.

(2.2) *Data complexity*. We next study the data complexity of $\mathsf{ARP}(\mathcal{L}_Q)$. It suffices to show that ARP is NP-hard for fixed SP queries, and that it is in NP for fixed queries in CQ, UCQ, $\exists\mathrm{FO}^+$, $\mathrm{DATALOG}_{\mathrm{nr}}$, FO or DATALOG.

Observe that $\mathsf{ARP}(\mathrm{SP})$ is NP-hard for a fixed identity query Q, empty Q_c and packages consisting of two tuples, as we show in (2.1.2) above. Therefore, the lower bound holds here.

For the upper bound, we have shown that the data complexity of $\mathsf{ARP}(\mathcal{L}_Q)$ for all query languages in Section 2 is NP-complete in the proof of Theorem 4. This upper bound readily carries over here to packages with a constant bound.

Putting these together, we have that the data complexity analysis of $\mathsf{ARP}(\mathcal{L}_Q)$ is NP-complete when $\mathcal{L}_Q$ ranges over SP, CQ, UCQ, $\exists\mathrm{FO}^+$, FO, $\mathrm{DATALOG}_{\mathrm{nr}}$ and DATALOG.

This completes the proof of Theorem 5. □

Theorem 5 tells us that the combined complexity bounds of $\mathsf{QRP}(\mathcal{L}_Q)$ and $\mathsf{ARP}(\mathcal{L}_Q)$ are quite robust: they remain the same no matter whether packages have a constant size or not.

In light of these, we identify conditions in addition to constant package size, such that QRP and ARP are tractable. More specifically, we show the following. (1) When k' is fixed, i.e., when the number of tuples in $\Delta(D,D')$ is bounded by a constant, the data complexity analyses of $\mathsf{ARP}(\mathcal{L}_Q)$ are in PTIME for all the query languages considered in this paper. (2) When k is fixed, i.e., the number of packages is a constant, the combined complexity and data complexity analyses of $\mathsf{QRP}(\mathcal{L}_Q)$ and $\mathsf{ARP}(\mathcal{L}_Q)$ become tractable, when $\mathcal{L}_Q$ is SP.

Corollary 2. *For packages with a constant bound B_p,*

- *when k' is a constant, $\mathsf{ARP}(\mathcal{L}_Q)$ is in PTIME for data complexity, for all the languages of Section 2; and*
- *when k is a constant, the combined complexity and data complexity of* QRP *and* ARP *are in PTIME, for SP.* □

Proof. We first study the data complexity of ARP when k' is fixed for various $\mathcal{L}_Q$. We then investigate the combined and data complexity of QRP and ARP when $\mathcal{L}_Q$ is SP, for fixed k.

(1) *When k' is a constant*. We develop a PTIME algorithm for checking $\mathsf{ARP}(\mathcal{L}_Q)$ as follows, which works when $\mathcal{L}_Q$ ranges over all the query languages considered in this paper.

1. Enumerate all update $\Delta(D,D')$ with at most k' tuples in $D\cup D'$. Then do the followings.
   (a) For each such update $\Delta(D,D')$, check whether $\Delta(D,D')$ is an adjustment for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$. If so, return "yes".
   Return"no" after all such updates are inspected, if none
2. of them satisfies the condition above.

One can easily see that the algorithm is correct. We next show that the algorithm is in PTIME. Observe that step 1 is in PTIME when k' is a constant. Moreover, step 2 is also in PTIME when packages have a bounded size B_p, and Q and Q' are fixed queries. Indeed, one can enumerate all packages consisting of at most B_p tuples in $Q(D)$, and check whether there exist k packages N such that $Q_c(N,D)=\varnothing$, $\mathsf{cost}(N)\le C$ and $\mathsf{val}(N)\ge B$. Obviously, these can be done in PTIME when B_p and k' are constants, and when Q and Q_c are both fixed. As a result, the algorithm is in PTIME.

(2) *When k is a constant*. We next consider the setting when k is a constant, $\mathcal{L}_Q$ is SP, and packages have a constant bound. Below we first settle QRP(SP), and then investigate ARP(SP).

(2.1) QRP*(SP)*. It suffices to show that the combined complexity analysis of QRP(SP) is in PTIME in this setting. To do this, we provide a PTIME algorithm for checking QRP(SP).

Before we present the algorithm, let us first take a closer look at relaxations of SP queries. Observe that all relaxations of an SP query Q are defined by replacing some constants a in a predicate $x=a$ of Q with variable w_a. In light of this, an SP query $Q(\vec{x})=\exists\vec{y}\ (R(\vec{x},\vec{y})\wedge\varphi(\vec{x},\vec{y}))$ has a set E of constants indicating parameters that are allowed to change, where $\vec{x}=(x_1,\ldots,x_m)$ and $\vec{y}=(y_1,\ldots,y_n)$. As remarked in Section 3.1, for each constant $c\in E$, we associate a variable w_c with c, where w_c is a fresh variable in neither $\vec{x}$ nor $\vec{y}$. We denote by $\vec{w}$ the tuple consisting of all such variables w_c.

Assume a positive integer g as the bound on the gap between Q and its relaxations Q_Γ. Then for each variable w_c associated with $c \in E$, we define a range of w_c by setting $\mathsf{dist}_{R.A}(w_c, c) \le \min(l_c, g)$, where c is in $\mathsf{dom}(R.A)$ and l_c is the maximum distance between c and all other values in $\mathsf{dom}(R.A)$. Intuitively, $\mathsf{dist}_{R.A}(w_c, c) \le \min(l_c, g)$ characterizes the "maximum" range of w_c. We denote by $\psi_w(\vec{w})$ the conjunction of all such predicts $\mathsf{dist}_{R.A}(w_c, c) \le \min(l_c, g)$. Then we define $Q^0_\Gamma(\vec{x}) = \exists \vec{w}\ (Q'(\vec{x}, \vec{w}) \wedge \psi^0_w(\vec{w}))$. Intuitively, $Q^0_\Gamma(\vec{x})$ can be seen as the "maximum relaxed query" of Q that subsumes all valid relaxations of Q. Indeed, SP queries have the following "monotonicity": for any $c \in E$ and $l'_c < l_c$, and for any D, $Q_{\Gamma'}(D) \subseteq Q^0_\Gamma(D)$, where $Q_{\Gamma'}(\vec{x})$ uses $\mathsf{dist}_{R.A}(w_c, c) \le \min(l'_c, g)$ as a predicate instead of $\mathsf{dist}_{R.A}(w_c, c) \le \min(l_c, g)$.

Based on these, we develop the algorithm for checking QRP(SP) as follows. Assume that E consists of n constants $c_1, \ldots, c_n$, where each c_i is associated with a variable w_i.

1. Compute $Q^0_\Gamma(D)$.
2. Enumerate all packages N consisting of no more than B_p tuples from $Q^0_\Gamma(D)$. Then do the following.
3. For each such package N, compute $d_1, \ldots, d_n$, such that (i) there exists a query $Q_\Gamma(\vec{x}) = \exists \vec{w}\ (Q'(\vec{x}, \vec{w}) \wedge \psi_w(\vec{w}))$, where $\psi_w(\vec{w}) = \bigwedge_{i \in [1,n]} (\mathsf{dist}(w_i, c_i) \le d_i)$, and $d_1 + \cdots + d_n \le g$; (ii) $N \subseteq Q_\Gamma(D)$.
4. Checkwhether there exist k packages such that their Q_Γ can be merged into a single relaxed query Q'_Γ, where $\mathsf{gap}(Q'_\Gamma) \le g$, and $N \subseteq Q'_\Gamma(D)$ for each N of the k packages. If so, return "yes", and otherwise return "no".

One can easily verify that the algorithm is correct. We next show that it is in PTIME. Indeed, step 1 is in PTIME since Q is in SP. Step 2 is also in PTIME since B_p is a constant. In addition, step 3 is in PTIME. Indeed, for each package N enumerated in step 2, and for each tuple $t \in N$, one can find $d_1, \ldots, d_n$ determined by the value of t. There are at most B_p tuples in N, where B_p is a constant. Hence one can simply merge those $d_1, \ldots, d_n$ determined by the tuples in N, derive Q_Γ, and check the condition specified in step 3, in PTIME. Finally, step 4 is also in PTIME. Indeed, k is a constant, and hence there exist polynomially many sets consisting of k packages, where each of these packages has at most B_p tuples. Furthermore, checking the existence of a uniform relaxed query Q'_Γ from these k packages can be done in PTIME, along the same lines as step 3. Putting these together, the algorithm is in PTIME.

(2.2) ARP(SP). We next show that the combined complexity analysis of ARP(SP) is in PTIME, and as a consequence, so is its data complexity analysis. To do it, we first observe the following about item adjustments when $\mathcal{L}_Q$ is SP.

(1) To produce adjustments $\Delta(D, D')$ for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$, it suffices to only consider insertions of tuples from D' into D, since SP queries are monotonic. Indeed, given any SP query Q, database D and collection D' of items, assume that $\Delta(D, D') = \Delta_1 \cup \Delta_2$ is an adjustment for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$, where Δ_1 and Δ_2 consist of tuples removed from D and those inserted into D from D', respectively. Then Δ_2 must also be an adjustment for $(Q, D, Q_c, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$ because $Q(D \oplus \Delta(D, D')) \subseteq Q(D \cup \Delta_2)$ and better still, $|\Delta_2| \le k'$.

(2) For any SP query Q and databases D and D', $Q(D \cup D') = Q(D) \cup Q(D')$. Indeed, let $Q(\vec{x}) = \exists \vec{y}\ \left(R(\vec{x}, \vec{y}) \wedge \psi(\vec{x}, \vec{y})\right)$. Then obviously $Q(D) \cup Q(D') \subseteq Q(D \cup D')$ since SP queries are monotonic. Conversely, assume that t is a tuple in $Q(D \cup D')$. Then there must exist a tuple $t' \in D \cup D'$ such that t' satisfies $\psi(\vec{x}, \vec{y})$ since Q is an SP query. That is, $t \in Q(D)$ if $t' \in D$ and $t \in Q(D')$ if $t' \in D'$. Hence $t \in Q(D) \cup Q(D')$.

Based on these, we give the algorithm as follows.

1. Compute $Q(D \cup D')$.
2. Enumerate all packages N in $Q(D \cup D')$ such that $|N| \le B_p$, $Q_c(N, D) = \varnothing$, $\mathsf{cost}(N) \le C$, $\mathsf{val}(N) \ge B$. Denote by $\mathcal{N}$ the set consisting of all such packages.
3. Enumerate all sets $\mathcal{N}'$ consisting of k distinct packages in $\mathcal{N}$. Then do the following.
   (a) For each such set $\mathcal{N}'$ and for each package N in $\mathcal{N}'$, let the set $N_{D'}$ consist of all tuples t in $N \backslash Q(D)$.
   (b) Check whether $|\bigcup\{N_{D'} \mid N \in \mathcal{N}'\}| \le k'$. If so, return "yes", and otherwise move to the next set $\mathcal{N}'$.
4. Return "no" after all set $\mathcal{N}'$ enumerated in step 3 are inspected, if none of them satisfies the conditions above.

To see that the algorithm is correct, consider an SP query $Q(\vec{x})$ and a database D. Note that for each tuple s in D, there exists at most one tuple t in $Q(D)$ such that t is obtained from s by Q, since Q is an SP query. Based on these and the two properties given above, one can readily verify that the algorithm return "yes" if and only if there exists an adjustment $\Delta(D, D')$ for $(Q, D, \mathsf{cost}(), \mathsf{val}(), C, B, k, k')$.

We next show that the algorithm is in PTIME. Obviously, step 1 is in PTIME since Q is an SP query; and step 2 is in PTIME since all packages have a size bounded by B_p, and Q_c is in SP. Moreover, step 3 is in PTIME since k is a constant and there are polynomially many packages enumerated in step 2. Putting these together, we have that the algorithm is in PTIME.

This completes the proof of Corollary 2. □

5.3. PTIME compatibility constraints

One might think that high complexity is introduced by compatibility constraints Q_c and thus, wants to consider simpler Q_c. In light of this, below we study compatibility constraints that are just PTIME functions rather than queries in $\mathcal{L}_Q$.

In this setting, we show the following. (1) PTIME compatibility constraints indeed simplify the combined complexity analyses of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$. (2) When it comes to DATALOG$_{\mathsf{nr}}$, FO and DATALOG, however, PTIME compatibility constraints do not help. This is because when $\mathcal{L}_Q$ subsumes DATALOG$_{\mathsf{nr}}$ or FO, computing packages from $Q(D)$ is already costly, and checking Q_c does not increase the complexity bounds. (3) The same results hold even when

compatibility constraints are absent, i.e., when Q_c is empty. (4) When $\mathcal{L}_Q$ is SP, the absence of compatibility constraints does not simplify the analyses of QRP and ARP. This is because Q_c in SP can be checked in PTIME and hence, its presence or absence has no big impact on the complexity bounds.

Corollary 3. *When Q_c is in* PTIME *or absent*,

- *for* CQ, UCQ *and* $\exists$FO$^+$, QRP($\mathcal{L}_Q$) *and* ARP($\mathcal{L}_Q$) *become* NP-*complete* (*combined complexity*);
- *for* DATALOG$_{\text{nr}}$, FO *and* DATALOG, *the combined complexity bounds of* QRP($\mathcal{L}_Q$) *and* ARP($\mathcal{L}_Q$) *remain the same as given in Theorems*1 *and*3, *respectively*;
- *for data complexity*, QRP($\mathcal{L}_Q$) *and* ARP($\mathcal{L}_Q$) *are* NP-*complete for all the languages of Section* 2; *and*
- *when $\mathcal{L}_Q$ is* SP, QRP($\mathcal{L}_Q$) *and* ARP($\mathcal{L}_Q$) *are* NP-*complete for both combined and data complexity*. □

Proof. We first study QRP($\mathcal{L}_Q$), and then consider ARP($\mathcal{L}_Q$).

(1) QRP($\mathcal{L}_Q$). We investigate QRP($\mathcal{L}_Q$) for various $\mathcal{L}_Q$.

(1.1) *When $\mathcal{L}_Q$ is CQ, UCQ,* $\exists$FO$^+$, DATALOG$_{\text{nr}}$, *FO or DATALOG*. We first settle the combined complexity of QRP($\mathcal{L}_Q$), and then investigate its data complexity.

(1.1.1) *Combined complexity*. We show that when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$, QRP($\mathcal{L}_Q$) is NP-complete when Q_c is in PTIME or absent. It suffices to show that QRP(CQ) is NP-hard and QRP($\exists$FO$^+$) is in NP. To this end, observe the following. (1) When $\mathcal{L}_Q$ is CQ, we have shown in Theorem 2 that QRP($\mathcal{L}_Q$) is NP-hard even when Q is fixed and Q_c is empty. This tells us that QRP(CQ) is NP-hard when Q_c is absent. Furthermore, the same holds when Q_c is in PTIME, since empty Q_c is obviously in PTIME. Hence QRP(CQ) is NP-hard in this setting. (2) When $\mathcal{L}_Q$ is $\exists$FO$^+$, recall the algorithm for $\exists$FO$^+$ given in the proof of Theorem 1. When Q_c is PTIME computable, step 2 of the algorithm is in PTIME, and hence the algorithm is in NP. Putting (1) and (2) together, we have that QRP($\mathcal{L}_Q$) is NP-complete for CQ, UCQ or $\exists$FO$^+$, and when Q_c is in PTIME or absent.

When $\mathcal{L}_Q$ is DATALOG$_{\text{nr}}$, FO or DATALOG, observe the following. (1) The lower bound proofs of QRP($\mathcal{L}_Q$) given in the proof of Theorem 1 do not use Q_c. Therefore, those lower bounds remain intact here. (2) The algorithms of QRP($\mathcal{L}_Q$) given there carry over here. Hence the combined complexity of QRP for DATALOG$_{\text{nr}}$, FO and DATALOG remains the same no matter whether Q_c is in $\mathcal{L}_Q$ or PTIME, present or absent.

(1.1.2) *Data complexity*. As shown in the proof of Theorem 2, QRP(CQ) is NP-hard when Q is fixed and Q_c is empty. Thus the lower bound carries over here. In addition, the algorithm given there can be used here for PTIME Q_c, without increasing the complexity. Hence the data complexity of QRP($\mathcal{L}_Q$) remains NP-complete when $\mathcal{L}_Q$ ranges over CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\text{nr}}$, FO and DATALOG.

(1.2) *When $\mathcal{L}_Q$ is SP*. It suffices to show that QRP is NP-hard for fixed SP queries, and is in NP when SP queries vary. Recall that it has been shown in the proof of Theorem 2 that QRP(CQ) is NP-hard for a fixed SP query and empty Q_c. Hence the data complexity of QRP(SP) is NP-hard in the absence of Q_c. For the upper bound, consider the algorithm given in the proof of Theorem 1 for QRP(FO), which works for SP. The algorithm is in NP here since its step 2 is in PTIME when Q_c is in PTIME, and its step 3 is also in PTIME when Q is in SP.

(2) ARP($\mathcal{L}_Q$). We next study ARP($\mathcal{L}_Q$) for various $\mathcal{L}_Q$.

(2.1) *When $\mathcal{L}_Q$ is CQ, UCQ,* $\exists$FO$^+$, *FO,* DATALOG$_{\text{nr}}$ *or DATALOG*.

(2.1.1) *Combined complexity*. Observe the following. (1) ARP(CQ) is NP-hard even when Q is fixed and Q_c is absent, as shown in the proof of Theorem 4. (2) Consider the algorithm for $\exists$FO$^+$ given in the proof of Theorem 3. When Q_c is in PTIME or absent, step 2 of the algorithm is in PTIME, and the algorithm is in NP. Hence when Q_c is in PTIME or absent, ARP($\mathcal{L}_Q$) is NP-complete for $\mathcal{L}_Q$ ranging over CQ, UCQ and $\exists$FO$^+$.

When $\mathcal{L}_Q$ is DATALOG$_{\text{nr}}$, FO or DATALOG, observe that the lower bounds of ARP($\mathcal{L}_Q$) given in Theorem 3 are established by using an empty Q_c. Moreover, the algorithm given there obviously works when Q_c is in PTIME. Thus, when Q_c is in PTIME or absent, the combined complexity bounds of ARP($\mathcal{L}_Q$) remain unchanged for FO, DATALOG$_{\text{nr}}$ and DATALOG.

(2.1.2) *Data complexity*. Recall that as shown in the proof of Theorem 4, ARP(CQ) is already NP-hard when Q is fixed and Q_c is an empty query. Thus this lower bound holds here since when Q_c is an empty query, Q_c is in PTIME. In addition, the algorithm given there can be used here as well, and when Q_c is in PTIME, the algorithm retains the same complexity. Hence the data complexity of ARP($\mathcal{L}_Q$) is NP-complete for $\mathcal{L}_Q$ ranging over CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\text{nr}}$, FO and DATALOG.

(2.2) *When $\mathcal{L}_Q$ is SP*. We need to show that ARP is NP-hard for fixed SP queries, and is in NP when SP queries vary. Recall that in the proof of Theorem 4, QRP(CQ) is shown NP-hard for a fixed identity query Q and empty query Q_c. Hence the data complexity of ARP(SP) is NP-hard when Q_c is absent. For the upper bound, the algorithm given in the proof of Theorem 3 for QRP(FO) obviously works on SP queries. When Q is in SP and Q_c is in PTIME or absent, the algorithm is in NP since its steps 2 and 3 are both in PTIME. Putting these together, we have that ARP(SP) is NP-complete for both combined and data complexity, when Q_c is in PTIME or absent. □

5.4. Item recommendation

Item recommendation is supported by many real-life recommendation systems. Given a database D, a query $Q \in \mathcal{L}_Q$, a utility function $f()$ and a natural number $k \geq 1$, it is to find a top-k item selection for (Q, D, f), i.e., a set $S = \{s_i | i \in [1,k]\}$ such that (a) $S \subseteq Q(D)$, (b) for all $s \in Q(D) \backslash S$ and $i \in [1,k]$, $f(s) \leq f(s_i)$, and (c) $s_i \neq s_j$ if $i \neq j$. In this section, we revisit QRP and ARP in the context of item recommendation.

As remarked in Section 2, item recommendations are a special case of package recommendations when (a) compatibility constraints Q_c are *absent*, (b) each package consists of a single item, i.e., bounded by *a constant size* $B_p = 1$, and (c) the aggregate constraints defined in terms of cost() and val() have a simple specific form. One might

think that when these restrictions are imposed, the analyses of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) would be much simpler. Unfortunately, this is not the case. More specifically, we show the following for item recommendation: when $\mathcal{L}_Q$ is CQ, UCQ, $\exists$FO$^+$, DATALOG$_{nr}$, FO or DATALOG, (1) QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) have the same combined complexity bounds as their counterparts in the absence of Q_c (Corollary 3), i.e., further fixing package size does not help here; and (2) the data complexity bounds of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) remain the same as their counterparts when packages are bounded a constant size (Theorem 5). That is, further requiring the absence of Q_c does not make our lives easier. (3) In contrast to Theorem 5 and Corollary 3, when $\mathcal{L}_Q$ is SP, ARP becomes tractable for the combined and data complexity analyses, while the combined complexity of QRP remains NP-complete, the same as in the case when packages have a bounded size (Theorem 5).

Theorem 6. *For item recommendations, when* $\mathcal{L}_Q$ *is* CQ, UCQ, $\exists$FO$^+$, DATALOG$_{nr}$, FO *or* DATALOG,

- QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) *have the same combined complexity as their counterparts in the absence of* Q_c;
- QRP($\mathcal{L}_Q$) *is in* PTIME *for data complexity*; *and*
- ARP($\mathcal{L}_Q$) *is* NP*-complete for data complexity.*

When $\mathcal{L}_Q$ *is* SP,

- QRP *remains* NP*-complete for combined complexity*, *and is in* PTIME *for data complexity*; *and*
- ARP *is in* PTIME *for combined and data complexity*. □

Proof. When $\mathcal{L}_Q$ is CQ, UCQ, $\exists$FO$^+$, DATALOG$_{nr}$, FO, DATALOG or SP, we first study the combined and data complexity of QRP($\mathcal{L}_Q$), and then investigate their ARP($\mathcal{L}_Q$) counterparts.

(1) QRP($\mathcal{L}_Q$). We first settle the combined complexity of QRP, and then investigate its data complexity.

(1.1) *Combined complexity*. We show that for item recommendations, QRP($\mathcal{L}_Q$) is NP-complete when $\mathcal{L}_Q$ is SP, CQ, UCQ or $\exists$FO$^+$, PSPACE-complete when $\mathcal{L}_Q$ is FO or DATALOG$_{nr}$, and EXPTIME-complete when $\mathcal{L}_Q$ is DATALOG.

(1.1.1) *When* $\mathcal{L}_Q$ *is SP, CQ, UCQ or* $\exists$FO$^+$. It suffices to show that QRP for items is NP-hard for SP, and is in NP for $\exists$FO$^+$. Recall that in the lower bound proof of Theorem 5 for the combined complexity of QRP(SP), we use empty compatibility constraints and packages with a single item, as well as cost() and val() of the form for item recommendations. Hence QRP(SP) remains NP-hard for items. The matching upper bounds follow from Corollary 3, which tells us that QRP($\exists$FO$^+$) is in NP in the absence of compatibility constraints; as a result, QRP($\exists$FO$^+$) is in NP in the more restricted setting of item recommendations. Hence for items, the combined complexity of QRP($\mathcal{L}_Q$) is NP-complete when $\mathcal{L}_Q$ is SP, CQ, UCQ or $\exists$FO$^+$.

(1.1.2) *When* $\mathcal{L}_Q$ *is* DATALOG$_{nr}$, *FO or DATALOG*. Observe that in the proof of Theorem 1, the lower bounds of QRP($\mathcal{L}_Q$) for DATALOG$_{nr}$, FO and DATALOG are verified by using empty compatibility constraints and top-1 package with a single item, with cost() and val() of the specific forms for items. Hence those lower bounds carry over to QRP($\mathcal{L}_Q$) for item recommendations. Obviously, the upper bounds given there remain valid for items. Therefore, for item recommendation, QRP($\mathcal{L}_Q$) is PSPACE-complete when $\mathcal{L}_Q$ is either DATALOG$_{nr}$ or FO, and EXPTIME-complete when $\mathcal{L}_Q$ is DATALOG.

(1.2) *Data complexity*. Theorem 5 tells us that when packages are bounded by a constant size, the data complexity analysis of QRP($\mathcal{L}_Q$) is already tractable when $\mathcal{L}_Q$ is CQ, UCQ, $\exists$FO$^+$, FO, DATALOG$_{nr}$, DATALOG or SP. Hence QRP($\mathcal{L}_Q$) remains in PTIME in the more restricted setting of item recommendations, for fixed queries in these languages.

(2) ARP($\mathcal{L}_Q$). We next investigate ARP($\mathcal{L}_Q$).

(2.1) *Combined complexity*. We first study the combined complexity of ARP($\mathcal{L}_Q$), for various $\mathcal{L}_Q$.

(2.1.1) *When* $\mathcal{L}_Q$ *is CQ, UCQ or* $\exists$FO$^+$. Corollary 3 tells us that in the absence of compatibility constraints, ARP($\mathcal{L}_Q$) is already in NP when $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$. From this it follows that ARP($\mathcal{L}_Q$) remains in NP in the more restricted setting of item recommendations, for CQ, UCQ or $\exists$FO$^+$.

We show that ARP(CQ) is NP-hard for items. Note that the proofs of Theorems 3 and 4 (resp. Theorem 5) for the intractability of ARP(CQ) (resp. ARP(SP)) no longer work here, as those proofs assume that $k=1$ but packages have variable sizes (resp. Q is in SP but packages have a constant bound $B_p=2$, with aggregate constraints defined in terms of cost() and val()). In light of this, we now give a new proof for the lower bound of ARP(CQ) in the context of item recommendations.

We show that ARP(CQ) is NP-hard by reduction from 3SAT (see the proof of Theorem 1 for the statement of 3SAT). Given an instance $\varphi = C_1 \wedge \cdots \wedge C_r$ of 3SAT defined over a set $X = \{x_1, \ldots, x_m\}$ of variables, we define a database D, a collection D' of items, queries Q and Q_c, functions cost(), val(), and natural numbers C, B, k and k' that meet the requirements of item recommendations (see Section 2 for the requirements). We show that φ is satisfiable if and only if there exists an adjustment $\Delta(D, D')$ for $(Q, D, Q_c, \text{cost}(), \text{val}(), C, B, k, k')$.

(1) Database D consists of three relations: (a) $I_X = \varnothing$ specified by $R_X = (X, V)$; (b) I_ψ specified by schema $R_\psi = (\text{id}_C, P_x, X, V_x, w)$, where I_ψ encodes the clauses in ψ; for each $j \in [1, r]$, clause $C_j = \ell^j_1 \vee k^j \vee \ell^j_3$ 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 $l^j_1, ?l^j_2, ?l^j_3$, respectively, such that $w_i = 1$ if $v_i = 1$ and l^j_i is x_{l_i}, $w_i = 0$ if $v_i = 0$ and l^j_i is x_{l_i}, $w_i = 1$ if $v_i = 0$ and l^j_i is $\overline{x}_{l_i}$, and $w_i = 1$ if $v_i = 1$ and l^j_i is $\overline{x}_{l_i}$; and (c) relation $I_\vee$ given in Fig. 2. We define the set D' of items to be $\{(x_i, 0), (x_i, 1) | i \in [1, n]\}$, encoding truth values of X.

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

$$Q(j, c, x, v, x', v') = \exists j, x_1, x_2, x_3, v_1, v_2, v_3 \left(\bigwedge_{i=1}^{3} R_X(x_i, v_i) \wedge R_X(x, v) \wedge R_X(x', v') \wedge Q_\varphi(j, x_1, x_2, x_3, v_1, v_2, v_3, c) \right),$$

where

$$Q_\varphi(j, x_1, x_2, x_3, v_1, v_2, v_3, c) = \exists w_1, w_2, w_3 \, (R_\psi(j, 1, x_1, v_1, w_1) \wedge R_\psi(j, 2, x_2, v_2, w_2) \wedge R_\psi(j, 3, x_3, v_3, w_3) \wedge Q_\vee(w_1, w_2, w_3, c)).$$

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 val() given below.

(3) We define Q_c to be the empty query.

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

One can verify that this encoding is for top-k item selections (see Section 2 for how item selections are encoded as package selections). Furthermore, $Q(D) = \varnothing$ since $I_X = \varnothing$, i.e., there exist no k packages that satisfy the selection criteria.

We next show that this is indeed a reduction.

($\Rightarrow$) 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 val(), only tuples of the form $(j, 1, x, v, x, v)$ are valid choices for all $x \in X$. Let S be the set of all such tuples (items). Obviously, $|\Delta(D, D')| \leq k'$, $|S| = k$, and for each $t \in S$, $\text{val}(\{t\}) \geq B$ (note that val() is the rating function $f()$ for items). Hence there exists an adjustment for $(Q, D, Q_c, \text{cost}(), \text{val}(), C, B, k, k')$.

($\Leftarrow$) 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 val(), there exist no k distinct tuples in $Q(D \oplus \Delta(D, D'))$ whose val()-ratings reach B. Hence there exists no adjustment for $(Q, D, Q_c, \text{cost}(), \text{val}(), C, B, k, k')$.

(2.1.2) *When $\mathcal{L}_Q$ is* DATALOG$_{\text{nr}}$, *FO or DATALOG*. Recall that when $\mathcal{L}_Q$ is DATALOG$_{\text{nr}}$, FO or DATALOG, the lower bounds for ARP($\mathcal{L}_Q$) given in Theorem 3 are verified by using top-1 package consisting of one item, in the absence of compatibility constraints. Thus, these lower bounds are still valid for items. For the upper bound, note that the algorithms given there for ARP($\mathcal{L}_Q$) carry over to the setting of item recommendations.

(2.1.3) *When $\mathcal{L}_Q$ is SP*. We next show that for SP queries, ARP for items is in PTIME, by giving a PTIME algorithm. Contrast this with Theorem 5, which shows that ARP is NP-complete for SP when packages have a constant bound $B_p = 2$, even in the absence of compatibility constraints. ARP for items is tractable because when $B_p = 1$ and when cost() and val() have a specific simple form required by item recommendation, one can easily identify how tuples in $Q(D \oplus \Delta(D, D'))$ are propagated from $\Delta(D, D')$, as illustrated by the algorithm below.

Recall from the proof of Corollary 2 that we only need to consider insertions in $\Delta(D, D')$ since SP queries are monotonic, and moreover, $Q(D \cup D') = Q(D) \cup Q(D')$ for any SP query Q and databases D and D'. In light of this, we present the algorithm for ARP(SP) in the context of items, as follows.

The algorithm takes as input a database D, a collection D' of items, an SP query Q, a utility function $f()$ and natural numbers $k, k' \geq 1$. It checks whether there exists an adjustment $\Delta(D, D')$ to the item collection D subject to Q, $f()$ and k.

1. Compute $Q(D \cup D')$.
2. Denote by S the set consisting of all tuples t in $Q(D)$ such that $f(t) \geq B$, and by S' the set consisting of all tuples t in $Q(D')$ such that $f(t) \geq B$ and t is not in $Q(D)$. Thus $S \cup S'$ consists of all tuples t in $Q(D \cup D')$ such that $f(t) \geq B$, and moreover, $S \cap S'$ is empty.
3. Check whether $k' \geq k - |S|$. If so, return "yes", and otherwise return "no".

Obviously, the algorithm is in PTIME since it is in PTIME to compute $Q(D \cup D')$ in step 1, and is in PTIME to compute S and S' in step 2, because Q is an SP query and $f()$ is PTIME computable. We next show that the algorithm is correct.

We show that $k' \geq k - |S|$ if and only if there exists an adjustment for $(Q, D, Q_c, \text{cost}(), \text{val}(), C, B, k, k')$, where Q_c is empty, cost(), val(), C and B are defined in terms of $f()$ as discussed in Section 2. Observe first that for any tuple $t' \in D'$, t' corresponds to at most one tuple t in $Q(D')$ such that t is obtained from t' by Q, since Q is an SP query. Let $\Delta(D, D')$ consist of $k - |S|$ distinct tuples t' in D' such that these tuples yield tuples in S' via Q. Obviously, $|\Delta(D, D')| \leq k'$ if $k' \geq k - |S|$. Thus $\Delta(D, D')$ is an adjustment for $(Q, D, Q_c, \text{cost}(), \text{val}(), C, B, k, k')$. Conversely, assume that $k' < k - |S|$. Then for any $\Delta(D, D')$ consisting of no more than k' tuples from D', there are at most $k' + |S|$ tuples t in $Q(D \oplus \Delta(D, D'))$ such that $f(t) \geq B$. Thus there exists no adjustment for $(Q, D, Q_c, \text{cost}(), \text{val}(), C, B, k, k')$ when $k' + |S| < k$.

It should be remarked that the same argument does not apply to the setting when val() and cost() are present, and when packages are bounded by a constant $B_p = 2$. As indicated in the proof of Theorem 5, in that setting one cannot decide in PTIME how tuples from D' are propagated to k packages satisfying the aggregate constraints defined in terms of val() and cost(), unless P = NP or when k is a constant. As a result, ARP(SP) remains NP-complete in that setting.

(2.2) *Data complexity*. We show that the data complexity of ARP($\mathcal{L}_Q$) for items is NP-complete for CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\text{nr}}$, FO and DATALOG, and it is in PTIME for SP.

Observe that the lower bound proofs of ARP(CQ) for items given in (2.1.2) use a fixed CQ query defined over a fixed relational schema. Hence the lower bound remains intact for data complexity. The matching upper bounds follow from Theorem 4, in which ARP($\mathcal{L}_Q$) for packages, more general than ARP($\mathcal{L}_Q$) for items, is shown to be in NP. Hence the data complexity of ARP($\mathcal{L}_Q$) for items is NP-complete when $\mathcal{L}_Q$ is CQ, UCQ, $\exists$FO$^+$, DATALOG$_{\text{nr}}$, FO or DATALOG.

For SP queries, we have already verified in (2.1.3) above that the combined complexity analysis of ARP for items is in PTIME; hence so is its data complexity analysis.

This completes the proof of Theorem 6. □

6. Conclusion

We have identified and studied two recommendation problems beyond POI recommendations, namely, QRP($\mathcal{L}_Q$) for query relaxation recommendation, and ARP($\mathcal{L}_Q$) for adjustment recommendation. We have provided a complete picture of the lower and upper bounds of these problems, *all matching*, for both their combined complexity and data complexity, when $\mathcal{L}_Q$ ranges over a variety of queries languages. We have also investigated several special cases of these problems, when $\mathcal{L}_Q$ is a query language for which membership problem is in PTIME, when all packages are bounded by a constant B_p, when compatibility constraints Q_c are in PTIME or absent, and when both Q_c is absent and B_p is fixed to be 1 for item selections. These results tell us where complexity of these problem comes from.

Tables 1 and 2 (in Section 1) summarize the main complexity results and the complexity of special cases, annotated with their corresponding theorems. From the tables we can see the following. (1) The combined complexity bounds of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$) are rather robust. When $\mathcal{L}_Q$ is DATALOG$_{nr}$, FO or DATALOG, the bounds remain unchanged no matter whether packages have variable sizes or a fixed size, and whether Q_c is present or absent. When $\mathcal{L}_Q$ is CQ, UCQ or $\exists$FO$^+$, PTIME compatibility constraints Q_c make our lives easier, but fixing package size does not have any impact. When it comes to SP, ARP becomes tractable for item recommendations, whereas QRP remains NP-complete in the same setting. (2) The data complexity analysis of QRP($\mathcal{L}_Q$) becomes simpler when packages are bounded by a constant size B_p. In contrast, ARP($\mathcal{L}_Q$) becomes tractable only for SP under one of the following conditions: when items are recommended (Q_c is absent and packages have a fixed size), when the number k of packages is a constant, or when the bound k' on the number of tuples to be modified is a constant (the latter two are not shown in Table 2).

The study of recommendation problems beyond POI is still preliminary. First, we have only considered simple rules for query relaxations and adjustments, to focus on the main ideas. It is interesting, also from practical point of view, to investigate the impact of more sophisticated rules for relaxations and adjustments on the analysis of QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$), respectively. Second, 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). One may want to study different languages for Q and Q_c when needed in practice. Third, we have focused on generic functions cost(), val() and $f()$. These need to be fine-tuned by incorporating information about users, collaborative filtering and specific aggregate functions. Fourth, the recommendation problems are mostly intractable. An interesting topic is to identify more practical and tractable cases. Moreover, to make practical use of relaxation and adjustment recommendations, we need to develop heuristic algorithms (approximation whenever possible) for those intractable cases, possibly in a specific application domain. Finally, there are other recommendation issues beyond POI, QRP($\mathcal{L}_Q$) and ARP($\mathcal{L}_Q$), such as relaxation recommendation of compatibility constraints in addition to selection queries, and recommendation of adjustments to rating functions, among other things.

Acknowledgments

Deng is supported in part by 973 Program 2014CB340302 and 863 Program 2012AA011203, China. Fan is supported in part by 973 Program 2012CB316200, NSFC 61133002, Guangdong Innovative Research Team Program 2011D005 and Shenzhen Peacock Program 1105100030834361, China; he is also supported in part by EPSRC EP/J015377/1, UK and a Google Faculty Research Award, USA.

References

[1] G. Adomavicius, A. Tuzhilin, Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions, IEEE Trans. Knowl. Data Eng. 17 (6) (2005) 734–749.
[2] M. Xie, L. Lakshmanan, P. Wood, Composite recommendations: from items to packages, Front. Comput. Sci. 6 (3) (2012) 264–277.
[3] T. Lappas, K. Liu, E. Terzi, Finding a team of experts in social networks, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2009, pp. 467–476.
[4] G. Koutrika, B. Bercovitz, H. Garcia-Molina, FlexRecs: expressing and combining flexible recommendations, in: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD), 2009, pp. 745–758.
[5] A.G. Parameswaran, H. Garcia-Molina, J.D. Ullman, Evaluating, combining and generalizing recommendations with prerequisites, in: Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM), 2010, pp. 919–928.
[6] A.G. Parameswaran, P. Venetis, H. Garcia-Molina, Recommendation systems with complex constraints: a course recommendation perspective, ACM Trans. Inf. Syst. 29 (4).
[7] G. Adomavicius, A. Tuzhilin, Multidimensional recommender systems: a data warehousing approach, in: Electronic Commerce, Lecture Notes in Computer Science, vol. 2232, 2001, pp. 180–192.
[8] A. Brodsky, S. Morgan Henshaw, J. Whittle, Card: a decision-guidance framework and application for recommending composite alternatives, in: Proceedings of the 2008 ACM Conference on Recommender Systems (RecSys), 2008, pp. 171–178.
[9] S. Amer-Yahia, Recommendation projects at Yahoo! IEEE Data Eng. Bull. 34 (2) (2011) 69–77.
[10] T. Deng, W. Fan, F. Geerts, On the complexity of package recommendation problems, in: Proceedings of the 31st Symposium on Principles of Database Systems (PODS), 2012, pp. 261–272.
[11] T. Deng, W. Fan, F. Geerts, On the complexity of package recommendation problems, SIAM J. Comput. 42 (5) (2013) 1940–1986.
[12] A. Angel, S. Chaudhuri, G. Das, N. Koudas, Ranking objects based on relationships and fixed associations, in: Proceedings of the 12th International Conference on Extending Database Technology (EDBT), 2009, pp. 910–921.
[13] I.F. Ilyas, G. Beskales, M.A. Soliman, A survey of top-k query processing techniques in relational database systems, ACM Comput. Surv. 40 (4) (2008) 11:1–11:58.
[14] C. Li, M.A. Soliman, K.C.-C. Chang, I.F. Ilyas, Ranksql: supporting ranking queries in relational database management systems, in: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), 2005, pp. 1342–1345.
[15] R. Fagin, A. Lotem, M. Naor, Optimal aggregation algorithms for middleware, J. Comput. Syst. Sci. 66 (4) (2003) 614–656.

[16] K. Schnaitter, N. Polyzotis, Evaluating rank joins with optimal cost, in: Proceedings of the 17th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), 2008, pp. 43–52.
[17] T. Deng, W. Fan, On the complexity of query result diversification, in: Proceedings of the VLDB Endowment, vol. 6 (8), 2013, pp. 577–588.
[18] M. Drosou, E. Pitoura, Search result diversification, SIGMOD Rec. 39 (1) (2010) 41–47.
[19] S. Gollapudi, A. Sharma, An axiomatic approach for result diversification, in: Proceedings of the 18th International Conference on World Wide Web (WWW), 2009, pp. 381–390.
[20] S. Chaudhuri, Generalization and a framework for query modification, in: Proceedings of the 6th International Conference on Data Engineering (ICDE), 1990, pp. 138–145.
[21] T. Gaasterland, J. Lobo, Qualifying answers according to user needs and preferences, Fundam. Inf. 32 (2) (1997) 121–137.
[22] A. Kadlag, A. Wanjari, J. Freire, J. Haritsa, Supporting exploratory queries in databases, in: Proceedings of Database Systems for Advanced Applications (DASFAA), Lecture Notes in Computer Science, vol. 2973, 2004, pp. 594–605.
[23] N. Koudas, C. Li, A.K.H. Tung, R. Vernica, Relaxing join and selection queries, in: Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB), 2006, pp. 199–210.
[24] K. Stefanidis, G. Koutrika, E. Pitoura, A survey on representation, composition and application of preferences in database systems, ACM Trans. Database Syst. 36 (3).
[25] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, 1995.
[26] S. Chaudhuri, M.Y. Vardi, On the equivalence of recursive and nonrecursive datalog programs, J. Comput. Syst. Sci. 54 (1) (1997) 61–78.
[27] L.J. Stockmeyer, The polynomial-time hierarchy, Theor. Comput. Sci. 3 (1) (1976) 1–22.
[28] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.
[29] M.Y. Vardi, The complexity of relational query languages (extended abstract), in: Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC), 1982, pp. 137–146.
[30] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, 1979.