Department of Computer Science

ONTOLOGY-BASED QUERY ANSWERING WITH GROUP PREFERENCES

Thomas Lukasiewicz
Maria Vanina Martinez
Gerardo I. Simari
Oana Tifrea-Marciuska

CS-RR-14-02

Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD

DEPARTMENT OF COMPUTER SCIENCE RESEARCH REPORT

DEPARTMENT OF COMPUTER SCIENCE RESEARCH REPORT 14-02, MAY 2014

ONTOLOGY-BASED QUERY ANSWERING WITH GROUP PREFERENCES

(PRELIMINARY VERSION, 9 MAY 2014)

Thomas Lukasiewicz Maria Vanina Martinez
Gerardo I. Simari Oana Tifrea-Marciuska [1]

Abstract. The Web has recently been evolving into a system that is in many ways centered on social interactions and is now more and more becoming what is called the Social Semantic Web. One of the many implications of such an evolution is that the ranking of search results no longer depends solely on the structure of the interconnections among Web pages — instead, the social components must also come into play. In this paper, we argue that such rankings can be based on ontological background knowledge and on user preferences. Another aspect that has become increasingly important in recent times is that of uncertainty management, since uncertainty can arise due to many uncontrollable factors. To combine these two aspects, we propose extensions of the Datalog+/– family of ontology languages that both allow for the management of partially ordered preferences of groups of users as well as uncertainty, which is represented via a probabilistic model. We focus on answering k-rank queries in this context, presenting different strategies to compute group preferences as an aggregation of the preferences of a collection of single users. We also study merging operators that are useful for combining the preferences of the users with those induced by the values obtained from the probabilistic model. We then provide algorithms to answer k-rank queries for DAQs (disjunctions of atomic queries) under these group preferences and uncertainty that generalizes top-k queries based on the iterative computation of classical skyline answers. We show that such DAQ answering in Datalog+/– can be done in polynomial time in the data complexity, under certain reasonable conditions, as long as query answering can also be done in polynomial time (in the data complexity) in the underlying classical ontology. Finally, we present a prototype implementation of the query answering system, as well as experimental results (on the running time of our algorithms and the quality of their results) obtained from real-world ontological data and preference models, derived from information gathered from real users, showing in particular that our approach is feasible in practice.

[1]Department of Computer Science, University of Oxford, UK; e-mail: {thomas.lukasiewicz, vanina.martinez, gerardo.simari, oana.tifrea}@cs.ox.ac.uk.

Acknowledgements: This work was supported by the EPSRC grant EP/J008346/1 "PrOQAW: Probabilistic Ontological Query Answering on the Web", by the European Research Council (FP7/2007–2013/ERC) grant 246858 ("DIADEM"), by a Google European Doctoral Fellowship, and by a Yahoo! Research Fellowship. We are grateful to the reviewers of this article's preliminary abstract at WI 2013 [24] for their extensive comments, which have helped to improve this work. Finally, we wish to thank Giorgio Orsi for his help with the Datalog+/– query answering engine, and all the participants who volunteered to take part in our empirical evaluation.

Contents

1 Introduction

In the recent years, several important changes have been taking place on the classical Web. First, the so-called Web of Data is increasingly being realized as a special case of the Semantic Web. Second, as a part of the Social Web, users are acting more and more as first-class citizens in the creation and delivery of contents on the Web. The combination of these two technological waves is called the *Social Semantic Web* (or also *Web 3.0*), where the classical Web of interlinked documents is more and more turning into (i) semantic data and tags constrained by ontologies, and (ii) social data, such as connections, interactions, reviews, and tags. The Web is thus shifting away from data on linked Web pages towards fewer such interlinked data in social networks on the Web relative to underlying ontologies. This requires new technologies for search and query answering, where the ranking of search results is not solely based on the link structure between Web pages anymore, but on the information available in the Social Semantic Web — in particular, the underlying ontological knowledge present in user-created content, as well as the user's preferences implicitly or explicitly present in such content.

Modeling the preferences of a group of users is also an important research topic in its own right. With the growth of social media, people post their preferences and expect to get personalized information. Moreover, people use social networks as a tool to organize events, where it is required to combine the individual preferences and suggest items obtained from aggregated user preferences. For example, if there is a movie night of friends, family trip, or dinner with working colleagues, one has to decide which is the ideal movie or location for the group, given the preferences of each member. To address this problem, individual user preferences can be adopted and then aggregated to group preferences. However, this comes along with two additional challenges. The first challenge is to define a group preference semantics that solves the possible *disagreement* among users (a system should return results in such a way that each individual benefits from the result). For example, people (even friends) often have different tastes in restaurants. The second challenge is to allow for efficient algorithms, e.g., to compute efficiently the answers to queries under aggregated group preferences [2].

Example 1 To illustrate this, consider a situation in which three friends Alberto, Bruno, and Charles must decide where to go for dinner among three possible restaurants: r_1, r_2, and r_3. Alberto's preferences are $\langle r_1, \{r_2, r_3\}\rangle$ (meaning that he prefers r_1 and then he does not have a preference between r_2 and r_3), Bruno's are $\langle r_2, \{r_1, r_3\}\rangle$, and Charles's are $\langle r_1, r_2, r_3\rangle$. Finally, the three friends have consulted a review site that gives a score to each restaurant stating how likely it is that it will have seating available tonight; these probabilities are 0.2 for r_1, 0.9 for r_2, and 0.6 for r_3.

Taking a simple voting approach where the majority decides would lead the friends to choose r_1, since it is the first choice of two of them; however, it is also the least likely to have seating available — r_2, on the other hand, seems to be a better choice, since it is well-ranked by both Bruno and Charles, and has a high probability as well.

1.1 Related Work

Many studies address the area of group modeling. Indirectly, it is related to the area of social choice (group decision making, i.e., aiming at the decision that is best for a user given the opinion of individuals), which was studied in mathematics, economics, politics, and sociology [30, 34]. Other areas related to social choice are meta-search [25], collaborative filtering [19], and multi-agent systems [35]. Current approaches that deal with group preferences have also been studied in the area of recommender systems [2, 13, 29], which focus on quantitative preferences. However, in many real-world scenarios, the ordering of preferences is incomplete — cf. Example 1, where Alberto and Bruno both only declared a favorite restaurant and failed to fully order the rest. This appears due to privacy issues or an incomplete elicitation process (for instance, users may not want to be asked too many questions). Furthermore, it is often difficult to determine the appropriate numerical preferences and weights that maximize the utility of a decision [5]. For example, it is difficult for a user to determine a numerical value (i.e., 0.7 or 0.9) to rate a restaurant. These arguments thus support the use of qualitative over quantitative formalisms for representing and reasoning with incomplete preferences [31, 18, 1, 21].

In [21], we have introduced an extension of the Datalog+/– family of ontology languages with preferences for a single user; i.e., the addition of groups is novel in this line of research. Datalog+/– enables a modular rule-based style of knowledge representation — it has been studied in depth in the context of lightweight ontology languages, and has been shown to embed the entire *DL-Lite family*, the $\mathcal{E}L$ description logic, as well as *F-Logic Lite* [7]. We now focus on extending Datalog+/– with preferences of a group of users that comes with the two additional aforementioned challenges. This paper solves them by providing aggregated answers for DAQs (disjunctions of atomic queries) in polynomial time.

The presence of uncertainty in the Web in general is undeniable [17, 20, 28, 11]. Different sources of uncertainty that must be dealt with in answering queries in the Social Semantic Web are, e.g., information integration (as in travel sites that query multiple sources to find interesting tours), automatic processing of Web data (analyzing an HTML document often involves uncertainty), as well as inherently uncertain data such as the probability of a restaurant being crowded as discussed in Example 1.

1.2 Outline of this Work

The current challenge for Web search is therefore inherently linked to: (1) leveraging the social components of Web content towards the development of some form of semantic search and query answering on the Web as a whole, and (2) dealing with the presence of uncertainty in a principled way throughout the process. In this paper, we develop a novel integration of ontology languages with both preferences of groups of users and uncertainty management mechanisms. We do this by developing an extension of the Datalog+/– family of ontology languages [7] with a preference model over the consequences of the ontology, as well as a probabilistic model that assigns probabilities to them. The preference and the probabilistic model are assumed to model the preferences of a group of users and the uncertainty in the domain, respectively.

The main contributions of this paper can be summarized as follows.

- We introduce GPP-Datalog+/–, a language that combines the Datalog+/– [7] ontology language with group preferences (a generalization of preference handling in relational databases) and probabilistic uncertainty. To our knowledge, this is the first combination of ontology languages with group preferences (with and without probabilistic uncertainty). The preference and the probabilistic models are assumed to represent the preferences of a group of users and the uncertainty in the domain, respectively.

- We formalize the notion of k-rank query answering based on operators for merging single-user and probability-based preferences (in the form of a strict partial and a weak order, respectively), and aggregating multiple single-user preference relations. We analyze two approaches to computing an answer to a k-rank query that are suitable for partially ordered sets of preferences: collapse to single user (CSU), which constructs a single virtual user that aggregates the preferences of all the individuals from the group and the k-rank answers are computed over this new preference relation; and voting-based aggregation, where k-rankings are computed first for each individual user and then aggregation techniques based on voting strategies are used to aggregate the answers and obtain a single k-ranking.

- Based on an algorithm for the above preference merging and aggregation, we give algorithms for answering k-rank queries for DAQs (disjunctions of atomic queries), which generalize top-k queries based on the iterative computation of classical skyline answers. We show that answering DAQs in GPP-Datalog+/– is possible in polynomial time in the data complexity modulo the cost of computing probabilities.
- Finally, we developed a prototype implementation of a group preference-based query answering system, and conducted a series of experiments based on real-world ontological data and preference models derived from information gathered from real users. The results (on the running time of our algorithms and the quality of their results) show in particular that the strategies proposed and developed in this work are feasible in practice.

The rest of this paper is organized as follows. In Section 2, we recall some basics on Datalog+/– and PrefDatalog+/–. Section 3 introduces the syntax and the semantics of GPP-Datalog+/–, in particular, the general group preference model and the probabilistic model, along with preference merging and aggregation operations. In Section 4, we present algorithms for k-rank query answering, along with correctness and data tractability results. Section 5 studies a set of properties that can be considered desirable for the aggregation of group preferences. The experimental setup and results are presented in Section 6. Finally, Section 7 summarizes the main results of this paper and gives an outlook on future research. Detailed proofs of all results in this paper are given in the appendix.

2 Preliminaries

In this section, we briefly recall some necessary background concepts on Datalog+/– and its generalization by preferences.

2.1 Datalog+/–

We first recall some basics on Datalog+/– [7], namely, on relational databases, (Boolean) conjunctive queries ((B)CQs), tuple- and equality-generating dependencies (TGDs and EGDs, respectively), negative constraints, the chase, and ontologies in Datalog+/–.

Databases and Queries We assume (i) an infinite universe of *(data) constants* Δ (which constitute the "normal" domain of a database), (ii) an infinite set of *(labeled) nulls* Δ_N (used as "fresh" Skolem terms, which are placeholders for unknown values, and can thus be seen as variables), and (iii) an infinite set of variables $\mathcal{V}$ (used in queries, dependencies, and constraints). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. We denote by $\mathbf{X}$ sequences of variables $X_1, \ldots, X_k$ with $k \geqslant 0$. We assume a *relational schema* $\mathcal{R}$, which is a finite set of *predicate symbols* (or simply *predicates*). A *term* t is a constant, null, or variable. An *atomic formula* (or *atom*) A has the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate, and $t_1, \ldots, t_n$ are terms. It is *ground* iff every t_i belongs to Δ.

A *database (instance)* D for a relational schema $\mathcal{R}$ is a (possibly infinite) set of atoms with predicates from $\mathcal{R}$ and arguments from $\Delta \cup \Delta_N$. A *conjunctive query (CQ)* over $\mathcal{R}$ has the form $Q(\mathbf{X}) = \exists \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms (possibly equalities, but not inequalities) with the variables $\mathbf{X}$

and $\mathbf{Y}$, and possibly constants, but without nulls. A *Boolean CQ (BCQ)* over $\mathcal{R}$ is a CQ of the form $Q()$, often written as the set of all its atoms, without quantifiers. Answers to CQs and BCQs are defined via *homomorphisms*, which are mappings $\mu\colon \Delta \cup \Delta_N \cup \mathcal{V} \to \Delta \cup \Delta_N \cup \mathcal{V}$ such that (i) $c \in \Delta$ implies $\mu(c) = c$, (ii) $c \in \Delta_N$ implies $\mu(c) \in \Delta \cup \Delta_N$, and (iii) μ is naturally extended to atoms, sets of atoms, and conjunctions of atoms. The set of all *answers* to a CQ $Q(\mathbf{X}) = \exists \mathbf{Y}\, \Phi(\mathbf{X},\mathbf{Y})$ over a database D, denoted $Q(D)$, is the set of all tuples $\mathbf{t}$ over Δ for which there exists a homomorphism $\mu\colon \mathbf{X} \cup \mathbf{Y} \to \Delta \cup \Delta_N$ such that $\mu(\Phi(\mathbf{X},\mathbf{Y})) \subseteq D$ and $\mu(\mathbf{X}) = \mathbf{t}$. The *answer* to a BCQ $Q()$ over a database D is *Yes*, denoted $D \models Q$, iff $Q(D) \neq \emptyset$.

Given a relational schema $\mathcal{R}$, a *tuple-generating dependency (TGD)* σ is a first-order formula of the form $\forall\mathbf{X}\forall\mathbf{Y}\, \Phi(\mathbf{X},\mathbf{Y}) \to \exists\mathbf{Z}\, \Psi(\mathbf{X},\mathbf{Z})$, where $\Phi(\mathbf{X},\mathbf{Y})$ and $\Psi(\mathbf{X},\mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$ (without nulls), called the *body* and the *head* of σ, denoted $body(\sigma)$ and $head(\sigma)$, respectively. Such σ is satisfied in a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h that maps the atoms of $\Phi(\mathbf{X},\mathbf{Y})$ to atoms of D, there exists an extension h' of h that maps the atoms of $\Psi(\mathbf{X},\mathbf{Z})$ to atoms of D. Since TGDs can be reduced to TGDs with only single atoms in their heads, in the sequel, every TGD has w.l.o.g. a single atom in its head.

Query answering under TGDs, i.e., the evaluation of CQs and BCQs on databases under a set of TGDs is defined as follows. For a database D for $\mathcal{R}$, and a set of TGDs Σ on $\mathcal{R}$, the set of *models* of D and Σ, denoted $mods(D,\Sigma)$, is the set of all (possibly infinite) databases B such that (i) $D \subseteq B$ and (ii) every $\sigma \in \Sigma$ is satisfied in B. The set of *answers* for a CQ Q to D and Σ, denoted $ans(Q,D,\Sigma)$, is the set of all tuples $\mathbf{a}$ such that $\mathbf{a} \in Q(B)$ for all $B \in mods(D,\Sigma)$. The *answer* for a BCQ Q to D and Σ is *Yes*, denoted $D \cup \Sigma \models Q$, iff $ans(Q,D,\Sigma) \neq \emptyset$. Note that query answering under general TGDs is undecidable [3], even when the schema and TGDs are fixed [6]. There are sets of TGDs for which answering BCQs is decidable and can even be done in polynomial time in the data complexity, for example, sets of linear, multi-linear, guarded, sticky, or sticky-join TGDs [7, 8].

Negative constraints (or simply *constraints*) γ are first-order formulas of the form $\forall\mathbf{X}\Phi(\mathbf{X}) \to \bot$, where $\Phi(\mathbf{X})$ (called the *body* of γ) is a conjunction of atoms (without nulls). Under the standard semantics of query answering of BCQs in Datalog+/– with TGDs, adding negative constraints is computationally easy, as for each constraint $\forall\mathbf{X}\Phi(\mathbf{X}) \to \bot$, we only have to check that the BCQ $\Phi(\mathbf{X})$ evaluates to false in D under Σ; if one of these checks fails, then the answer to the original BCQ Q is true, otherwise the constraints can simply be ignored when answering the BCQ Q. As another component, the Datalog+/– language allows for special types of *equality-generating dependencies (EGDs)*. Since they can also be modeled via negative constraints, we omit them here, and we refer to [7] for their details.

The chase algorithm for a database D and a set of TGDs Σ consists of an exhaustive application of the TGDs (cf. Appendix A). The (possibly infinite) chase relative to TGDs is a *universal model*, i.e., there exists a homomorphism from $chase(D,\Sigma)$ onto every $B \in mods(D,\Sigma)$ [7]. This implies that BCQs Q over D and Σ can be evaluated on the chase for D and Σ, i.e., $D \cup \Sigma \models Q$ is equivalent to $chase(D,\Sigma) \models Q$. For tractable fragments, such BCQs Q can be evaluated in polynomial time in the data complexity.

Datalog+/– Ontologies A *Datalog+/– ontology* $O = (D,\Sigma)$, where $\Sigma = \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}$, consists of a finite database D over Δ, a finite set of TGDs Σ_T, a finite set of non-conflicting EGDs Σ_E, and a finite set of negative constraints Σ_{NC}. In the rest of this paper, we assume that all ontologies belong to one of the data-tractable fragments of Datalog+/–.

Example 2 A simple Datalog+/– ontology $O = (D,\Sigma)$ for food is given below. Intuitively, the database D encodes that (i) *sushi*, *pizza*, *pasta*, *salad*, and *fish* are types of food, (ii) p_1 and p_2 are places, where p_1 is a *pizzeria*, serving *Italian cuisine*, while p_2 is a *sushi bar*, serving *Japanese cuisine*, and located in the city c_1, and (iii) *Italian cuisine* includes *pizza* and *pasta*, *Japanese cuisine* includes *sushi* and *fish*, and *vegetarian cuisine* includes *pizza* and *salad*. The set of constraints Σ encodes that every place is located in a city, and that every place that serves a type of food of a particular cuisine also serves that cuisine.

$$\begin{aligned}
D \;=\; &\{\mathit{food}(\mathit{sushi}),\ \mathit{food}(\mathit{pizza}),\ \mathit{food}(\mathit{pasta}),\ \mathit{food}(\mathit{salad}),\ \mathit{food}(\mathit{fish}),\ \mathit{place}(p_1),\ \mathit{place}(p_2),\\
&\mathit{type}(p_1,\mathit{pizzeria}),\ \mathit{serves}(p_1,\mathit{Italian}),\ \mathit{type}(p_2,\mathit{sushi\_bar}),\ \mathit{serves}(p_2,\mathit{Japanese}),\\
&\mathit{in\_city}(p_2,c_1),\ \mathit{cuisine}(\mathit{Italian},\mathit{pizza}),\ \mathit{cuisine}(\mathit{Italian},\mathit{pasta}),\ \mathit{cuisine}(\mathit{Japanese},\mathit{sushi}),\\
&\mathit{cuisine}(\mathit{Japanese},\mathit{fish}),\ \mathit{cuisine}(\mathit{vegetarian},\mathit{pizza}),\ \mathit{cuisine}(\mathit{vegetarian},\mathit{salad})\}\,,\\
\Sigma \;=\; &\{\mathit{place}(T)\rightarrow\exists C\ \mathit{in\_city}(T,C),\ \mathit{serves}(P,F)\wedge\mathit{food}(F)\wedge\mathit{cuisine}(C,F)\rightarrow\mathit{serves}(P,C)\}\,.\ \blacksquare
\end{aligned}$$

2.2 Preference Datalog+/–

We now recall the PrefDatalog+/– language introduced in [21], which is a generalization of Datalog+/– with preferences. For a more general survey of preferences in the context of databases, we refer the reader to [33]. The approach to define preferences logically was pursued in [9].

In the following, we denote by Δ_{Ont}, $\mathcal{V}_{\mathit{Ont}}$, and $\mathcal{R}_{\mathit{Ont}}$ the infinite set of constants, the infinite set of variables, and the finite set of predicates, respectively, of standard Datalog+/– ontologies, as described in the previous section. For the preference extension, we assume a finite set of constants $\Delta_{\mathit{Pref}}\subseteq\Delta_{\mathit{Ont}}$, an infinite set of variables $\mathcal{V}_{\mathit{Pref}}\subseteq\mathcal{V}_{\mathit{Ont}}$, and a finite set of predicates $\mathcal{R}_{\mathit{Pref}}\subseteq\mathcal{R}_{\mathit{Ont}}$. These sets give rise to corresponding Herbrand bases $\mathcal{H}_{\mathit{Ont}}$ and $\mathcal{H}_{\mathit{Pref}}$, as well as Herbrand universes $\mathcal{U}_{\mathit{Ont}}$ and $\mathcal{U}_{\mathit{Pref}}$, respectively, consisting of all constructible ground atoms and ground terms, respectively, where $\mathcal{H}_{\mathit{Pref}}\subseteq\mathcal{H}_{\mathit{Ont}}$ and $\mathcal{U}_{\mathit{Pref}}\subseteq\mathcal{U}_{\mathit{Ont}}$.

A *preference relation* is any binary relation $\succ\,\subseteq\mathcal{H}_{\mathit{Pref}}\times\mathcal{H}_{\mathit{Pref}}$. In this work, we are interested in strict partial orders (SPOs), which are irreflexive and transitive relations — we consider these to be the minimal requirements for a preference relation to be useful in the applications that we envision. In the following, we consider two kinds of preference models that induce preference relations. The first, which follows the qualitative approach to specifying preferences, consists of a set of *preference formulas* (here, we consider a slight generalization of the proposal by [9]); these are first-order expressions of the form *pf*: $C(a,b)$ defining a preference relation $\succ_{\mathit{pf}}$ on $\mathcal{H}_{\mathit{Pref}}$ as follows: $a\succ_{\mathit{pf}}b$ if $C(a,b)$. We call $C(a,b)$ the *condition* of *pf*, denoted $\mathit{cond}(\mathit{pf})$. The preference relation defined via a set of preference formulas P is denoted by $\succ_P$.

Example 3 Consider again the ontology $O=(D,\Sigma)$ from Example 2. The food preferences of a user may be represented by the preference formulas in U_1 below, which encode that food that belongs to the *Italian cuisine* is preferred by the user over food that belongs to the *Japanese cuisine*, that *pizzerias* are preferred over *sushi bars*, and that places that serve *pasta* are preferred over those that serve *salad*. Figure 1, top left, shows the SPO induced by these preference formulas.

$$U_1:\begin{cases}C_1:\mathit{food}(X)\succ\mathit{food}(Y)\text{ if }\mathit{cuisine}(\mathit{Italian},X)\wedge\mathit{cuisine}(\mathit{Japanese},Y)\wedge X\neq Y\\ C_2:\mathit{place}(X)\succ\mathit{place}(Y)\text{ if }\mathit{type}(X,\mathit{pizzeria})\wedge\mathit{type}(Y,\mathit{sushi\_bar})\wedge X\neq Y\\ C_3:\mathit{place}(X)\succ\mathit{place}(Y)\text{ if }\mathit{serves}(X,\mathit{pasta})\wedge\mathit{serves}(Y,\mathit{salad})\wedge X\neq Y\,.\ \blacksquare\end{cases}$$

3 GPP-Datalog+/–

In this section, we introduce the GPP-Datalog+/– language, an extension of Datalog+/– with both a group preference model and a probabilistic model; it is based on several previously proposed formalisms: the

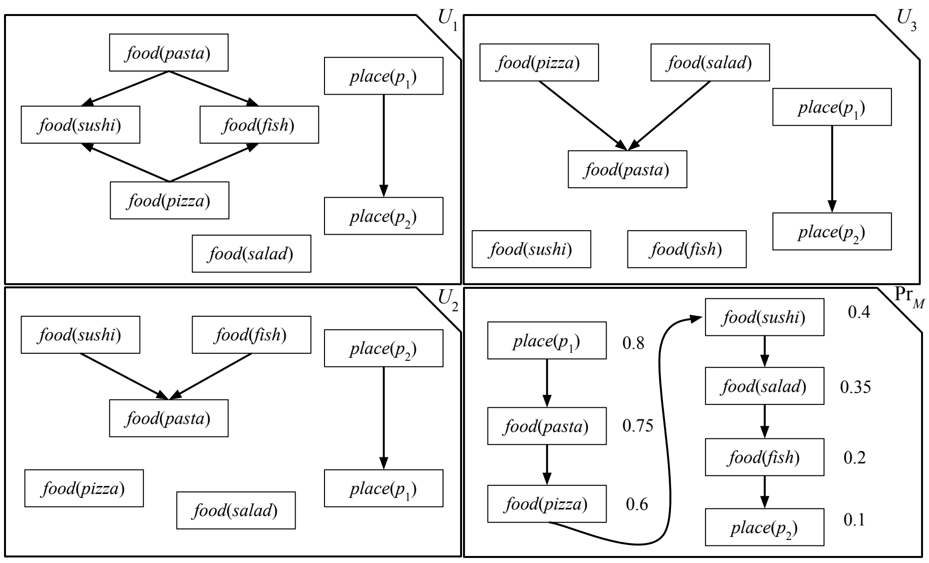


Figure 1: Group preferences and probability-based preference relation for Example 2.

PrefDatalog+/– language described above [21] (this language does not contemplate probabilities nor groups), PP-Datalog+/– [22] (which includes probabilities, but not groups), and G-PrefDatalog+/ [23] (including groups, but not probabilities). As we discuss in this paper, the combination of all three characteristics yields challenges not present in these previous proposals.

Group Preference Model We start by generalizing the preference models presented in Section 2.2 to represent preferences about groups of individuals. Specifically, a *group preference model* $\mathcal{U} = (U_1, \ldots, U_n)$ for $n \geqslant 1$ users is a collection of n user preference models.

Example 4 Consider again the Datalog+/– ontology $O = (D, \Sigma)$ from Example 2. Suppose that we have the following group preference model for three users $\mathcal{U} = (U_1, U_2, U_3)$:

$$U_1 : \begin{cases} C_{1,1} : \mathit{food}(X) \succ \mathit{food}(Y) \text{ if } \mathit{cuisine}(\mathit{Italian}, X) \wedge \mathit{cuisine}(\mathit{Japanese}, Y) \wedge X \neq Y, \\ C_{1,2} : \mathit{place}(X) \succ \mathit{place}(Y) \text{ if } \mathit{type}(X, \mathit{pizzeria}) \wedge \mathit{type}(Y, \mathit{sushi\_bar}) \wedge X \neq Y, \\ C_{1,3} : \mathit{place}(X) \succ \mathit{place}(Y) \text{ if } \mathit{serves}(X, \mathit{pasta}) \wedge \mathit{serves}(Y, \mathit{salad}) \wedge X \neq Y, \end{cases}$$

$$U_2 : \begin{cases} C_{2,1} : \mathit{food}(X) \succ \mathit{food}(\mathit{pasta}) \text{ if } \mathit{cuisine}(\mathit{Japanese}, X), \\ C_{2,2} : \mathit{place}(X) \succ \mathit{place}(Y) \text{ if } \mathit{type}(X, \mathit{sushi\_bar}) \wedge \mathit{in\_city}(X, c_1) \wedge X \neq Y, \end{cases}$$

$$U_3 : \begin{cases} C_{3,1} : \mathit{food}(X) \succ \mathit{food}(Y) \text{ if } \mathit{cuisine}(\mathit{vegetarian}, X) \wedge \mathit{cuisine}(\mathit{Italian}, Y) \wedge X \neq Y, \\ C_{3,2} : \mathit{place}(X) \succ \mathit{place}(Y) \text{ if } \mathit{type}(Y, \mathit{sushi\_bar}) \wedge X \neq Y. \end{cases}$$

Figure 1 (where we assume the transitive closure of the graphs) shows the SPOs induced by the preference models U_1, U_2, and U_3. ■

Probabilistic Model To incorporate the probabilistic model, in addition to what was stated in Section 2.2, we also assume the existence of a finite set of constants Δ_M, a finite set of predicates $\mathcal{R}_M$ such that $\mathcal{R}_M \cap \mathcal{R}_{Ont} = \emptyset$, and an infinite set of variables $\mathcal{V}_M$. We denote the corresponding *Herbrand base* (the sets of all possible ground atoms) with $\mathcal{H}_M$. We assume the existence of a probabilistic model M that represents a probability distribution $\Pr_M$ over some set $X = \{X_1, \dots, X_n\}$ of Boolean variables such that there is a 1-to-1 mapping from X to the set of all ground atoms over $\mathcal{R}_M$ and Δ_M. Examples of the type of probabilistic models that we assume in this work are Markov logic and Bayesian networks. The probabilistic extension adopted here was first introduced in [15, 14].

In the rest of this paper, we assume the existence of a preference relation $\succ_M$, defined as $a \succ_M b$ iff $\Pr_M(a) > \Pr_M(b)$ — we refer to $\succ_M$ as the probability-based (preference) relation.

Example 5 Continuing with the running example, suppose that we have access to an online probabilistic model that assigns probabilities specifying how likely it is that certain events happen. This uncertainty could arise, for instance, from the fact that the system is aggregating information from multiple sources, which may contain conflicting information, as well as uncertainty due to other factors. Such a system could inform the user of the probability of a place, specific food or cuisine, of being available and recommendable at the time of the query by taking into account reviews, crowds, season, etc. For instance, we can query the probability of a particular place, or type of place, of being open with availability of space, or the probability of finding in the area at a certain time a specific kind of food or cuisine.

Figure 1 gives an example of such a probability assignment, along with the preference relation as a graph that is induced by these values, assuming that higher probabilities are more preferable. The model M assigns to *place*(p_1) the highest probability, while it assigns to *place*(p_2) the lowest. ■

We are now ready to define GPP-Datalog+/– ontologies.

Definition 1 A *GPP-Datalog+/– ontology* has the form $KB = (O, \mathcal{U}, M)$, where O is a Datalog+/– ontology, $\mathcal{U} = (U_1, \dots, U_n)$ is a group preference model with $n \geqslant 1$, and M is a probabilistic model (with Herbrand bases $\mathcal{H}_{Ont}$, $\mathcal{H}_{Pref}$, and $\mathcal{H}_M$, respectively, such that $\mathcal{H}_{Pref} \subseteq \mathcal{H}_{Ont}$).

4 Query Answering in GPP-Datalog+/–

In this section, we concentrate on skyline queries [4], a well-known class of queries that can be issued over preference-based formalisms — intuitively, answers to skyline queries consist of those elements that are not dominated by any other according to the input preferences. We also focus on the iterated computation of skyline answers [9] that allows us to assign a *rank* to each answer by iterating the following procedure: (1) initialize ℓ to 0; (2) assign the rank ℓ to the skyline answers; and (3) remove from consideration all answers of rank ℓ, increment ℓ by one, and go back to (2). As a generalization of classical top-k queries, we adopt here k-rank queries; answers to them are k-tuples of answers sorted by rank.

4.1 Preference Merging and Aggregation

As seen in Figure 1, there are two challenges encountered in query answering with GPP-Datalog+/– ontologies. The first challenge is that each user preference model yields a certain precedence relation that might be in disagreement with the one induced by the probabilistic model. The second challenge is that the user preference models may be in disagreement with each other.

Preference Merging. To address the first challenge, we introduce the notion of *preference merging operators*, which are functions that take two preference relations and produce a third one satisfying two basic properties as stated below.

Definition 2 Let $\succ_U$ be an SPO and M be a probabilistic model. A *preference merging operator* $\otimes(M, \succ_U)$ yields a relation $\succ^*$ such that (i) $\succ^*$ is an SPO, and (ii) if $a_1 \succ_U a_2$ and $a_1 \succ_M a_2$, then $a_1 \succ^* a_2$.

The two properties required by Definition 2 are the minimal required to produce a "reasonable" merging of the two relations. A simple example of such an operator is to produce a new SPO that sorts elements according to their rank in $\succ_U$ and uses $\succ_M$ to break ties.

In the sequel, we adopt the following family of preference merging operators to combine individual preferences with those based on probabilities, defined by the following algorithm. Given a relation $\succ_U$, probability-based relation $\succ_M$, and value $t \in [0, 1]$ (that allows the user to choose how much influence the probabilistic model has on the output preference relation), the algorithm works by iterating through all pairs (a, b) of elements in $\succ_U$ and, if (i) $\succ_M$ disagrees with $\succ_U$, (ii) the difference in probability is greater than t, and (iii) changing (a, b) to (b, a) does not introduce a cycle in the associated graph, then the pair is inserted in reverse order into the output; otherwise, the output contains the same pair as $\succ_U$. Finally, the algorithm outputs the transitive closure of this relation. In the rest of this paper, we use the notation $\otimes_t$ to denote the particular instance of this operator $\otimes$ given a specific value of t. The following result shows that $\otimes_t$ is indeed a preference merging operator according to Definition 2.

Proposition 1 *Let $\succ_U$ be an SPO, M be a probabilistic model, and $t \in [0, 1]$. Then, $\otimes_t$ as defined above is a preference merging operator.*

The following is an example of our merging operator.

Example 6 Consider again the running example. Figure 3 shows the result of the individual mergings of the preference relation for each user with the probability-based relation using $t = 0$ for u_1, $t = 0.1$ for u_2, and $t = 0.3$ for u_3. Observe that even though user u_2 prefers *place*(p_2) to *place*(p_1), after merging with $\succ_M$, this preference is reversed (Figure 3). ■

Finally, the next proposition states that for $t = 0$, the result depends on the ordering given by $\succ_M$ and not the actual probabilities.

Proposition 2 *Let $\succ_U$ be an SPO, M be a probabilistic model, and $t \in [0, 1]$. If M' is a probabilistic model such that $\succ_M = \succ_{M'}$ and $t = 0$, then $\otimes_t(M, \succ_U) = \otimes_t(M', \succ_U)$.*

Preference Aggregation. To address the second challenge, we define *preference aggregation operators*, which are functions that take a set of preference models and produce a new one.

Definition 3 Let $\mathcal{U} = (U_1, \ldots, U_n)$ be a group preference model, where every U_i is an SPO. A *preference aggregation operator* $\uplus$ on $\mathcal{U}$ yields an SPO $\succ^*$.

A simple example of an aggregation operator is to consider all pairs of elements and do a Pareto composition [9].

We now explore two different approaches to a preference aggregation operator $\uplus$. In the first one, called *collapse to single user (CSU)*, we reduce the group modeling problem to a single-user problem by creating

```
Algorithm 1: AggPrefsCSU($\succ_{U_1}, \ldots, \succ_{U_n}$)
Input: SPOs ($\succ_{U_1}, \ldots, \succ_{U_n}$).
Output: Preference relation $\succ^* \subseteq \mathcal{H}_{Ont} \times \mathcal{H}_{Ont}$.
 1. initialize $G$ as an empty graph;
 2. add as nodes in $G$ all elements appearing in the preference relations $\succ_{U_i}$;
 3. for every user $i \in \{1, \ldots, n\}$ do
 4.    initialize currUserG as the graph corresponding to $\succ_{U_i}$;
 5.    for every edge $(u, v)$ in currUserG do
 6.       if there is no edge $(u, v)$ in $G$ then
 7.          add edge $(u, v)$ to $G$ and label it with 1;
 8.       if there is an edge $(u, v)$ in $G$ and it is labeled with $n \geqslant 1$ then
 9.          increase the label of edge $(u, v)$ in $G$ by 1;
10.       if there is an edge $(v, u)$ in $G$ and it is labeled with 1 then
11.          remove edge $(u, v)$ from $G$;
12.       if there is an edge $(v, u)$ in $G$ and it is labeled with $n > 1$ then
13.          decrease the label of edge $(v, u)$ in $G$ by 1;
14. return inducedPreferenceRelation(removeCycles(transitiveClosure($G$))).
```

Figure 2: An algorithm for combining the relations in a group preference model with a probabilistic preference relation.

a single virtual user that is constructed by aggregating the preferences of the individuals from the group. In the second approach, called *voting-based aggregation*, we first compute the k-rankings according to each individual user and then apply aggregation techniques based on voting strategies (originally developed for quantitative preferences [27]) to aggregate the relations induced by the rankings.

4.1.1 Collapse to Single User

Under the CSU strategy, the preference relation for all users, along with the probabilistic preference relation, are taken into account in the generation of a new preference relation that encodes the dominant preferences. This single-user preference relation is then used to compute the answers to queries. The following algorithm computes this particular approach to defining a $\uplus$ operator; below, we provide an algorithm that uses this operator for answering k-rank queries to GPP-Datalog+/– ontologies in polynomial time in the data complexity (modulo the cost of computing probabilities with respect to the probabilistic model M).

Algorithm AggPrefsCSU**.** The algorithm in Figure 2 implements preference aggregation operator $\uplus_{CSU}$; the output is a new preference relation consisting of the collapsed preferences of all relations. A graph is used as an intermediate data structure representing the collapsed preferences; the nodes of this graph are all the atoms that appear in the preference relations, while the edges are labeled with an integer representing the number of relations that have this edge in their individual graph. The algorithm iterates through all the users i, looks at all the edges in *currUserG*, and updates the general graph G by incrementing or decrementing the edge labels and introducing or removing edges. After the final iteration of the for-loop in Line 3, the edge labels of G correspond to the number of users that have that edge in their preference relation. The final step of the algorithm computes the transitive closure of the graph and eliminates any cycles by applying the procedure *removeCycles* (note that cycles can arise even though all individual relations are cycle-free). We say that this subroutine does not unnecessarily remove edges if there does not exist an edge e in G such that *removeCycles*$(G) \cup \{e\}$ does not contain cycles. Since this subroutine is not explicitly given, $\uplus_{CSU}$ is

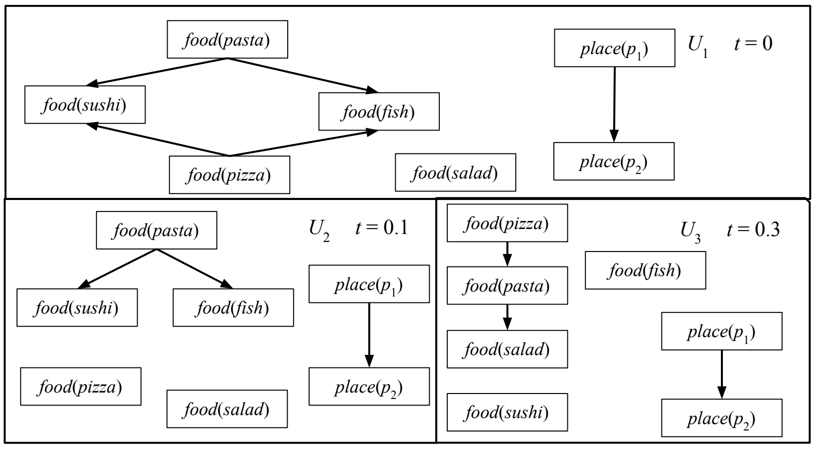


Figure 3: The merged preference relation obtained for each user. The three graphs show the merging of each individual preference with the probabilistic preference.

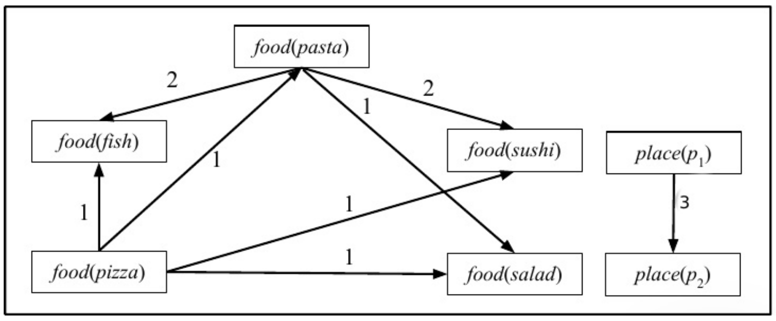


Figure 4: Collapse to single user graph.

actually a family of operators.

Example 7 Continuing from Example 6, Figure 4 shows the final collapsed graph. Consider the atoms *place*(p_1) and *place*(p_2) in this graph. We mentioned in Example 6 that the preference of user u_2 was reversed, because of the probabilities for those items; as a result, we have an unanimous preference of *place*(p_1) over *place*(p_2) in the final graph. ■

The following theorem states that $\uplus_{CSU}$ satisfies Definition 3.

Theorem 3 *Let* $KB = (O, \mathcal{U}, M)$ *be a GPP-Datalog+/– ontology, where* $\mathcal{U} = (U_1, \ldots, U_n)$*. Let* $\succ^* = \uplus_{CSU}$ $(\succ_{U_1}, \ldots, \succ_{U_n})$*. Then, the following properties hold: (i) if removeCycles preserves transitivity, then* $\succ^*$ *is a preference aggregation operator; and (ii) if removeCycles only removes edges* (v_1, v_2) *whenever there does not exist another edge in the cycle labeled with a lower number then, given* $a_1, a_2 \in \mathcal{H}_{Ont}$ *such that for all* $U_i \in \mathcal{U}$ *it holds that* $(a_1, a_2) \in U_i$*, then we have that* $a_1 \succ^* a_2$*.*

4.1.2 Voting-Based Preference Aggregation

As an alternative to the approach described in the previous section, we now briefly discuss specific strategies that can be used to combine the answers to k-rank queries computed individually for each user based on a small set of well-known voting mechanisms from the social choice literature. Recall that this is essentially different from the CSU approach above, where a single k-ranking is computed from a preference relation distilled from all the users' individual preferences. We consider the following voting mechanisms: *plurality voting*, where each user votes for their top-preferred items, the items' frequency for all the users are summed up, and the items with highest number of votes win; the *least misery* strategy first removes from consideration the elements that are the least preferred by each user, and then applies plurality voting — the idea behind it is that a group is as happy as its least happy member; in the *average without misery* strategy, the least misery approach is generalized by removing the t least liked elements for each member (instead of just one); and the *fairness strategy*, which is often applied when people try to fairly divide a set of items — one person chooses first, then another, until everyone has made one choice, and next, everybody chooses a second item, often starting with the person who had to choose last in the previous round; an advantage of this strategy is that the top items from all individuals are always selected.

In the following, we use $\uplus_{\mathit{vote}}$ to denote the family of preference aggregation operators defined as: (i) for each input SPO, compute a ranking (linear order); and (ii) apply a voting-based strategy over these linear orders. For particular voting strategies, we use specific names such as $\uplus_{\mathit{plu}}$, $\uplus_{\mathit{plu,fair}}$, $\uplus_{\mathit{plu,mis}}$, and so on. Therefore, we must adapt the voting strategies to work over linearly sorted lists of elements. Plurality voting then works by assigning one vote to each element in the list. If a *misery* strategy is used, then each user's undesired elements are marked as unavailable before obtaining the individual rankings. Finally, if *fairness* is used, we iterate through each user, who picks their highest-ranked elements in turn.

Example 8 Consider the preference relations for users u_1, u_2, and u_3 from Figure 6 from our running example. The first step towards applying a voting-based aggregation technique is to compute a ranking for each user from their corresponding SPOs. For $k = 4$, the following are some possible k-rankings for these users:

$$\begin{aligned}
r_1 &= \langle \mathit{place}(p_1), \mathit{food}(\mathit{pasta}), \mathit{food}(\mathit{pizza}), \mathit{food}(\mathit{salad})\rangle,\\
r_2 &= \langle \mathit{food}(\mathit{pasta}), \mathit{food}(\mathit{pizza}), \mathit{place}(p_1), \mathit{food}(\mathit{salad})\rangle,\\
r_3 &= \langle \mathit{food}(\mathit{pizza}), \mathit{food}(\mathit{fish}), \mathit{food}(\mathit{sushi}), \mathit{place}(p_1)\rangle.
\end{aligned}$$

If we apply the aggregation operator $\uplus_{\mathit{plu}}$, then we have 3 votes for $\mathit{place}(p_1)$, 3 votes for $\mathit{food}(\mathit{pizza})$, 2 votes for $\mathit{food}(\mathit{salad})$ and $\mathit{food}(\mathit{pasta})$, 1 vote for $\mathit{food}(\mathit{fish})$ and $\mathit{food}(\mathit{sushi})$, and 0 votes for $\mathit{place}(p_1)$. The SPO output by $\uplus_{\mathit{plu}}$ arises from this tally. ■

4.2 Answering k-Rank Queries to GPP-Datalog+/– Ontologies

In this section, we first consider disjunctive atomic queries (DAQs) and then briefly discuss the extension to conjunctive queries (CQs). We show that the former can be answered in polynomial time, while answering the latter turns out to be Σ_2^p-complete.

We use the standard notions of substitutions and most general unifiers. More specifically, a *substitution* is a mapping from variables to variables or constants. Two sets S and T *unify* via a substitution θ iff $\theta S = \theta T$, where θA denotes the application of θ to all variables in all elements of A; here, θ is a *unifier*.

```
Algorithm 2: $k$-Rank-GPP$(KB, Q, k, \mathbf{t}, \otimes, \uplus)$
Input: GPP-Datalog+/– ontology $KB = (O, \mathcal{U}, M)$, where $\mathcal{U} = (U_1, \ldots, U_n)$, DAQ $Q(\mathbf{X})$,
       $k \geqslant 0$, $\mathbf{t} = (t_1, \ldots, t_n) \in [0,1]^n$, and operators $\otimes$ and $\uplus$.
Output: $k$-rank answer $\langle a_1, \ldots, a_{k'} \rangle$ to $Q$, with $k' \leqslant k$.
 1. for every user $i \in \{1, \ldots, n\}$ do
 2.     $\succ_i := \otimes_{t_i} (M, \succ_{U_i})$;
 3. return iteratedSkyline$(KB, Q, \uplus(\succ_1, \ldots, \succ_n), k)$.
```

Figure 5: Algorithm for computing a k-rank answer to DAQ Q.

A *most general unifier (mgu)* is a unifier θ such that for all other unifiers ω, there is a substitution σ such that $\omega = \sigma \circ \theta$.

We now formally define skyline and k-rank answers to DAQs in a GPP-Datalog+/– ontology.

Definition 4 Let $KB = (O, \mathcal{U}, M)$ be a GPP-Datalog+/– ontology, where $\mathcal{U} = (U_1, \ldots, U_n)$, $\otimes$ and $\biguplus$ be preference merging and combination operators, respectively, and let $Q(\mathbf{X}) = q_1(\mathbf{X}) \vee \cdots \vee q_n(\mathbf{X})$ be a DAQ. A *skyline answer* to $Q(\mathbf{X})$ relative to $\succ^* = \biguplus(\otimes(M, \succ_{U_1}), \ldots, \otimes(M, \succ_{U_n}))$ is any θq_i entailed by O such that no θ' exists with $O \models \theta' q_j$ and $\theta' q_j \succ^* \theta q_i$, where θ and θ' are ground substitutions for the variables in $Q(\mathbf{X})$. For transitive relations, a *k-rank* answer to $Q(\mathbf{X})$, $k \geqslant 0$, is a sequence $S = \langle \theta_1 q_{l_1}, \ldots, \theta_{k'} q_{l_{k'}} \rangle$ of maximal length of ground instances $\theta_i q_{l_i}$ of atoms q_{l_i} in $Q(\mathbf{X})$, built by subsequently appending the skyline answers to $Q(\mathbf{X})$, removing these atoms from consideration, and repeating the process until (a) either the length of S is k, or (b) no more skyline answers to $Q(\mathbf{X})$ remain.

In Definition 4, note that k-rank answers are only defined when the preference relation is transitive; this kind of answer can be seen as a generalization of traditional top-k answers [33] that are still defined when $\succ^*$ is not a weak order, and their name arises from the concept of *rank* introduced in [9]. Conceivably, other ways of sorting answers given an SPO are possible; here, we focus on iterated skylines, since it is the most general approach.

Intuitively, for DAQs, both kinds of answers can be seen as atomic consequences of O that satisfy the query: the skyline answers can be seen as sets of atoms that are not dominated by any other such atom, while k-rank answers are k-tuples sorted according to the preference relation. We refer to these as answers in *atom form*.

We now present a general algorithm for obtaining k-rank answers using the machinery developed up to now. Figure 5 presents the pseudocode — essentially, the algorithm follows Definition 4 by first applying the merging operator $\otimes$ to each user's preference relation and then applying the aggregation operator $\uplus$. Finally, the result is obtained by computing a k-rank over this aggregation.

Example 9 Consider the running example, with $Q = \mathit{food}(X)$, For $k = 4$ and $\mathbf{t} = (t_1, t_2, t_3) = (0, 0.1, 0.3)$, one possible k-rank answer to Q given by both k-Rank-GPP$(KB, Q, k, \mathbf{t}, \otimes, \uplus_{CSU})$ and k-Rank-GPP$(KB, Q, k, \mathbf{t}, \otimes, \uplus_{plu})$ is $\langle \mathit{food}(\mathit{pizza}), \mathit{food}(\mathit{pasta}), \mathit{food}(\mathit{salad}), \mathit{food}(\mathit{sushi}) \rangle$ — clearly, if different rankings from the ones computed in Example 8 are used, then the result with respect to $\uplus_{plu}$ could be different from that of $\uplus_{CSU}$. ■

The following theorem proves the correctness of the k-Rank-GPP algorithm, and it shows that it runs in polynomial time under certain conditions.

Theorem 4 *Let KB* $= (O, \mathcal{U}, M)$ *be a GPP-Datalog+/– ontology,* $\otimes$ *and* $\uplus$ *be merging and aggregation operators, respectively,* Q *be a DAQ, and* $k \geqslant 0$*. Then, (i) Algorithm* k-Rank-CSU *correctly computes* k*-rank answers to* Q*; (ii) if* $\uplus = \uplus_{CSU}$ *and the removeCycles subroutine does not unnecessarily remove any edges, then* k-Rank-CSU *runs in* $O(poly(\|D\|) \cdot S + C)$ *time in the data complexity; and (iii) if* $\uplus = \uplus_{vote}$*, and* $\uplus$ *can be computed in polynomial time in the data complexity, then* k-Rank-CSU *runs in* $O(poly(\|D\|) \cdot S)$ *time in the data complexity. Here,* $\|D\|$ *denotes the size of* D*, poly*$(\|D\|)$ *is a polynomial in* $\|D\|$*,* S *is the cost of computing* $\Pr_M(a)$ *for any atom* a *such that* $O \models a$*, and* C *is the cost of removeCycles.*

Note that the running time depends on the cost of the *removeCycles* subroutine. Though cycles can be removed in polynomial time, depending on the properties that we wish the output of this subroutine to satisfy, the actual cost may vary considerably — for instance, if we wish to remove the minimum number of edges possible, this is already NP-complete. The computational cost also depends on the implementation of the voting strategies, which can clearly be computed in polynomial time in the data complexity for the strategies discussed above; since cycles can never arise, the C factor from part (ii) does not appear.

k**-Ranking for Conjunctive Queries.** For (non-atomic) conjunctive queries, the substitutions in answers no longer yield single atoms but rather *sets* of atoms; thus, to answer such queries relative to a preference relation, we must extend the preference specification framework to take into account sets of atoms instead of individual ones. One such approach was proposed in [37], where a mechanism to define a preference relation over tuple sets $\succ_{PS}$: $2^{\mathcal{H}_{Pref}} \times 2^{\mathcal{H}_{Pref}}$ is introduced. In the following, we briefly treat their complexity and discuss how methods from the relational databases can be applied to answer them. The following result says that this change has a direct impact on the complexity of both skyline and k-rank query answering; its proof is a straightforward generalization of the result presented in [21, Theorem 16].

Theorem 5 *Let KB* $= (O, \mathcal{U}, M)$ *be a GPP-Datalog+/– ontology, where* $O = (D, \Sigma)$ *and* $\mathcal{U} = (U_1, \ldots, U_n)$*. Let each* $\succ_{U_i}$ *describe a preference relation* $\succ_{PS}$: $2^{\mathcal{H}_{Pref}} \times 2^{\mathcal{H}_{Pref}}$ *such that membership can be tested in polynomial time. Let* $\otimes$ *and* $\uplus$ *be preference merging and aggregation operators, respectively, and let* Q *be a CQ. If the relational schema and* Σ *are fixed, deciding whether the set of skyline answers or a* k*-rank answer to* Q *are non-empty is* Σ_2^p*-complete.*

The heuristic algorithms presented in [37] for relational databases can be applied to GPP-Datalog+/– by computing the *chase* relative to the query and thus materializing the necessary part of the ontology into a database. The result in [21, Theorem 17] provides a way to leverage tools developed for relational databases for first-order (FO) rewritable fragments; if the user preferences are given as sets of preference formulas [9], this result can be easily extended for GPP-Datalog+/– ontologies.

5 Semantic Properties of k-Ranking over Group Preferences

We now study a set of properties that can be considered desirable for the results of the aggregation of group preferences. These properties are based on the ones usually studied in social choice theory [12], but extended for k-rank query answers. Since the two kinds of aggregation strategies being studied are different in nature, we study a different set of properties for each; essentially, the ones for voting-based strategies focus on the ordering of elements (by individual users, subgroups, and groups), while the properties for CSU focus on the relationship between specific pairs of elements. Note that these properties are only with respect to the aggregation strategies, and do not involve the merging operations used to combine SPOs with preferences arising from probabilistic models.

In the presentation of the properties, when we say that *an element is added*, we refer to the modification of individual SPOs by adding a new element along with any preference edges between that element and existing ones; we assume that the rest of the SPO is left unchanged, except for the addition of any necessary edges to satisfy transitivity. Similarly, when we say that *an element is removed*, we refer to the operation of removing the element from the SPO and any preference edges associated with it. In the sequel, let $\mathcal{P} = (U_1, \ldots, U_n)$, $\mathcal{P}_A = (U_1^A, \ldots, U_n^A)$, and $\mathcal{P}_B = (U_1^B, \ldots, U_n^B)$ be group preference models, and $\mathcal{P}_{A \cup B} = (U_1^A, \ldots, U_n^A, U_1^B, \ldots, U_m^B)$. Given a group preference model $\mathcal{P}$, we refer to the SPO corresponding to the aggregation of the preferences in $\mathcal{P}$ as $\succ^*_{\mathcal{P}} = \uplus\,(\succ_{U_1}, \ldots, \succ_{U_n})$.

5.1 Semantic Properties of Voting-Based Strategies

Unanimous Winner (UW): If an element a is in a k-rank answer to Q for each $U_i \in \mathcal{P}$, then there exists a k-rank answer to Q for $\mathcal{P}$ that contains element a.

Unanimous Loser (UL): If element a is in none of the k-rank answers to Q for each $U_i \in \mathcal{P}$, then a does not belong to any k-rank answer to Q for $\mathcal{P}$.

In the following, let $\mathcal{P}_{add}$ be a group preference model obtained from $\mathcal{P}$ by adding a new element c to choose from and $\mathcal{P}_{rem}$ be a group preference model obtained from $\mathcal{P}$ by removing from consideration an element c.

Weak Stability (WS): For each k-rank answer $r = \langle a_1, \ldots, a_k \rangle$ to Q for $\mathcal{P}$, there exists a k-rank answer $r' = \langle b_1, \ldots, b_k \rangle$ to Q for $\mathcal{P}_{add}$ such that either $r = r'$ or $\{b_1, \ldots, b_k\} - \{a_1, \ldots, a_k\} = \{c\}$.

This property states that whenever a new element is added, the set of elements in the result either remains unchanged or differs in the added element.

Stability 1 (S1): For each k-rank answer $r = \langle a_1, \ldots, a_k \rangle$ to Q for $\mathcal{P}$, there exists a k-rank answer $r' = \langle b_1, \ldots, b_k \rangle$ to Q for $\mathcal{P}_{add}$ such that either $r = r'$ or (i) $\{b_1, \ldots, b_k\} - \{a_1, \ldots, a_k\} = \{c\}$, (ii) for some j, $1 \leqslant j \leqslant k$, $b_j = c$, and (iii) for each b_i, b_j such that $1 \leqslant i < j \leqslant k$ and $b_i = a_{i'}$ and $b_j = b_{j'}$ it holds that $i' < j'$.

As a stronger version of WS, this property requires the order to be the same among the elements in the result.

Stability 2 (S2): For each k-rank answer $r = \langle a_1, \ldots, a_k \rangle$ to Q for $\mathcal{P}$, if $c \neq a_i$ for $1 \leqslant i \leqslant k$, then every k-rank answer $r = \langle b_1, \ldots, b_k \rangle$ to Q for $\mathcal{P}_{rem}$ is such that $b_i = a_i$ and $b_i \neq c$, for $1 \leqslant i \leqslant k$.

If an element not in the k-rank is removed from consideration, the result must be the same.

Monotonicity 1 (M1): Let $r = \langle a_1, \ldots, a_{i-1}, c, a_{i+1}, \ldots, a_k \rangle$ be a k-rank answer to Q for some $U_i \in \mathcal{P}$, and $\mathcal{P}'$ be equal to $\mathcal{P}$ except that one of the U_i's is changed to U_i' such that there exists a k-rank answer $r' = \langle a_1', \ldots, a_{j-1}', c, a_{j+1}', \ldots, a_k' \rangle$ to Q with $j \leqslant i$. Then, there exists a k-rank answer $r'' = \langle b_1, \ldots, b_k \rangle$ to Q for $\mathcal{P}'$ such that $b_m = c$, for some $1 \leqslant m \leqslant k$.

Intuitively, Monotonicity 1 states that if an element is in a k-rank, it is still in a k-rank if a profile changes to rank it higher.

Monotonicity 2 (M2): Let c be an element such that for every k-rank answer $r = \langle a_1, \ldots, a_k \rangle$ to Q for $\mathcal{P}$ we have $c \neq a_i$ for $1 \leqslant i \leqslant k$, and let $\mathcal{P}'$ be equal to $\mathcal{P}$ except that one of the U_i's is changed to U_i' such that for every k-rank answer $r' = \langle a_1', \ldots, a_k' \rangle$ to Q for U_i' we have that $c \neq a_i'$ for $1 \leqslant i \leqslant k$. Then, every k-rank answer $r = \langle b_1, \ldots, b_k \rangle$ to Q for $\mathcal{P}'$ is such that $c \neq b_i$ for $1 \leqslant i \leqslant k$.

	UW	UL	WS	S1	S2	M1	M2	ND
P	✓	✓	✓	✓	✓	✓	✓	✓
F	×	✓	✓	✓	✓	✓	✓	× $(k = 1)$ / ✓ $(k \neq 1)$
PM	✓	✓	✓	✓	✓	✓	✓	✓
FM	×	✓	✓	✓	✓	✓	✓	✓

Figure 6: Summary of properties satisfied by each voting-based strategy.

If an element is not in the k-rank, then it is not in the k-rank when some profile is changed to lower its rank.

Non-dictatorship (ND): There is no U_i in $\mathcal{P}$ such that the k-rank answers to Q for $\mathcal{P}$ are determined by that preference model alone.

Proposition 6 *The following properties hold for the specific voting-based strategies:*

- Plurality *satisfies* UW, UL, WS, S1, S2, M1, M2, *and* ND.
- Fairness *satisfies* UL, WS, S1, S2, M1, M2, *and* ND *with* $k \neq 1$.
- Plurality with misery *satisfies* UW, UL, WS, S1, S2, M1, M2, *and* ND.
- Fairness with misery *satisfies* UL, WS, S1, S2, M1, M2, *and* ND.

5.2 Semantic Properties of the CSU Strategy

Group Union Preservation First Position (GUPF): Let $r = \langle a_1, \dots, a_k \rangle$ and $r' = \langle b_1, \dots, b_k \rangle$ be k-rank answers to Q for $\mathcal{P}_A$ and $\mathcal{P}_B$, respectively. If $a = a_1 = b_1$, then there exists a k-rank answer $r'' = \langle a, c_1, \dots, c_{k-1} \rangle$ to Q for $P_{A \cup B}$.

Intuitively, if element a appears in the first position of a k-rank answer to Q for both $\mathcal{P}_A$ and P_B, then a appears in the first position of a k-rank answer to Q for the union of the group preference models.

UW First Position (UWF): If for every U_i in $\mathcal{P}$, there exists a k-rank answer to Q for U_i of the form $\langle a, b_1, \dots, b_{k-1} \rangle$, then there exists a k-rank answer to Q for $\mathcal{P}$ of the form $\langle a, c_1, \dots, c_{k-1} \rangle$.

Intuitively, if element a appears in the first position of a k-rank answer to Q for each $U_i \in \mathcal{P}$, then a appears in the first position of a k-rank answer to Q for $\mathcal{P}$.

Unanimity on preference relation (UP): If *removeCycles* only removes edges (v_1, v_2) whenever there does not exist another edge in the cycle labeled with a lower number, then whenever $t_1 \succ_{U_i} t_2$ for each U_i in $\mathcal{P}$, then $t_1 \succ^*_{\mathcal{P}} t_2$.

Proposition 7 *The CSU strategy (as implemented in Algorithm* AggPrefsCSU*) satisfies* GUPF, UWF, *and* UP.

6 Implementation and Experimental Evaluation

In this section, we evaluate and analyze the running time of our algorithms and the quality of their results, over experiments done with real-world data. Further details are included in Appendix C.

City	No. of Businesses	Average No. of Nodes	Average No. of Pref. Edges
Peoria	109	42.33	1,136.53
Gilbert	163	61.68	2,872.91
Glendale	242	91.93	7,171.96
Chandler	349	136.17	16,222.71
Tempe	465	180.97	27,835.80

Figure 7: Size information for the different subsets of the used dataset and corresponding average size of the CSU-PrefChase built during the query answering process (using the CSU aggregation strategy).

6.1 Implementation and Hardware

We implemented G-PrefDatalog+/– by extending the Datalog+/– query answering engine in [16], which supports FO-rewritable fragments of the Datalog+/– family of ontology languages. Essentially, the extension involved adding query answering based on group preferences, as well as handling of merging and aggregation operators. As for the latter, we implemented collapse to single user (CSU), plurality voting, and plurality voting with misery. All graph operations were implemented using the JGraphT library (`http://jgrapht.org/`), which provides efficient data structures for the representation of graph structures as well as efficient implementations of operations such as reachability and cycle detection. The entire implementation was done in Java.

All runs were done on an Intel Core i7 processor at 2.2 GHz and 8GB of RAM, under the Mac OS X 10.6.8 operating system and a Sun JVM Standard Edition with maximum heap size set at 512 MB of RAM. To minimize experimental variation, all results are averages of three independent runs.

6.2 Experimental Setup

Inputs to our system consist of tuples of the form $\langle Q, O, \mathcal{P}, k\rangle$ with $O = (D, \Sigma)$, where Q is a query, D is a database, Σ is a finite set of TGDs, $\mathcal{P}$ is a group preference model, and k is the number of query results that we are interested in. The CSU-PrefChase consists of a graph that stores all user preferences after constructing the chase. The preference graph is a labeled directed multigraph (N, E, ℓ), where N is the node set (the atoms), E is the edge set (the preference relation), and the labeling function ℓ stores for each edge the identity of its user. That is, edges in E are pairs of nodes (u, v), which state that u is preferred to v by the user specified in the label $\ell(u, v)$. The CSU-PrefChase is adapted for obtaining the answer to all queries, depending on the chosen strategy. A high-level overview of the architecture of our system can be found in Appendix C.

Data. All runs were carried out using an ontology built on the basis of the data from the Yelp Dataset Challenge [36]; this dataset contains $11,537$ businesses in the Phoenix metropolitan area in the United States, $8,282$ check-in sets, $43,873$ users, and $229,907$ reviews — each business has one or more associated categories. The Datalog+/– ontology used for these experiments was constructed as follows. We manually wrote TGDs using categories related to the dataset, yielding a set of 53 TGDs. For example, the “Asian” and “Chinese” categories give rise to the TGD $\mathit{chinese}(X) \rightarrow \mathit{asian}(X)$. The database instance was automatically generated using the mapping between businesses and categories. To test our algorithms on instances of different sizes, we considered five different subsets of the dataset corresponding to different cities in the state of Arizona (cf. Figure 7). Finally, the database for this ontology was stored in a PostgreSQL 9.3 database. See Appendix C for further details on the construction of the underlying ontology.

User preferences. We requested users to define preferences over places along two dimensions: meal (break-fast, lunch, and dinner) and details of the meal: *cuisine* (Italian, Asian, etc.), *type of food* (pizza, sandwich, etc.), and *type of place* (bar, cafe, etc.) — each combination of these produced a set of nine so-called *choice scenarios*, such as "cuisine for lunch" or "type of food for dinner". Preferences were entered using a graphical interface (illustrated in Figure 14 in Appendix C). A total of 49 people responded to our request, yielding a total of 364 SPOs (not all users entered their preferences for all nine scenarios). These SPOs were then processed using the set of businesses in the dataset in such a way that business X is preferred over business Y iff the SPO establishes the preference over their corresponding categories (for instance, if the users specified that they prefer bagels over sushi, then all bagel businesses are preferred over all sushi businesses).

Group definition. The groups of users were created manually, such that all members know each other. From the total of 49 users, we generated 19 different groups, ranging from 3 to 9 users. This produced a grand total of $19 \cdot 9 = 171$ group choice scenarios; for each data point in the group size variation experiments below, we randomly chose ten SPOs and reported the average.

Probabilistic model and choice of parameters. For each business, the Yelp dataset provides a numerical rating ranging from 0 to 5. We used this information to create our probabilistic model by using the category supplied in the dataset; the probability assigned to a business in a category is computed as $(\mathit{rating}+0.5)/5.5$ if the categories match, and zero otherwise. Though more complex models are of course possible, this is outside the scope of our experiments. Finally, values for the parameter t were generated automatically as uniformly distributed random values in the interval $[m-\sigma, m+\sigma]$, where m is the mean value of the ratings, and σ is the standard deviation.

Queries. We used the following three queries: $Q_1(X) = \mathit{food}(X)$, $Q_2(X) = \mathit{cuisine}(X)$, and $Q_3(X) = \mathit{place}(X)$, representing the situations where groups wish to decide about where to go to eat, based on the preferences over kinds of food, cuisine, and place, respectively. Unless stated otherwise, all experiments were run with $5 \leqslant k \leqslant 20$, in steps of 5.

6.3 Performance Evaluation

The main performance metric that we investigate in this study is the time that is required to carry out query answering when certain parameters are varied. Figure 8, left side, shows scatterplots for the running time when three different parameters are varied: aggregation strategy (one graph per strategy), group size (x axis), and number of businesses in the database (different markers in each graph); as described above, the numbers for the latter arise from the selection of five different cities that were chosen in order to evaluate how running times vary when the number of businesses increases gradually. Note that groups of size 6 do not have data points associated with them — this is due to the fact that our data did not contain enough instances of combinations of groups of this size with choice scenarios. The dotted/dashed lines in these graphs correspond to trend lines showing the increase in running time when the group size increases.

The graphs in Figure 8, right side, show how the running time increases as the number of businesses in the database increases (each point is an average over the group sizes considered) — note that the y axis in these figures is on a logarithmic scale. These results show that our prototype implementation yields a top-3 answer for a city with 465 businesses to choose from in about 27 seconds. The table in Figure 7 shows the number of nodes in the preference chase for each city, as well as the number of edges for the collapsed graph when the CSU strategy is used.

The results in Figure 8 show that the three different strategies have similar running times for these runs; however, the way in which the running time is used by the two main approaches (CSU and plurality) is different. To illustrate this difference, Figure 9 plots the partial times in pie charts; for instance, observe

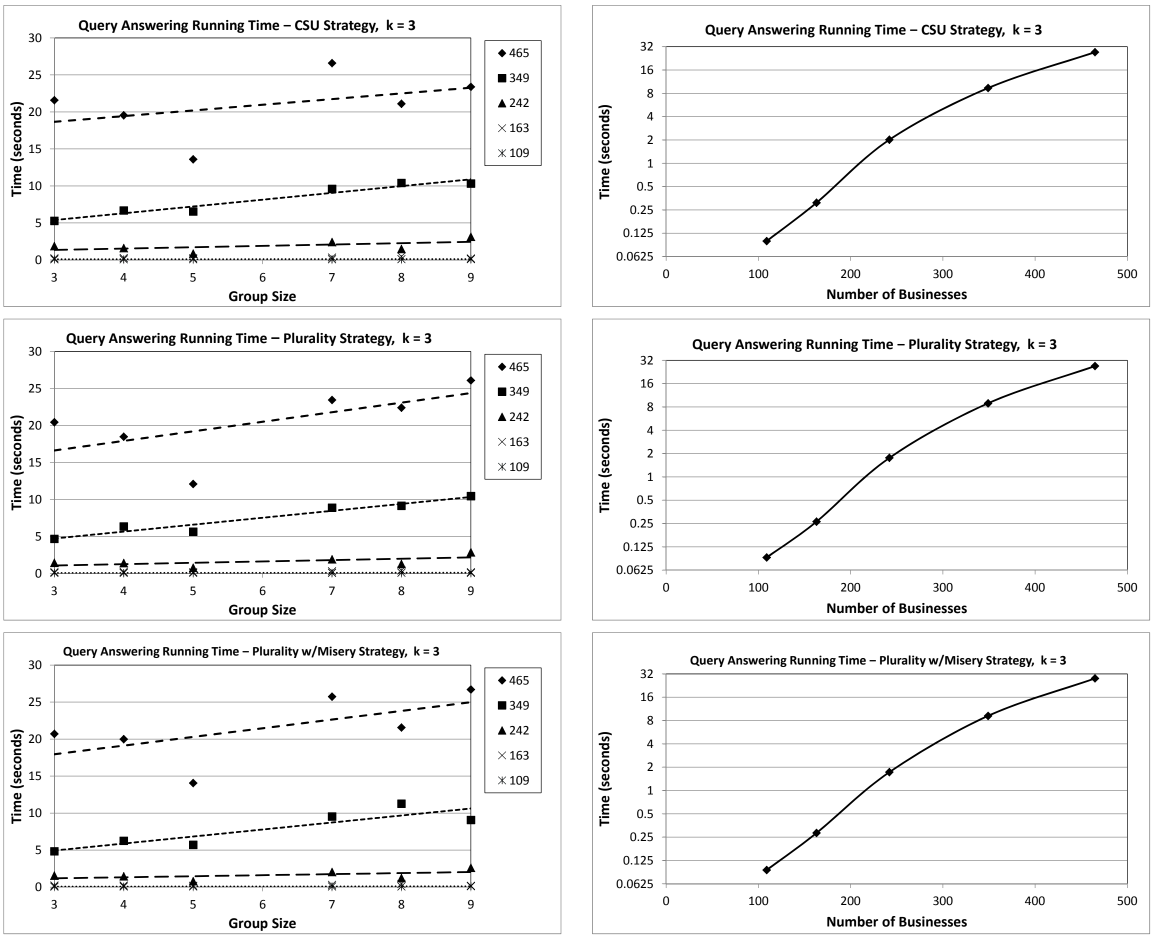


Figure 8: Running times for top-3 query answering when varying the number of businesses in the database, the group size, and the preference aggregation strategies.

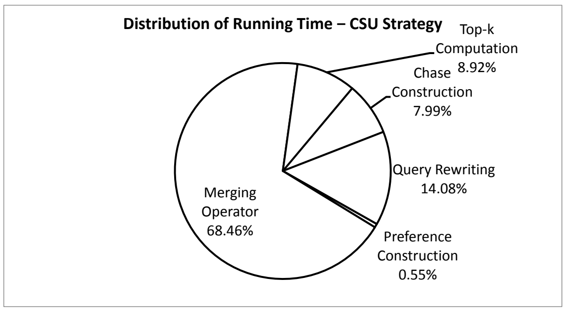
that CSU takes more time computing the top-k, while plurality takes more time computing the result of the merging operator.

6.4 Quality Evaluation

In this section, we discuss experiments carried out to evaluate the quality of the results produced by our algorithms. We first discuss the metrics used, and then go on to discuss the results. The experimental setup is identical to the one used in the previous experiments, except that we focused on a single city (*Gilbert*), since the number of businesses is not varied.

6.4.1 Evaluation Metrics

Usually, methods based on precision and recall are used when comparing two lists of results — however, since these methods rely on the availability of "ground truth" and there is no clear notion of what such a



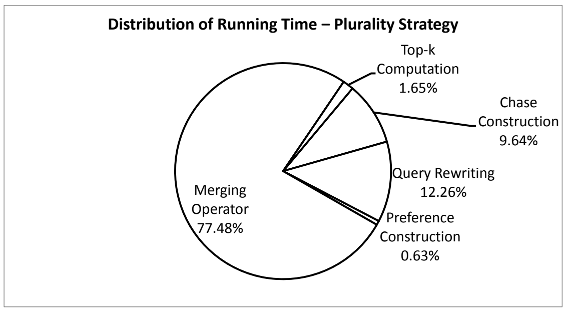


Figure 9: Distribution of running times for the CSU and plurality strategies.

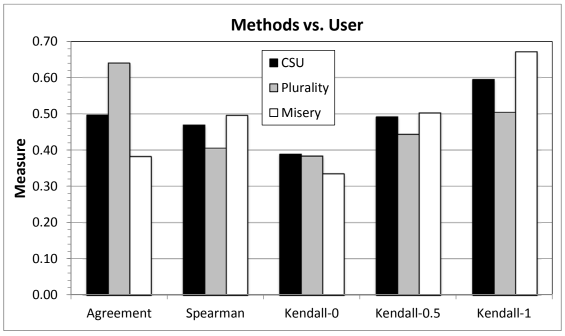
ground truth is for our setting, these methods are not applicable. Therefore, we use quality measurements that are often applied in evaluating information retrieval and group recommender systems [10, 29]; the main ones adopted are *tuple agreement*, *Kendall tau distance*, and *Spearman's footrule*.

Given two top-k lists, *tuple agreement* is defined as the number of tuples that appear in both lists (i.e., it does not consider the actual ordering of the tuples). To allow comparisons across different values of k, we normalize the tuple agreement and call this value *Agreement*. Note that *higher is better for Agreement*, a value of 1 indicating lists with the same elements (perhaps in different order). The second measure is a variation of the *Kendall tau distance* for partial orders that computes the distance between two partial rankings based on the number of pairwise disagreements between them [10] — a disagreement receives a penalty of 1, while for agreement there is no penalty. In case the two top-k lists are permutations of each other, this is the number of exchanges required to convert one into the other. However, if a pair (i, j) appears in one list but not in the other, we do not know if the penalty should be 0 or 1; therefore, it takes a parameter p indicating how pessimistic the metric should be ($p = 1$ is most pessimistic). The measure is normalized by the square of the length of the union of the two lists, and this value is called *Kendall*$_p$. Note that *lower is better for Kendall*, since it measures differences instead of agreement. Finally, the third measure is a variation of the *Spearman's footrule* metric for partial orders that computes the distance between two partial rankings using the difference of positions of elements of one list in comparison with the other [10]. Where *Kendall* just counts number of swaps, this metric counts how far each element must be moved to reach the place occupied in the other list. This measure is normalized as before, and the result is called *Spearman*. Note that, like *Kendall*, *lower is better for Spearman*.

6.4.2 Results

We now discuss two research questions: "*Which aggregation method yields the best results?*" and "*How different are the results produced by each aggregation method?*". We discuss each of these questions in turn.

Question 1: "Which aggregation method yields the best results?" To answer this question, for each group, we computed a top-k query answer r_g, as well as a top-k query answer r_i for each user in the group. We then computed the value of each measure over r_g and r_i, and finally report the mean of all such results. We carried out three different comparisons: overall, varying group size, and varying the value of k — for the first, we report five different measures, while for the other two, we focus on *Agreement* and *Spearman* for reasons of space.



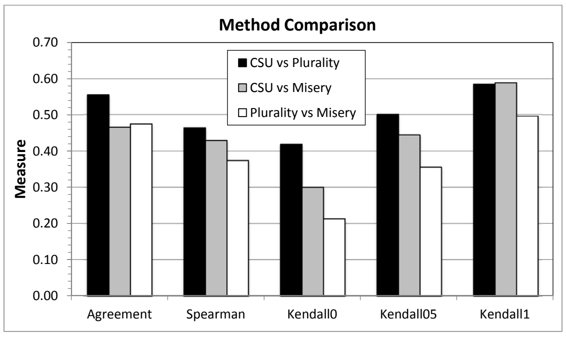
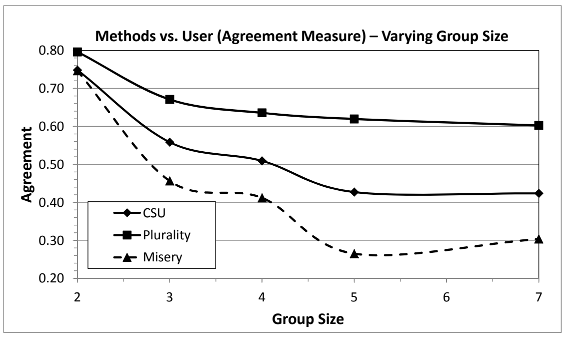


Figure 10: Comparison of the approaches with user preferences (left) and pairwise comparison of approaches (right). Higher is better for *Agreement*, while lower is better for the rest.



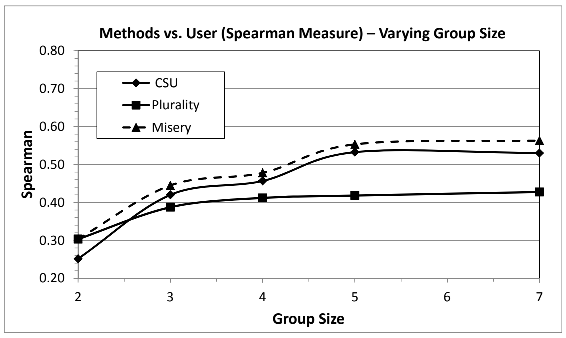
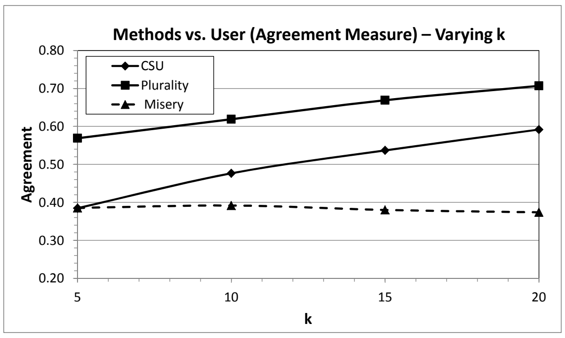


Figure 11: How Agreement (left, higher is better) and Spearman (right, lower is better) change depending on group size.



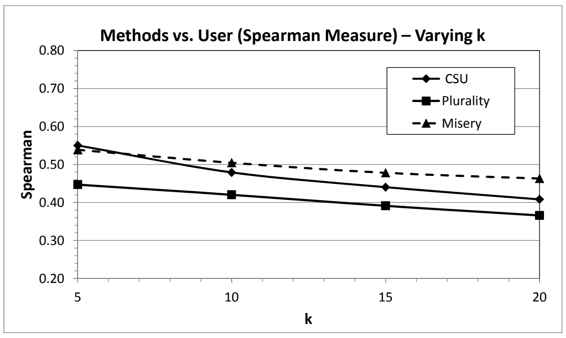


Figure 12: How Agreement (left, higher is better) and Spearman (right, lower is better) change depending on the value of k.

Overall comparison. Figure 10 (left) shows an overall comparison between each aggregation method and the users' rankings. Here, we see that plurality voting performed best in our experiments both in terms of the agreement between the desires of each individual user and what the aggregated result yields (*Agreeement* measure), as well as in terms of the ordering of the results (the rest of the measures). Plurality with misery was the worst performer, both in terms of agreement and ordering of the top-k. This is likely due to the fact that this method empowers users to veto options that could be ranked high by several others. Another problem could arise due to cycles; for efficiency, our implementation of *removeCycles* iteratively chooses a random edge from a cycle and removes it without taking into consideration the effect of this operation on the opinions of the other members. Developing a better way to address cycles is an interesting topic of future research.

Varying group size. Figure 11 shows the results of experiments varying the group size between 2 and 7 (*Agreement* measure on the left, *Spearman* on the right) — we eliminated group sizes for which we did not have enough data. Results show that all aggregation methods tend to worsen with both measures as group size increases — this is consistent with the fact that groups of larger size are more difficult to satisfy when k is fixed since it is more likely that conflicting preferences exist.

Varying value of k. Figure 12 shows our results when varying k between 5 and 20 — as before, we have results for the *Agreement* measure on the left and for *Spearman* on the right. Opposite to what was observed when varying group size, we see that increasing the value of k affords an increase of performance for all aggregation methods over both measures. One possible exception is misery with respect to the *Agreement* measure, which seems to slightly worsen as k increases – this special behavior is likely a consequence of the same issues discussed above. This overall tendency to perform better as k increases makes sense for a group of fixed size, since more space becomes available in the result to accommodate potentially conflicting user preferences.

Question 2: "How different are the results produced by each aggregation method?" Figure 10 (right) shows an overall comparison between the results produced by each of our aggregation methods. Focusing on plurality (the overall best performer in the experiments for Question 1), we now compare how similar CSU and misery are to plurality. We can see that with respect to the *Agreement* measure, plurality and CSU are the most similar for these runs; interestingly, for the rest of the measures, the most similar to plurality was misery. Figures 17 and 18 in Appendix C complement these results — the former shows the same behavior when varying k for *Agreement*, though for *Spearman*, CSU turns out to be a little better than misery. The latter experiment, however, returns to the same pattern with respect to variations in group size, and CSU is most similar to plurality relative to *Agreement*, while misery is most similar relative to *Spearman* (though the differences are not large at the highest setting of the parameter).

These results suggest that CSU is good at getting the right elements in the result, but misery is good at getting the order right (using the results obtained by plurality as a baseline).

7 Discussion

In this paper, we have proposed a two-fold extension of the Datalog+/– ontology language: allowing for partially ordered preferences of groups of users, and the management of probabilistic uncertainty. To our knowledge, this is the first combination of ontology languages with group preferences. We focused on answering k-rank queries in this context, and studied different approaches to computing answers to queries based on group preferences; the main challenge is to find a principled way in which to combine the preferences of each individual in the group, while also taking into account the preferences induced by the probabilities obtained from the uncertainty model. We then studied algorithms to answer k-rank queries for disjunctions of atomic queries under these conditions, and showed that it can be done in polynomial time in the data complexity, as long as query answering can also be done so in the underlying classical ontology.

Finally, we presented a prototype implementation of our framework, including an empirical analysis using real-world data and preference models obtained from real users, showing that our approach is feasible in practice.

An interesting question that comes up when implementing these techniques in a real-world scenario is how to decide which aggregation technique to use — this will indeed have an impact on how group decisions are made, as seen in Section 6. Some guidelines regarding this question for the techniques studied here are the following: (i) CSU is a fine-grained approach that, given its consideration of all pairs, allows for a better leveraging of user preference information; (ii) on the other hand, the voting-based approach taken in this paper computes k-rankings as a first step, which may end up containing arbitrary elements. For instance, coming back to the friends in Example 1, consider the case in which $k = 2$ and Alberto's preferences yields ranking $\langle r_1, r_3 \rangle$, Bruno's yields $\langle r_2, r_3 \rangle$, and Charles's $\langle r_1, r_2 \rangle$ — note that Charles is the only one who actually cares about the position of the second element, while the answers of the other two were completed arbitrarily after the first position. The *misery* and *fairness* techniques can help mitigate these issues, but in general they can still arise. On the other hand, CSU has the weakness of only working on partial graphs — in carrying out our experiments with real users, we found that people do not find partial orderings of elements natural (most questions were variations of "Do I have to fill in all possible pairs?"). Thus, the voting-based approaches have the advantage of simplicity, since it is possible to directly request a linear ordering of elements from each user, since this is the first step in the aggregation anyway.

Regarding the limitations of our work, perhaps the most evident is the fact that the only information being leveraged when computing answers to k-rank queries are the preferences of the users; this affords a lightweight query answering framework that requires a relatively minimal amount of input from the user. The weakness, however, is that cases may arise in which users pose completely opposite preferences leading to a tie which cannot be broken without making an arbitrary choice. Some of the suggestions discussed below on how to continue this research address this issue directly. Also, from our quality evaluation it seems that plurality is the most similar to what individual users want; still, many times users are willing to make compromises depending on other group members and our current approach does not contemplate this.

Topics for future work include further exploration of the similarity of preference aggregation and merging strategies to human judgment; this will shed light on how well-suited each of them is as a general aggregation strategy for search and query answering in the Social Semantic Web. Related to this effort, and to the limitations mentioned above, our experimental evaluation shows that new methods for measuring user satisfaction within a group should be developed, perhaps based on the difference between least and most satisfied users, and incorporated into our framework. Another interesting vein for future development is deriving explanations along with the answers to queries — in social situations, it is likely that users' satisfaction with the answer is tied to how it was produced; for instance, if close friends heavily influenced the result the user will probably respond better than if the result was influenced by strangers. Towards this end, we propose to explore the use of provenance models [26] as well as approaches based on argumentation [32], among others, in our framework. Finally, another interesting social aspect to work on is that of contemplating "past footprints" of group decisions as part of further information that can be leveraged when answering queries over closely related groups — in this paper, we assumed a one-shot process where this is not applicable, but this kind of information could be valuable, for instance, in enforcing another form of fairness since individuals favored in the past can be guaranteed to have less weight in future decisions.

References

[1] Michael Ackerman, Sul-Young Choi, Peter Coughlin, Eric Gottlieb, and Japheth Wood. Elections with partially ordered preferences. *Public Choice*, 157(1/2):145–168, 2012.

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

[3] C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *Proc. ICALP*, volume 115 of *LNCS*, pages 73–85. Springer, 1981.

[4] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. of ICDE*, pages 421–430. IEEE Computer Society, 2001.

[5] R.I. Brafman and C. Domshlak. Preference handling — An introductory tutorial. *AI Mag.*, 30(1):58–86, 2009.

[6] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *Proc. KR*, pages 70–80. AAAI Press, 2008.

[7] A. Calì, G. Gottlob, and T. Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.

[8] A. Calì, G. Gottlob, and A. Pieris. Query answering under non-guarded rules in Datalog+/–. In *Proc. of RR*, volume 6333 of *LNCS*, pages 1–17. Springer, 2010.

[9] J. Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, 2003.

[10] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. *SIAM J. Discrete Math.*, 17(1):134–160, 2003.

[11] M. Finger, R. Wassermann, and F.G. Cozman. Satisfiability in $\mathcal{EL}$ with sets of probabilistic ABoxes. In *Proc. of DL*, 2011.

[12] W. Gaertner. *A Primer in Social Choice Theory: Revised Edition*. Oxford University Press, 2009.

[13] M. Gartrell, X. Xing, Q. Lv, A. Beach, R. Han, S. Mishra, and K. Seada. Enhancing group recommendation by incorporating social relationship interactions. In *Proc. of GROUP*, pages 97–106. ACM Press, 2010.

[14] G. Gottlob, T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Query answering under probabilistic uncertainty in Datalog+/– ontologies. *Ann. Math. Artif. Intell.*, 69(1):37–72, 2013.

[15] G. Gottlob, T. Lukasiewicz, and G. I. Simari. Answering threshold queries in probabilistic Datalog+/– ontologies. In *Proceedings SUM*, volume 6929 of *LNCS*, pages 401–414. Springer, 2011.

[16] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *Proc. of ICDE*, pages 2–13. IEEE Computer Society, 2011.

[17] J. C. Jung and C. Lutz. Ontology-based access to probabilistic data with OWL QL. In *Proc. of ISWC*, volume 7649 of *LNCS*, pages 182–197. Springer, 2012.

[18] J. Lang, M. S. Pini, F. Rossi, D. Salvagnin, K. B. Venable, and T. Walsh. Winner determination in voting trees with incomplete preferences and weighted votes. *Auton. Agent. Multi-Agent Syst.*, 25(1):130–157, 2012.

[19] G. Linden, B. Smith, and J. York. Industry report: Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[20] T. Lukasiewicz, M. V. Martinez, G. Orsi, and G. I. Simari. Heuristic ranking in tightly coupled probabilistic description logics. In *Proc. of UAI*, pages 554–563. AUAI, 2012.

[21] T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Preference-based query answering in Datalog+/– ontologies. In *Proc. of IJCAI*, pages 1017–1023. IJCAI/AAAI, 2013.

[22] T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Preference-based query answering in probabilistic Datalog+/– ontologies. In *Proc. of ODBASE*, volume 8185 of *LNCS*, pages 501–518. Springer, 2013.

[23] T. Lukasiewicz, M. V. Martinez, G. I. Simari, and O. Tifrea-Marciuska. Group preferences for query answering in Datalog+/– ontologies. In *Proc. of SUM*, volume 8078 of *LNCS*, pages 360–373. Springer, 2013.

[24] T. Lukasiewicz, M. V. Martinez, G. I. Simari, and O. Tifrea-Marciuska. Query answering in probabilistic Datalog+/– ontologies under group preferences. In *Proc. of WI*, pages 171–178. IEEE Computer Society, 2013.

[25] M. Manoj and Elizabeth Jacob. Information retrieval on internet using meta-search engines: A review. *J. Sci. Ind. Res.*, 67(10):739–746, 2008.

[26] U. Marjit, K. Sharma, and U. Biswas. Provenance representation and storage techniques in linked data: A state-of-the-art survey. *Int. J. Comput. Appl.*, 38(9):23–28, 2012.

[27] J. Masthoff. Group modeling: Selecting a sequence of television items to suit a group of viewers. *User Model. User-Adapt. Interact.*, 14(1):37–85, 2004.

[28] J. Noessner and M. Niepert. ELOG: A probabilistic reasoner for OWL EL. In *Proc. of RR*, volume 6902 of *LNCS*, pages 281–286. Springer, 2011.

[29] E. Ntoutsi, K. Stefanidis, K. Nørvåg, and H.-P. Kriegel. Fast group recommendations by applying user clustering. In *Proc. of ER*, volume 7532 of *LNCS*, pages 126–140. Springer, 2012.

[30] P. K. Pattanaik. *Voting and Collective Choice: Some Aspects of the Theory of Group Decision-Making*. Cambridge University Press, 1971.

[31] M. S. Pini, F. Rossi, K. B. Venable, and T. Walsh. Aggregating partially ordered preferences. *J. Log. Comput.*, 19(3):475–502, 2009.

[32] I. Rahwan and G. R. Simari. *Argumentation in Artificial Intelligence*. Springer, 1st edition, 2009.

[33] K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM T. Database Syst.*, 36(3):19:1–19:45, 2011.

[34] A. D. Taylor. *Social Choice and the Mathematics of Manipulation*. Cambridge University Press, 2005.

[35] M. Wooldridge. *An Introduction to Multiagent Systems*. Wiley, 2009.

[36] Yelp. Yelp Dataset Challenge, 2012.

[37] X. Zhang and J. Chomicki. Preference queries over sets. In *Proc. of ICDE*, pages 1019–1030. IEEE Computer Society, 2011.

A The Chase

The *chase* was first introduced to enable checking implication of dependencies, and later also for checking query containment. By "chase", we refer both to the chase procedure and to its output. The TGD chase works on a database via so-called TGD *chase rules* (see [7] for an extended chase with also EGD chase rules).

TGD Chase Rule. Let D be a database, and σ be a TGD of the form $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$. Then, σ is *applicable* to D iff there exists a homomorphism h that maps the atoms of $\Phi(\mathbf{X}, \mathbf{Y})$ to atoms of D. Let σ be applicable to D, and h_1 be a homomorphism that extends h as follows: for each $X_i \in \mathbf{X}$, $h_1(X_i) = h(X_i)$; for each $Z_j \in \mathbf{Z}$, $h_1(Z_j) = z_j$, where z_j is a "fresh" null, i.e., $z_j \in \Delta_N$, z_j does not occur in D, and z_j lexicographically follows all other nulls already introduced. The *application of* σ on D adds to D the atom $h_1(\Psi(\mathbf{X}, \mathbf{Z}))$ if not already in D.

The chase algorithm for a database D and a set of TGDs Σ consists of an exhaustive application of the TGD chase rule in a breadth-first (level-saturating) fashion, which outputs a (possibly infinite) chase for D and Σ. Formally, the *chase of level up to* 0 of D relative to Σ, denoted $\mathit{chase}^0(D, \Sigma)$, is defined as D, assigning to every atom in D the *(derivation) level* 0. For every $k \geqslant 1$, the *chase of level up to* k of D relative to Σ, denoted $\mathit{chase}^k(D, \Sigma)$, is constructed as follows: let $I_1, \ldots, I_n$ be all possible images of bodies of TGDs in Σ relative to some homomorphism such that (i) $I_1, \ldots, I_n \subseteq \mathit{chase}^{k-1}(D, \Sigma)$ and (ii) the highest level of an atom in every I_i is $k-1$; then, perform every corresponding TGD application on $\mathit{chase}^{k-1}(D, \Sigma)$, choosing the applied TGDs and homomorphisms in a (fixed) linear and lexicographic order, respectively, and assigning to every new atom the *(derivation) level* k. The *chase* of D relative to Σ, denoted $\mathit{chase}(D, \Sigma)$, is defined as the limit of $\mathit{chase}^k(D, \Sigma)$ for $k \rightarrow \infty$.

B Proofs

Proof of Proposition 1. Let $\succ^* = \otimes_t(M, \succ_U)$. We now prove that (i) $\succ^*$ is an SPO, and (ii) if $a_1 \succ_U a_2$ and $a_1 \succ_M a_2$, then $a_1 \succ^* a_2$.

(i) We must show that $\succ^*$ is irreflexive and transitive. Let $\succ'$ be the intermediate result of the $\otimes_t$ operator just before computing the transitive closure. By assumption, $\succ_U$ is irreflexive, and (as $\succ'$ cannot contain any new self-edges) this property is preserved in $\succ'$. By construction, $\succ^*$ is the transitive closure of $\succ'$. Since this operation does not add cycles, $\succ^*$ is also irreflexive, in addition to clearly being transitive. Overall, $\succ^*$ is an SPO.

(ii) A necessary condition for $\otimes_t$ to change the order of a pair in $\succ_U$ is that the pair in $\succ_M$ be reversed. Since by hypothesis this is not the case, the statement follows. □

Proof of Proposition 2. If $t = 0$, then we check whether there are pairs (a, b) in $\succ_U$ with $\Pr_M(b) - \Pr_M(a) > t$ relative to $\succ_M$. Since by assumption, $\succ_M = \succ_{M'}$, it must be the case that $\Pr_M(b) - \Pr_M(a) > t$ relative to $\succ_M$ iff this is the case relative to $\succ_{M'}$, and thus the outputs in both cases must be identical. □

Proof of Theorem 3. (i) By hypothesis, $\succ^*$ is transitive and cycle-free — the former is assumed by hypothesis and the latter is ensured by *removeCycles*. Therefore, $\succ^*$ is irreflexive and antisymmetric, and thus an SPO.

(ii) By Proposition 1, for $1 \leqslant i \leqslant n$, we know that $\otimes_{t_i}(M, \succ_{U_i})$ are all SPOs and therefore cycle-free. Towards a contradiction, suppose that $a_1 \not\succ^* a_2$; by the construction of the graph G in AggPrefsCSU and the hypothesis that $(a_1, a_2) \in \otimes_{t_i}(M, \succ_{U_i})$ for all $1 \leqslant i \leqslant n$, this can only happen if (a_1, a_2) belongs to a cycle, and the algorithm chose to remove this cycle by deleting this edge. Since the label of (a_1, a_2) is n and, by the assumption of how *removeCycles* works, the entire cycle is composed of edges labeled with n; but this contradicts $\otimes_{t_i}(M, \succ_{U_i})$ being SPOs. □

Proof of Theorem 4. First of all, we show that the computation of $\succ^*$ in Line 2 of k-Rank-CSU is such that $\succ^* = \biguplus(\otimes_{t_1}(M, \succ_{U_1}), \ldots, \otimes_{t_n}(M, \succ_{U_n}))$. The AggPrefsCSU algorithm iterates through all the users u and directly applies the definition of the $\otimes_{t_i}$ operator to produce the *currUserG* graph. Once this graph is computed, the algorithm updates graph G and its labels accordingly — the edge labels after the final iteration of the for-loop in Line 3 correspond to the number of users that have that edge in their preference relation after combining it with the probabilistic one according to t (the last computation steps compute the transitive closure and remove cycles).

Now, correctness is a consequence of the direct application of the definition of k-rank: the while-loop in Line 5 iteratively computes the skyline answers to Q by means of a subroutine, adds these results to the output in arbitrary order, and removes them from consideration. Line 10 ensures that at most k results are returned.

Finally, we show that k-Rank-CSU runs in $O(\mathit{poly}(\|D\|) \cdot S + C)$ time in the data complexity. As O is assumed to be tractable, the (necessary finite portion of the) chase of O relative to Q can be computed in polynomial time in the data complexity [7]. The $\succ^*$ relation is computed in time $O(\|\succ_U\| \cdot S + C)$, where $\|\succ_U\|$ is a polynomial in $\|D\|$. Now, *computeSkyline*$(KB, Q, \succ^*)$ can be computed in polynomial time by a linear-time scan of the chase structure for Q (of polynomial size, by hypothesis), and the results can be removed by another such scan. □

Proof of Proposition 6. Results for *Unanimous Winner* (UW)

- *Plurality* satisfies UW.

  Proof: If everyone in a group of n users has element a in a k-rank answer, then element a has n votes. The only way that no ranking with element a exists is that there are k elements with at least $n + 1$ votes, which is impossible. □

- *Fairness* does not satisfy UW.

  Proof by counterexample: Suppose we have two group preference models $\mathcal{P}_1 = \langle U_1 \rangle$ and $P_2 = \langle U_2 \rangle$, a query Q, and the following are k-rank answers to Q for each of them with $k = 3$:

$\mathcal{P}_1$:

Element
b
a
c

$\mathcal{P}_2$:

Element
d
e
c

Element c belongs to a 3-rank answer to Q for $\mathcal{P}_1$ and $\mathcal{P}_2$; however, no matter which order we take between $\mathcal{U}_1$ and $\mathcal{U}_2$, no 3-rank answer to Q for $\mathcal{P}_1 \cup \mathcal{P}_2$ contains element c.

- *Plurality with misery* satisfies UW.

  Proof: Analogous to the proof for *plurality*; note that if the an element is a unanimous winner then it cannot be a misery element. □

- *Fairness with misery* does not satisfy UW.

  Proof by counterexample: Consider the same counterexample for the proof that *fairness* does not satisfy UW.

Results for *Unanimous Loser* (UL):

- *Plurality* satisfies UL.

  Proof: If element a is in none of the individual k-rank answers, then it has zero votes in the final k-rank answer. By hypothesis, there are at least k elements in the union of all the k-rank answers that have non-zero votes, which means that they will beat element a in any final ranking. □

- *Fairness* satisfies UL.

  Proof: If the element does not appear in any k-rank answer, it is impossible for it to be chosen for the final one. □

- *Plurality with misery* satisfies UL.

  Proof: Analogous to *plurality* — it does not matter if the element is a misery element or not. □

- *Fairness with misery* satisfies UL.

  Proof: If the element does not appear in any k-rank answer, it is impossible for it to be chosen for the final one; note that it does not matter if the element is a misery element or not. □

Let *rank*$(a, \succ, Q, KB)$ be defined as follows: *rank*$(a, \succ, Q, KB) = 1$ iff a is a skyline answer to Q for $\succ$, and *rank*$(a, \succ, Q, KB) = k + 1$ iff a is a skyline answer to Q for $\succ$ after removing from consideration all elements b such that *rank*$(b, \succ, Q, KB) = k$.

Lemma 8 *Let $\succ$ be an SPO, $KB = (O, \mathcal{U}, M, \otimes, \uplus)$ be a GPP-Datalog+/– ontology, and Q be a DAQ. If $\succ_{-a}$ is an SPO that results from removing element a from $\succ$, then for every element b in $\succ$ we have that either rank$(b, \succ_{-a}, Q, KB) =$ rank$(b, \succ, Q, KB)$ or rank$(b, \succ_{-a}, Q, KB) =$ rank$(b, \succ, Q, KB) - 1$. Also, If $\succ_{+a}$ is an SPO that results from adding an element a to $\succ$, then for every element b in $\succ$ we have that either rank$(b, \succ_{+a}, Q, KB) =$ rank$(b, \succ, Q, KB)$ or rank$(b, \succ_{+a}, Q, KB) =$ rank$(b, \succ, Q, KB) + 1$.*

Proof: By direct consequence of the definition of *rank* (above), we have that an element's rank is equal to the length of the longest path from any element in the skyline to that element. Therefore, adding an element can only increase such length by one. On the other hand, removing an element can only decrease the length by one, given that the transitivity property holds. □

Results for *Weak Stability* (WS):

- *Plurality* satisfies WS.

  Proof sketch: By Lemma 8, adding an element a to choose from can at most have the effect that each user now incorporates a into their k-rank answers, and so no longer vote their previously least preferred of the k — this only benefits element a, and no other. Thus, at group level, this can only make a be in a k-rank answer and shift the rest of the elements down one place, as WS requires. □

- *Fairness* satisfies WS.

  Proof: By Lemma 8 we have that the added element can appear in each individual k-rank answer, but the rest of the elements maintain their original relative ordering. Therefore, if the added element ranks high enough to be chosen, the end result will differ by that element only. Otherwise, the result will be the same. □

- *Plurality with misery* satisfies WS.

  Proof: If the element added is a misery element, the result will be the same. Otherwise, the same argument used for *plurality* applies. □

- *Fairness with misery* satisfies WS.

  Proof: If the element added is a misery element, the result will be the same. Otherwise, the same argument used for *fairness* applies. □

Results for *Stability 1* (S1):

- *Plurality* satisfies *S1*.

  Proof sketch: By the same argument made for *plurality* in WS, the new element a might get enough votes to make it into position i of the final k-rank answer; these votes were previously held by other elements. Since the votes lost by the rest of the elements only go to element a, a final k-rank answer exists that maintains the relative order given by the original rank. □

- *Fairness* satisfies S1.

  Proof: Analogous to the argument for WS. □

- *Plurality with misery* satisfies S1.

  Proof: If the element is a misery element, the result will be the same. Otherwise, the proof is analogous to that for *plurality*. □

- *Fairness with misery* satisfies S1.

  Proof: If the element is a misery element, the result will be the same. Otherwise, the proof is analogous to that of WS for *fairness*. □

Stability 2 (S2):

- *Plurality* satisfies S2.

  Proof: Using Lemma 8, we can ensure that there exists a k-rank answer that is the same as the original. The removed element can never have enough votes to beat the last one in the original ranking, since otherwise it would have beaten it in the first place. Therefore, whatever element (or elements) comes in to replace it will be in the same conditions. □

- *Fairness* satisfies S2.

  Proof: By Lemma 8, we have that when removing an element not in the a k-rank answer, the ranks of those in the final k-rank answer cannot improve nor decrease. Therefore, the result will be the same. □

- *Plurality with misery* satisfies S2.

  Proof: If the element is a misery element, the result will be the same. Otherwise, the proof is analogous to that for *plurality*. □

- *Fairness with misery* satisfies S2. □

  Proof: If the element is a misery element, the result will be the same. Otherwise, it is analogous to *fairness*.

Results for *Monotonicity 1 (M1)*:

- *Plurality* satisfies M1.

  Proof: Trivial (the element can only gain votes). □

- *Fairness* satisfies M1.

  Proof: If the element was chosen, improving its position in some rankings can only help it to be chosen earlier. □

- *Plurality with misery* satisfies M1.

  Proof: Analogous to the proof for *plurality*, since if the element is chosen, it cannot be a misery element. □

- *Fairness with misery* satisfies M1.

  Proof: Same argument as for*fairness*; note that if the element is chosen, it cannot be a misery element. □

Monotonicity 2 (M2):

- *Plurality* satisfies M2.

  Proof: Trivial (the element can only lose votes). □

- *Fairness* satisfies M2.

  Proof: If an element is not chosen, ranking it even lower cannot cause it to be chosen. □

- *Plurality with misery* satisfies M2.

  Proof: If the element is a misery element, the result will be the same. Otherwise, the proof is analogous to that for *plurality*. □

- *Fairness with misery* satisfies M2.

  Proof: Same argument as for *fairness*; note that if the element is a misery element, the result will be the same. □

Non-dictatorship (ND):

- *Plurality* satisfies *ND*.

  Proof: One user can only influence the choice by adding k votes (one to each element in their ranking). Therefore, a single user cannot dictate the outcome of the result. □

- *Fairness* satisfies ND for any $k \neq 1$.

  Proof: If $k = 1$, the first user to choose is the dictator. Otherwise, the result is determined by at least two users. □

- *Plurality with misery* satisfies ND.

  Proof: Same argument as that of *plurality*. □

- *Fairness with misery* satisfies ND.

  Proof: Though the scenario for $k = 1$ could cause the first user to choose to be a dictator, that user's choice might be in another user's misery list. □

Proof of Proposition 7.

- *CSU* satisfies GUPF.

  Proof: If element a is in the first position of a k-rank answer to both $\mathcal{P}_A$ and $\mathcal{P}_B$, then it belongs to the skylines of both $\mathcal{P}_A$ and $\mathcal{P}_B$, which implies that it also belongs to the skyline of $\mathcal{P}_{A \cup B}$. Therefore, there exists a k-rank answer to Q for $\mathcal{P}$ that contains a in the first position. □

- *CSU* satisfies UWF.

  Proof: It is a direct consequence of the fact that *CSU* satisfies GUPF. □

- *CSU* satisfies UP if *removeCycles* only removes edges (v_1, v_2) whenever there does not exist another edge in the cycle labeled with a lower number.

  Proof: Follows directly from condition (ii) in Theorem 3. Note that if there is a cycle, there must exist an edge in the cycle with a lower number of votes than the removed edge. □

C Further Details for the Experimental Evaluation

A high-level overview of the architecture of our system can be found in Figure 13. The input consists of a tuple of the form $\langle Q, O, \mathcal{P}, k\rangle$ with $O = (D, \Sigma)$, where Q is a query, D is a database, Σ is a finite set of TGDs, $\mathcal{P}$ is a group preference model, and k is the number of query results that we are interested in. The set of TGDs Σ for the ontology in our experiments is given in Figure 15, while the structure of the database for this ontology is shown in Figure 16. Note that this ontology is different from the one that we used in the examples throughout the paper — we have based our examples on this domain, but have rewritten the ontology for a greater readability.

The constraint and query managers and the rewriting engine are directly taken from the Datalog+/– system of [16]: the constraint manager checks unions of conjunctive queries that are used to check if D satisfies Σ; the query manager schedules queries for rewriting and execution; finally, the rewriting engine converts the input query into a union of conjunctive queries after all the TGDs are applied. The list of queries obtained after the rewriting step is the input for the CSU-PrefChase subsystem, which constructs the preferences over the atoms that are answers to this query.

The CSU-PrefChase builds a graph that stores all the preferences of the users after the standard chase is computed. The preference graph is a labeled directed graph (N, E, ℓ), where N is the node set (the atoms), E is the edge set (the preference relation), and the labeling function ℓ stores for each edge the identity of its user. That is, edges in E are pairs of nodes (u, v), which state that u is preferred over v by the user specified in the label $\ell(u, v)$. Note that CSU-PrefChase is used not only for the CSU strategy, but also for the plurality with misery and fairness with misery strategies — the latter strategies involve calling CSU-PrefChase individually for each user to obtain the result of merging the users' SPOs with the one derived from the probabilistic model.

Additional Results. Figures 17 and 18 contain results that complement our analysis in Section 6.4 (Question 2, Page 21).

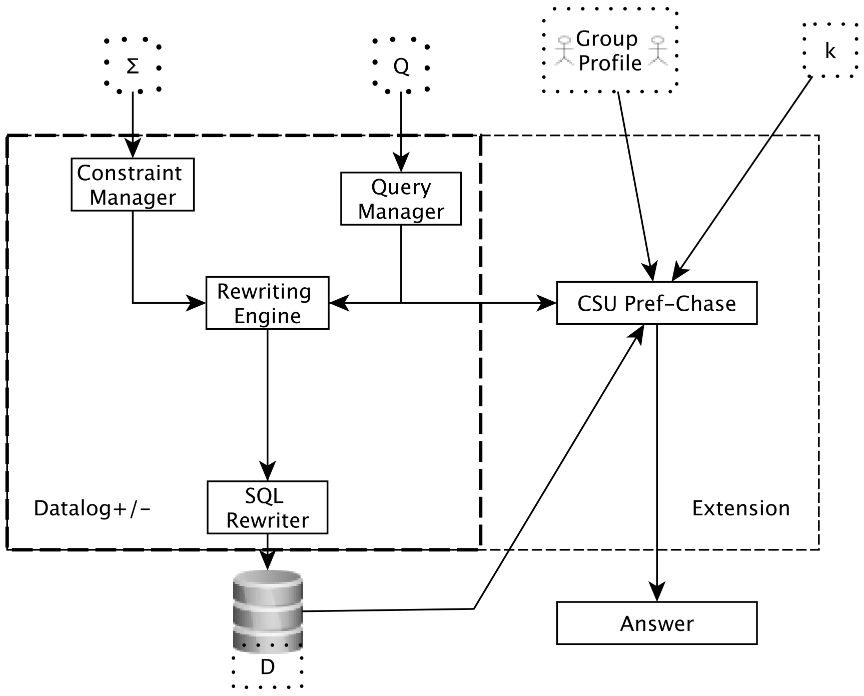


Figure 13: The architecture of our system.

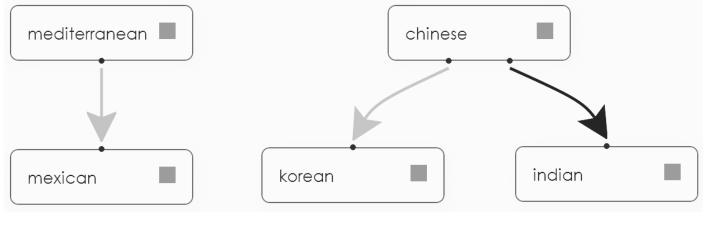


Figure 14: Preferences of a user for dinner (e.g., prefers Mediterranean food over Mexican food)

$$
\begin{array}{lll}
\Sigma = \{ & \mathit{barbecue}(X) \rightarrow \mathit{food}(X), & \mathit{local\_flavor}(X) \rightarrow \mathit{food}(X), \\
& \mathit{hot\_dogs}(X) \rightarrow \mathit{food}(X), & \mathit{vegetarian}(X) \rightarrow \mathit{food}(X), \\
& \mathit{seafood}(X) \rightarrow \mathit{food}(X), & \mathit{chicken\_wings}(X) \rightarrow \mathit{food}(X), \\
& \mathit{pizza}(X) \rightarrow \mathit{food}(X), & \mathit{dessert}(X) \rightarrow \mathit{food}(X), \\
& \mathit{sandwiches}(X) \rightarrow \mathit{food}(X), & \mathit{gluten\_free}(X) \rightarrow \mathit{food}(X), \\
& \mathit{burgers}(X) \rightarrow \mathit{food}(X), & \mathit{fast\_food}(X) \rightarrow \mathit{food}(X), \\
& \mathit{donuts}(X) \rightarrow \mathit{dessert}(X), & \mathit{bagels}(X) \rightarrow \mathit{dessert}(X), \\
& \mathit{ice\_cream}(X) \rightarrow \mathit{dessert}(X), & \mathit{dance\_clubs}(X) \rightarrow \mathit{place}(X), \\
& \mathit{juice\_bars\_and\_smoothies}(X) \rightarrow \mathit{place}(X), & \mathit{cafes}(X) \rightarrow \mathit{place}(X), \\
& \mathit{bakeries}(X) \rightarrow \mathit{place}(X), & \mathit{breweries}(X) \rightarrow \mathit{place}(X), \\
& \mathit{steakhouses}(X) \rightarrow \mathit{place}(X), & \mathit{lounges}(X) \rightarrow \mathit{place}(X), \\
& \mathit{buffets}(X) \rightarrow \mathit{place}(X), & \mathit{bars}(X) \rightarrow \mathit{place}(X), \\
& \mathit{dive\_bars}(X) \rightarrow \mathit{bars}(X), & \mathit{pubs}(X) \rightarrow \mathit{bars}(X), \\
& \mathit{sports\_bars}(X) \rightarrow \mathit{bars}(X), & \mathit{sushi\_bars}(X) \rightarrow \mathit{bars}(X), \\
& \mathit{wine\_bars}(X) \rightarrow \mathit{bars}(X), & \mathit{mexican}(X) \rightarrow \mathit{cuisine}(X), \\
& \mathit{american}(X) \rightarrow \mathit{cuisine}(X), & \mathit{local\_services}(X) \rightarrow \mathit{cuisine}(X), \\
& \mathit{asian}(X) \rightarrow \mathit{cuisine}(X), & \mathit{middle\_eastern}(X) \rightarrow \mathit{cuisine}(X), \\
& \mathit{mediterranean}(X) \rightarrow \mathit{cuisine}(X), & \mathit{chinese}(X) \rightarrow \mathit{asian}(X), \\
& \mathit{indian}(X) \rightarrow \mathit{asian}(X), & \mathit{thai}(X) \rightarrow \mathit{asian}(X), \\
& \mathit{korean}(X) \rightarrow \mathit{asian}(X), & \mathit{japanese}(X) \rightarrow \mathit{asian}(X), \\
& \mathit{vietnamese}(X) \rightarrow \mathit{asian}(X), & \mathit{greek}(X) \rightarrow \mathit{mediterranean}(X), \\
& \mathit{italian}(X) \rightarrow \mathit{mediterranean}(X), & \mathit{french}(X) \rightarrow \mathit{mediterranean}(X), \\
& \mathit{food}(X) \rightarrow \exists Y\, \mathit{hasName}(X, Y), & \mathit{food}(X) \rightarrow \exists Y\, \mathit{inCity}(X, Y), \\
& \mathit{food}(X) \rightarrow \exists Y\, \mathit{inState}(X, Y), & \mathit{cuisine}(X) \rightarrow \exists Y\, \mathit{hasName}(X, Y), \\
& \mathit{cuisine}(X) \rightarrow \exists Y\, \mathit{inCity}(X, Y), & \mathit{cuisine}(X) \rightarrow \exists Y\, \mathit{inState}(X, Y), \\
& \mathit{place}(X) \rightarrow \exists Y\, \mathit{hasName}(X, Y), & \mathit{place}(X) \rightarrow \exists Y\, \mathit{inCity}(X, Y), \\
& \mathit{place}(X) \rightarrow \exists Y\, \mathit{inState}(X, Y)\} &
\end{array}
$$

Figure 15: Set of TGDs for the ontology used in the experimental evaluation.

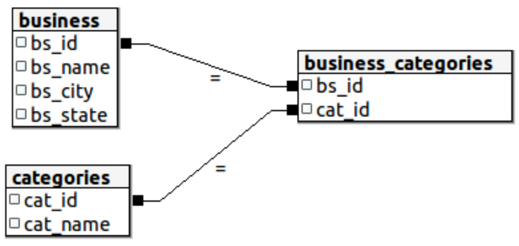
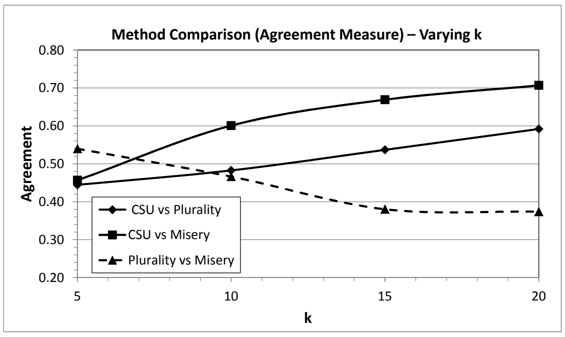


Figure 16: The database structure of the Datalog+/– ontology.



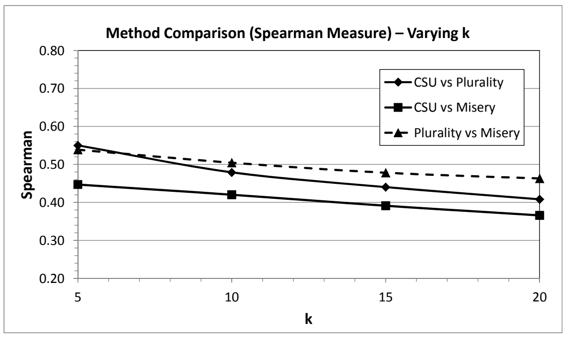
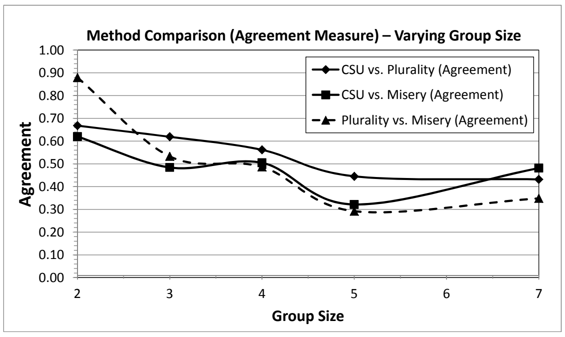


Figure 17: How Agreement (left, higher is better) and Spearman (right, lower is better) change depending on the value of k.



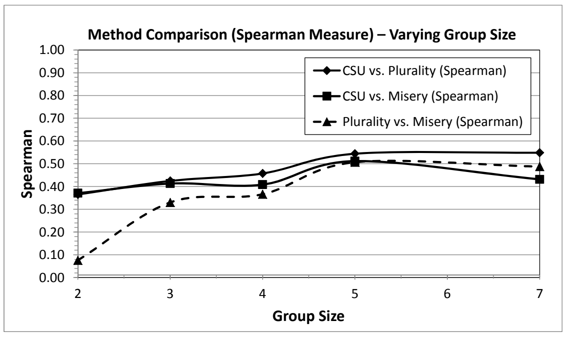


Figure 18: How Agreement (left, higher is better) and Spearman (right, lower is better) change depending on group size.