J Data Semant (2015) 4:81–101
DOI 10.1007/s13740-014-0040-x

ORIGINAL ARTICLE

Preference-Based Query Answering in Probabilistic Datalog+/– Ontologies

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

Received: 16 December 2013 / Accepted: 7 April 2014 / Published online: 22 May 2014

Abstract The incorporation of preferences into information systems, such as databases, has recently seen a surge in interest, mainly fueled by the revolution in Web data availability. Modeling the preferences of a user on the Web has also increasingly become appealing to many companies since the explosion of popularity of social media. The other surge in interest is in modeling uncertainty in these domains, since uncertainty can arise due to many uncontrollable factors. In this paper, we propose an extension of the Datalog+/– family of ontology languages with two models: one representing user preferences and one representing the (probabilistic) uncertainty with which inferences are made. Assuming that more probable answers are in general more preferable, one asks how to rank answers to a user's queries, since the preference model may be in conflict with the preferences induced by the probabilistic model—the need thus arises for preference combination operators. We propose four specific operators and study their semantic and computational properties. We also provide an algorithm for ranking answers based on the iteration of the well-known skyline answers to a query and show that, under certain conditions, it runs in polynomial time in the data complexity. Furthermore, we report on an implementation and experimental results.

T. Lukasiewicz (✉)· M. V. Martinez · G. I. Simari · O. Tifrea-Marciuska
Department of Computer Science, University of Oxford, Oxford, UK
e-mail: thomas.lukasiewicz@cs.ox.ac.uk

M. V. Martinez
e-mail: vanina.martinez@cs.ox.ac.uk

G. I. Simari
e-mail: gerardo.simari@cs.ox.ac.uk

O. Tifrea-Marciuska
e-mail: oana.tifrea@cs.ox.ac.uk

1 Introduction

There has recently been a marked push in the research and development of technology surrounding the Web and, perhaps most centrally, its vast repositories of data. Ontology and query languages are examples of such technology, used to share, integrate, and query large-scale and less structured data and knowledge bases, such as those occurring on the Web. One of the central issues in this domain is that Web search is still centered around the paradigm of receiving keywords from a user and returning a list of links to Web documents that are considered to be pertinent. *Semantic search*, on the other hand, has been proposed as an evolution of this paradigm that identifies objects, rather than whole documents, as candidates to answering the users' queries. At the same time, we have recently been witnessing another revolution in the rapid growth of what is generally referred to as the *Social Web* (which is often also implicitly connected to the Semantic Web [3]); the Social Web is centered around a (mostly) loosely coupled set of platforms that people make use of with the objective of sharing, viewing, and searching for information (in the form of pictures, text documents of varying lengths, videos, etc.) in a social and sometimes collaborative environment. The use of semantic search in the Social Web is of central importance, due to the missing link structure between Web pages, which is well-known from ranking (such as PageRank) in standard Web search. In addition, the fundamentally human component of these systems makes each user's *personal preferences* have a much more prevalent role than what was observed before this paradigm shift. Finally, the presence of uncertainty in the Web in general is undeniable [10,17,21,25]: information integration (as in travel sites that query multiple sources to find hotels and flights), automatic processing of Web data (analyzing an HTML document often involves uncertainty), as well as inherently uncertain data (such as user comments) are all examples of uncertainty that must be dealt with in answering queries in the Social Web. 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 preference and uncertainty management mechanisms by developing an extension of the Datalog+/– family of ontology languages [6] with a preference model over the consequences of the ontology, as well as a probabilistic model that assigns probabilities to them—note that all previous work on extending Datalog+/– with uncertainty (cf. [12] and references therein) does not deal with preferences. The former is assumed to model a user's (or group of users') preferences, while the latter is assumed to model the uncertainty in the domain. The specific mechanisms by which these models are given are left unspecified, since our focus is on the study of how to rank answers to a query when the two models may be in disagreement regarding the ranking: assuming that higher-probability consequences are more preferable than lower-probability ones, it is clear that such situations can arise.

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

- We introduce the PP-Datalog+/– framework, which combines ontology languages with both preferences as in relational databases and probabilistic uncertainty. To our knowledge, it is the first such extension.
- We formalize the concept of *preference combination* operators, which take as input a preference relation in the form of a strict partial order (SPO) and a score-based SPO (a weak order), and produce a new preference relation satisfying certain basic properties.
- We develop four specific preference combination algorithms, two egalitarian ones (ComPrefsGen and ComPrefsRank) and two user-biased ones (ComPrefsPT and ComPrefsSort). They satisfy the requirements of a preference combination operator as well as other desirable properties, among which are several postulates from the literature for a less general case. They can also be implemented to run in polynomial time in the data complexity, modulo the cost of computing probabilities.
- We give an algorithm for answering k-rank queries, a generalization of top-k queries based on the iterative computation of classical skyline answers, for disjunctions of atomic queries (DAQs), along with proofs of correctness and running time, showing that answering DAQs in PP-Datalog+/– is possible in polynomial time in the data complexity, modulo the cost of computing probabilities.
- We illustrate PP-Datalog+/– along a real-world application with preference-based query answering in the context of the Internet Movie Database (IMDB).
- We also report on an implementation of PP-Datalog+/– along with extensive experimental results, evaluating and analyzing the running time of our algorithms over a combination of real-world and synthetic data.

The rest of this paper is organized as follows. In Sect. 2, we present preliminary concepts on classical Datalog+/–. Section 3 introduces the general preference and probabilistic models that are used in this work, motivates the need to combine the user preferences with those induced by probability values by means of preference combination operators, and then goes on to present the syntax and semantics of PP-Datalog+/–. In Sects. 4 and 5, we present algorithms for the implementation of four preference combination operators and study their semantic and computational properties. Section 6 presents skyline and k-rank queries and a basic algorithm to answer them, and proves its correctness and running time, showing that under certain conditions, k-rank queries can be answered in polynomial time in the data complexity. In Sects. 7 and 8, we provide a real-world application and report on an implementation along with extensive experimental results, respectively. Finally, Sects. 9 and 10 discuss related work and conclude, respectively.

2 Preliminaries on Datalog+/–

In this section, we briefly recall some basics on Datalog+/– [6], namely, on relational databases, (Boolean) conjunctive queries ((B)CQs), tuple- and equality-generating dependencies (TGDs and EGDs, respectively), negative constraints, the chase procedure, and ontologies in Datalog+/–.

2.1 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 assume a lexicographic order on $\Delta \cup \Delta_N$, with every symbol in Δ_N following all symbols in Δ. We denote by $\mathbf{X}$ the 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*) $\mathbf{a}$ has the form $P(t_1, \ldots, t_n)$, where P is an n-ary predicate and $t_1, \ldots, t_n$ are terms. 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 Δ. 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 no 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 $\mu\colon \Delta \cup \Delta_N \cup \mathcal{V} \rightarrow \Delta \cup \Delta_N \cup \mathcal{V}$ 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 and sets/conjunctions of atoms. An *answer* to a CQ $Q(\mathbf{X}) = \exists \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y})$ over D, denoted $Q(D)$, is a tuple $\mathbf{t}$ over Δ for which there exists a homomorphism $\mu\colon \mathbf{X} \cup \mathbf{Y} \rightarrow \Delta \cup \Delta_N$ such that $\mu(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$ and $\mu(\mathbf{X}) = \mathbf{t}$. The *answer* to a BCQ $Q()$ over 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 $\forall \mathbf{X} \forall \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$, 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 is a homomorphism h that maps the atoms of $\Phi(\mathbf{X}, \mathbf{Y})$ to atoms of D, there is an extension h' of h that maps the atoms of $\Psi(\mathbf{X}, \mathbf{Z})$ to atoms of D. As 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. A TGD σ is *guarded* iff its body has an atom that contains all universally quantified variables of σ. The leftmost such atom is the *guard atom* (or *guard*) of σ.

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 [2], even when the schema and TGDs are fixed [5]. Decidability and tractability in the data complexity of query answering for the guarded case follow from a bounded tree-width property.

Negative constraints (or simply *constraints*) γ are first-order formulas of the form $\forall \mathbf{X}\, \Phi(\mathbf{X}) \rightarrow \bot$, 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 $\forall \mathbf{X}\, \Phi(\mathbf{X}) \rightarrow \bot$, 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.

Equality-generating dependencies (or *EGDs*) σ are first-order formulas $\forall \mathbf{X}\, \Phi(\mathbf{X}) \rightarrow X_i = X_j$, where $\Phi(\mathbf{X})$, called the *body* of σ, denoted $body(\sigma)$, is a conjunction of atoms (without nulls), and X_i and X_j are variables from $\mathbf{X}$. Such σ is satisfied in a database D for $\mathcal{R}$ iff, whenever there is a homomorphism h such that $h(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$, it holds that $h(X_i) = h(X_j)$. Adding EGDs over databases with TGDs along with negative constraints does not increase the complexity of BCQ query answering as long as they are *non-conflicting* [6]. Intuitively, this ensures that if the chase (see below) fails (due to strong violations of EGDs), then it already fails on the database D; and if it does not fail, then whenever "new" atoms (from the logical point of view) are created in the chase by the application of the EGD chase rule, atoms that are logically equivalent to the new ones are guaranteed to be generated also in the absence of the EGDs, guaranteeing that the EGDs do not influence the chase with respect to query answering.

We usually omit the universal quantifiers in TGDs, negative constraints, and EGDs, and we implicitly assume that all sets of dependencies and/or constraints are finite.

2.2 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 [6] 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 if 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 $chase^0(D, \Sigma)$, is 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 $chase^k(D, \Sigma)$, 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 each 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 $k \to \infty$.

The (possibly infinite) chase relative to TGDs is a *universal model*, i.e., there is a homomorphism from $\mathit{chase}(D, \Sigma)$ onto every $B \in \mathit{mods}(D, \Sigma)$ [6]. 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 $\mathit{chase}(D, \Sigma) \models Q$. For sets of guarded TGDs Σ, such BCQs Q can be evaluated on an initial fragment of $\mathit{chase}(D, \Sigma)$ of constant depth $k \cdot |Q|$, which is possible in polynomial time in the data complexity.

2.3 Datalog+/– Ontologies

A *Datalog+/– ontology* $\mathit{KB} = (D, \Sigma)$, where $\Sigma = \Sigma_T \cup \Sigma_E \cup \Sigma_{NC}$, consists of a database D, a set of TGDs Σ_T, a set of non-conflicting EGDs Σ_E, and a set of negative constraints Σ_{NC}. We say *KB* is *guarded* (resp., *linear*) iff Σ_T is guarded (resp., linear). Example 1 illustrates a simple Datalog+/– ontology, which is used in the sequel as a running example.

Example 1 Consider the following simple Datalog+/– ontology $O = (D, \Sigma)$ for movies with:

$$\begin{aligned} D = \{&\mathit{biography}(m_1),\ \mathit{psy\_thriller}(m_2),\ \mathit{biography}(m_3),\\ &\mathit{thriller}(m_4),\ \mathit{documentary}(m_5),\ \mathit{psy\_thriller}(m_6),\\ &\mathit{hasActor}(m_1, a_1),\ \mathit{hasActor}(m_2, a_2),\\ &\mathit{hasActor}(m_2, a_3),\ \mathit{hasActor}(m_3, a_4),\\ &\mathit{hasActor}(m_4, a_2),\ \mathit{hasDirector}(m_4, d_1)\}; \end{aligned}$$

$$\begin{aligned} \Sigma = \{&\mathit{documentary}(T) \to \mathit{movie}(T),\\ &\mathit{thriller}(T) \to \mathit{movie}(T),\\ &\mathit{biography}(T) \to \mathit{documentary}(T),\\ &\mathit{psy\_thriller}(T) \to \mathit{thriller}(T),\\ &\mathit{movie}(T) \to \exists A\, \mathit{hasActor}(T, A),\\ &\mathit{movie}(T) \to \exists D\, \mathit{hasDirector}(T, D)\}. \end{aligned}$$

Intuitively, D encodes that m_1 is a biography with actor a_1, m_2 is a psychological thriller with actors a_2 and a_3, m_3 is a biography with actor a_4, m_4 is a thriller with actor a_2 and director d_1, m_5 is a documentary, and m_6 is a psychological thriller. The set of TGDs Σ encodes a classification for movies based on the genre, i.e., documentaries and thrillers are both movies, biographies are documentaries, and psychological thrillers are thrillers. The last two TGDs say that every movie has an actor and a director. ■

3 PP-Datalog+/–: Syntax and Semantics

In this section, we introduce the PP-Datalog+/– language, an extension of Datalog+/– with both a preference model and a probabilistic model; this formalism is based on the one first presented in [23], which extends Datalog+/– with preferences (but not probabilities). To this end, we assume the following sets giving rise to the logical languages for ontologies, preferences, and probability models: Δ_{Ont}, Δ_{Pref}, and Δ_M are finite sets of constants, $\mathcal{R}_{\mathit{Ont}}$, $\mathcal{R}_{\mathit{Pref}}$, and $\mathcal{R}_M$ are finite sets of predicate symbols such that $\mathcal{R}_M \cap \mathcal{R}_{\mathit{Ont}} = \emptyset$, and $\mathcal{V}_{\mathit{Ont}}$, $\mathcal{V}_{\mathit{Pref}}$, and $\mathcal{V}_M$ are infinite sets of variables. In the following, we assume w.l.o.g. that $\mathcal{R}_{\mathit{Pref}} \subseteq \mathcal{R}_{\mathit{Ont}}$, $\Delta_{\mathit{Pref}} \subseteq \Delta_{\mathit{Ont}}$, and $\mathcal{V}_{\mathit{Pref}} \subseteq \mathcal{V}_{\mathit{Ont}}$. These sets give rise to corresponding *Herbrand bases*, denoted $\mathcal{H}_{\mathit{Ont}}$, $\mathcal{H}_{\mathit{Pref}}$, and $\mathcal{H}_M$, respectively, consisting of all possible ground atoms that can be formed. Clearly, we have $\mathcal{H}_{\mathit{Pref}} \subseteq \mathcal{H}_{\mathit{Ont}}$, i.e., preference relations are defined over a subset of the possible ground atoms.

3.1 Preference Models

A *preference relation* is any binary relation $\succ\ \subseteq \mathcal{H}_{\mathit{Pref}} \times \mathcal{H}_{\mathit{Pref}}$. In this paper, we are interested in *strict partial orders (SPOs)*, which are irreflexive and transitive (and thus asymmetric) relations—we consider these to be the minimal requirements for a preference relation to be useful in the applications that we envision. One way of specifying such a relation, which is especially compatible with our approach, is the preference formula framework of [7]. In this work, we assume the existence of a general *preference model* P specifying an SPO over a subset of $\mathcal{H}_{\mathit{Ont}}$, denoted $\succ_P$; in general, we treat $\succ_P$ as a set of ordered pairs. When an SPO $\succ$ is induced by an assignment of a numeric score $\mathit{score}(a)$ to each element a in such a way that $a_1 \succ a_2$ iff $\mathit{score}(a_1) > \mathit{score}(a_2)$, then $\succ$ is *score-based*.

Example 2 Continuing Example 1, a preference relation may be specified by a user over the *movie* atoms; an example of such a relation is the transitive closure of the graph shown in Fig. 1 (left side). This preference relation can, e.g., be obtained from a user with the following preferences:

– Prefers all movies where actor a_2 appears to any other movie, e.g., $\mathit{movie}(m_2)$ and $\mathit{movie}(m_4)$ are preferred and not comparable with each other.
– Prefers thrillers and psychological thrillers to documentaries, e.g., we have that $\mathit{movie}(m_6)$ is preferred over $\mathit{movie}(m_1)$, $\mathit{movie}(m_3)$, and $\mathit{movie}(m_5)$.
– As a particular preference, the user prefers $\mathit{movie}(m_1)$ to $\mathit{movie}(m_5)$. ■

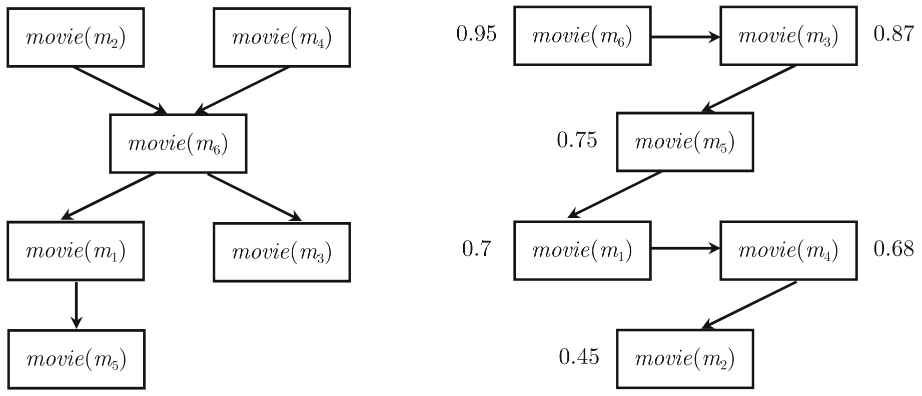


Fig. 1 User preference relation $\succ_P$ for Example 1 (*left side*), and preference relation $\succ_M$ induced by probability values (*right side*)

3.2 Probabilistic Models

For modeling uncertainty, we assume the existence of a probabilistic model M that represents a probability distribution $\Pr_M$ over some set $X = \{X_1, \ldots, X_n\}$ 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 of Datalog+/– adopted here was first introduced in [22]; probabilistic query answering (without preferences) in a version of this model using Markov logic was also studied in [12].

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

Definition 1 Let M be a probabilistic model. Then, a *(probabilistic) annotation* λ relative to M is a (finite) set of expressions $\langle A_i = x_i \rangle$, where (i) A_i is an atom over $\mathcal{R}_M$, $\mathcal{V}_M$, and Δ_M, and (ii) $x_i \in \{0, 1\}$. A probabilistic annotation is *valid* iff for any two different expressions $\langle A = x \rangle, \langle B = y \rangle \in \lambda$, no substitution exists that unifies A and B.

Intuitively, a probabilistic annotation is used to describe the class of events in which the random variables in a probabilistic model M are compatible with the settings of the random variables described by λ, i.e., each X_i has the value x_i. A *probabilistic scenario* is a valid probabilistic annotation λ for which $|\lambda| = |X|$ and all $\langle A = x_i \rangle \in \lambda$ are such that A is ground. We use $scn(M)$ to denote the set of scenarios in M.

Example 3 Continuing with the running example, suppose that an online movie rating system is used to derive a probabilistic model that assigns probabilities specifying how likely it is that the user will like each movie in the knowledge base. The system in question could be aggregating information from user feedback, their friends' feedback, and any other information available to it. Thus, the system informs the user of the probability associated with each *movie* atom; this could be done by extending the ontology from Example 1 by replacing such atoms in the database with formulas of the form $movie(M) : \{enjoy(M) = 1\}$, where $enjoy(M)$ denotes the probabilistic event that movie M will be enjoyed by the user—we are assuming here that the model is designed for a single user. Figure 1 (right side) gives an example of such a probability assignment (derived as explained in the semantics section below), along with the preference relation in graph form that is induced by these values, assuming that higher probabilities are more preferable. ■

3.3 Syntax of PP-Datalog+/–

As seen in Fig. 1, the particular challenge encountered in PP-Datalog+/– ontologies is that the preference model yields a certain precedence relation that may disagree with the one induced by the probabilistic model. To address this, we make use of *preference combination operators*, which take two preference relations (an SPO and a score-based SPO) and produce a third one satisfying two basic properties.

Definition 2 Let $\succ_P$ be an SPO and $\succ_M$ be a score-based SPO. A *preference combination operator* $\otimes(\succ_P, \succ_M)$ yields a relation $\succ^*$ such that (i) $\succ^*$ is an SPO, and (ii) if $a_1 \succ_P a_2$ 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" combination of the two relations; as we show in Sect. 4 below, particular implementations may satisfy further desirable properties.

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

Definition 3 A *PP-Datalog+/– ontology* (or *PP-KB*) is of the form $KB = (O, P, M, \otimes)$, where O is a set of elements of the form $\sigma : \lambda$ such that σ is a ground atom, TGD, EGD, or constraint, and λ is a probabilistic annotation relative to M, P is a preference model, M is a probabilistic model with Herbrand bases $\mathcal{H}_{Ont}$, $\mathcal{H}_{Pref}$, and $\mathcal{H}_M$, respectively, such that

$\mathcal{H}_{Pref} \subseteq \mathcal{H}_{Ont}$, and $\otimes$ is a preference combination operator. If O is a guarded Datalog+/– ontology, then *KB* is *guarded*.

3.4 Semantics of PP-Datalog+/–

We now define the semantics of PP-Datalog+/– ontologies. Let $KB = (O, P, M, \otimes)$ be PP-Datalog+/– ontology; the semantics of PP-Datalog+/– arises as a direct combination of the semantics of Datalog+/– and that of the preference and probabilistic models. Relative to the probabilistic model, we have that $\Pr_{KB}(a) = p$ iff

$$p = \sum_{\lambda \in scn(M),\, O_\lambda \models a} \Pr_M(\lambda).$$

Here, O_λ is the Datalog+/– ontology *induced* from O by λ, which is the set of all formulas σ such that there exists $\sigma : \lambda'$ in O and λ is a ground instance of λ'. We refer to the score-based preference relation induced by $\Pr_M$, denoted $\succ_M$, as the *probabilistic preference relation* associated with *KB*. Let $\succ^* = \otimes(\succ_P, \succ_M)$; we say that $KB \models a_1 \succ^* a_2$ iff:

(i) $O \models a_1$ and $O \models a_2$; and
(ii) $a_1 \succ^* a_2$.

Intuitively, the consequences of $KB = (O, P, M, \otimes)$ are computed in terms of the classical consequences of the Datalog+/– ontology O, and the preference combination operator yields a preference relation over pairs of atoms in $\mathcal{H}_{Ont}$.

3.5 Skyline and k-Rank Queries

We next define the types of queries that we address in this paper. In detail, we are interested in skyline queries [4], a well-known class of queries that can be issued over preference-based formalisms, and the iterated computation of skyline answers [7] that allows us to *rank* atoms relative to an SPO $\succ$ as follows: $rank(a, \succ) = 1$ iff there is no atom b such that $b \succ a$; otherwise, $rank(a, \succ) = k$ where k is the maximum number such that there exists an atom b with $b \succ a$ and $rank(b, \succ) = k - 1$. Note that $rank(a, \succ) = score(a)$ for SPOs $\succ$ based on a score function *score*.

In the following, we focus on a specific kind of classical queries, called disjunctive atomic queries (DAQs), which are disjunctions of atoms.

Definition 4 Let $KB = (O, P, M, \otimes)$ be a PP-Datalog+/– ontology, $\succ^* = \otimes(\succ_P, \succ_M)$, and $Q(\mathbf{X}) = q_1(\mathbf{X}) \vee \cdots \vee q_n(\mathbf{X})$ be a DAQ. Then, an *answer* to Q is any θq_i entailed by O, where θ is a ground substitution for the variables in $Q(\mathbf{X})$. A *skyline answer* to Q relative to $\succ^*$ 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 either (a) the length of S is k or (b) no more skyline answers to $Q(\mathbf{X})$ remain.

Intuitively, both types of answers to DAQs are atomic consequences of O that satisfy the query: skyline answers are sets of atoms that are not dominated by any other such atom, while k-rank answers are k-tuples of atoms sorted according to the preference relation. We refer to these as answers in *atom form*. Note that k-rank answers are only defined when the preference relation is transitive; they are a generalization of traditional top-k answers [30] that are still defined when $\succ^*$ is not a weak order, and their name arises from the concept of *rank* introduced in [7].

In the next section, we present algorithms for implementing particular preference combination operators, which (as we show through a series of properties) produce a new preference relation $\succ^*$ that is useful in answering k-rank queries and that adequately reflects both the initial preferences expressed by the user as well as the fact that higher-probability answers are more desirable.

4 Preference Combination Operators

In this section, we study two families of preference combination operators and the semantic properties of several particular instantiations. In Sect. 6, we then provide an algorithm that uses these operators for answering k-rank queries to PP-Datalog+/– ontologies in polynomial time in the data complexity (modulo the cost of computing probabilities with respect to the probabilistic model M).

4.1 An Egalitarian Class of Combination Operators

The first class of operators that we propose, called *egalitarian* operators, in principle allows the combination of its input preference relations to closely resemble either one.

4.1.1 A General Preference Combination Operator

Algorithm ComPrefsGen in Fig. 2 implements a family of preference combination operators using a value $t \in [0, 1]$ that allows the user to choose how much influence the probabilistic model has on the output preference relation; we use $\succ^*_t$ to denote the output relation. The algorithm iterates through all pairs of elements in $\succ_P$ and, if (i) $\succ_M$ disagrees with $\succ_P$, (ii) the difference in score is greater than t, and (iii) inserting the pair according to $\succ_M$ does not produce a cycle, then the pair is inserted in reverse order into the output; oth-

```
Algorithm 1: ComPrefsGen($\succ_P, \succ_M, t$)
Input: SPO $\succ_P$, score-based SPO $\succ_M$, both over $\mathcal{H}_{Ont}$,
        and $t \in [0, 1]$.
Output: SPO $\succ_t^* \subseteq \mathcal{H}_{Ont} \times \mathcal{H}_{Ont}$.
 1. Initialize $\succ_t^*$ as a copy of $\succ_P$;
 2. For every pair $(a_i, a_j) \in \succ_P$ do begin
 3.     if $rank(a_j, \succ_M) - rank(a_i, \succ_M) > t$, and changing $(a_i, a_j)$
             to $(a_j, a_i)$ in $\succ_t^*$ does not introduce a cycle in its
             associated graph, then
 4.         change $(a_i, a_j)$ to $(a_j, a_i)$ in $\succ_t^*$;
 5. End;
 6. Return $transitiveClosure(\succ_t^*)$.
```

Fig. 2 An algorithm for combining an arbitrary SPO with a weak order in the form of a score-based SPO

erwise, the output contains the same pair as $\succ_P$. Finally, the algorithm outputs the transitive closure of this relation. Note that if $t = 0$, then the probability-induced relation has precedence; at the other end of the spectrum, if $t = 1$, then ComPrefsGen($\succ_P, \succ_M, t$) $= \succ_P$ (these properties are presented formally in Theorem 3). The following is an example of how the algorithm works.

Example 4 Consider again the running example. Figure 3 (top) shows the result of running ComPrefsGen over the two SPOs with $t = 0$ (left side) and $t = 0.3$ (right side)—note that, for clarity, the transitive closures are not shown in these figures. Figure 3 (bottom) shows the result of computing the rank via the iterative computation of the skyline answers for the three SPOs. Here, the user's original preferences ($\succ_P$) are preserved to a greater extent in $\succ_{0.3}^*$ than in $\succ_0^*$: the answers in the former have a difference in rank of at most 1, whereas those in the latter have a difference of 2 (out of a total of 4) in multiple cases. This reflects the greater influence that $\succ_P$ has in the result for $t = 0.3$ as compared to $t = 0$. ■

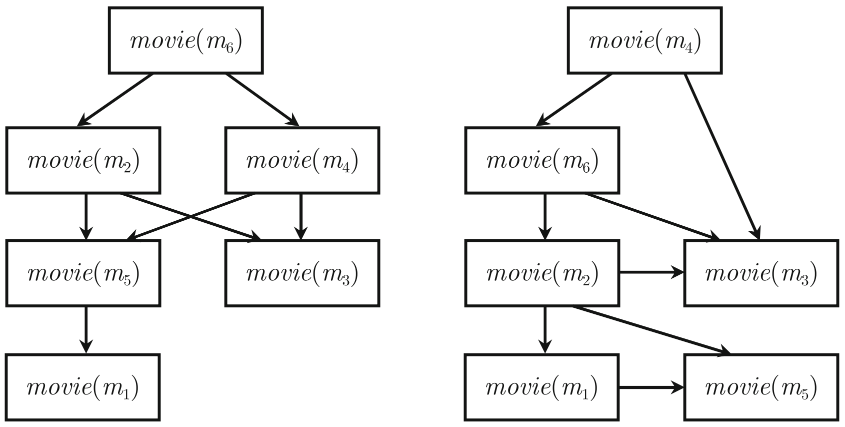
The following result shows that ComPrefsGen encodes indeed a family of preference combination operators (cf. Definition 2); given $t \in [0, 1]$, we denote the corresponding operator with $\otimes_t^{gen}$.

Theorem 1 *Let $\succ_P$ be an SPO, $\succ_M$ be a score-based SPO, $t \in [0, 1]$, and $\succ_t^* =$ ComPrefsGen($\succ_P, \succ_M, t$). Then, for any value of t,*

(i) *$\succ_t^*$ is an SPO, and*
(ii) *if $a_1 \succ_P a_2$ and $a_1 \succ_M a_2$, then $a_1 \succ_t^* a_2$.*

Proof (i) We must show that $\succ^*$ is irreflexive and transitive. Let $\succ'$ be the intermediate result of the ComPrefsGen algorithm just before computing the transitive closure. By assumption, $\succ_P$ 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 ComPrefsGen to change the order of a pair in $\succ_P$ is that the pair in $\succ_M$ be reversed. As by assumption, this is not the case; the statement follows. □

We now study the properties satisfied by the output of ComPrefsGen. The following theorem states that the output for $t = 0$ is invariant to changes in the assigned scores, as long as the order is the same. This is an important property, as it implies that approximation algorithms can be used



Rank	$\succ_P$	$\succ_0^*$	$\succ_{0.3}^*$
1	$[X/m_2]$, $[X/m_4]$	$[X/m_6]$	$[X/m_4]$
2	$[X/m_6]$	$[X/m_2]$, $[X/m_4]$	$[X/m_6]$
3	$[X/m_1]$, $[X/m_3]$	$[X/m_3]$, $[X/m_5]$	$[X/m_2]$
4	$[X/m_5]$	$[X/m_1]$	$[X/m_1]$, $[X/m_3]$
5	–	–	$[X/m_5]$

Fig. 3 Combinations of user preferences and probability-induced preferences with the ComPrefsGen algorithm (transitive closure not shown)—*top*, *left side*: $t = 0$; *top*, *right side*: $t = 0.3$. *Bottom*: answers to query *movie*(X) according to their rank relative to the original preference relation (Fig. 1, left side) and the two combinations

to compute the probability values that induce $\succ_M$—as long as the relative order of the atoms is guaranteed to be correct, there is no need to compute the exact values when $t = 0$.

Theorem 2 *Let $\succ_P$ be an SPO, and $\succ_M$ and $\succ_{M'}$ be score-based SPOs. If $\succ_M = \succ_{M'}$, then:*

$\mathsf{ComPrefsGen}(\succ_P, \succ_M, 0) = \mathsf{ComPrefsGen}(\succ_P, \succ_{M'}, 0)$.

Proof For $t = 0$, the algorithm checks (in line 3) if there exist pairs (a_i, a_j) in $\succ_P$ such that $\Pr(a_j) > \Pr(a_i)$. Since $\succ_M = \succ_{M'}$, by assumption, $\Pr(a_j) > \Pr(a_i)$ relative to $\succ_M$ iff this is the case relative to $\succ_{M'}$, and thus the outputs in both cases must be identical. □

The next theorem shows the behavior of the operator for the extreme values of the parameter t.

Theorem 3 *Let $\succ_P$ be an SPO and $\succ_M$ be a score-based SPO. Then,*

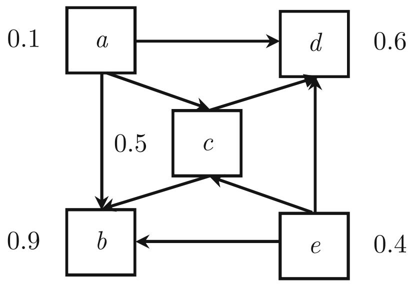
(i) *$rank(a, \mathsf{ComPrefsGen}(\succ_P, \succ_M, 0)) = rank(a, \succ_M)$, for every atom a; and*
(ii) *$\mathsf{ComPrefsGen}(\succ_P, \succ_M, 1) = \succ_P$.*

Proof (i) For $t = 0$, the output relation is the result of comparing all pairs in $\succ_P$ and replacing them with the order imposed on their elements by $\succ_M$. Thus, given that $\succ_P$ is transitive, the sequence of atoms giving rise to the rank of each atom is reordered in the resulting relation according to the rank of each element in $\succ_M$, and the statement follows.
(ii) Direct consequence of the condition in line 3 of the algorithm: since the difference in probabilities can never be greater than 1, the output relation is a copy of $\succ_P$. □

Finally, the following theorem studies a property that is analogous to the "sensitivity postulates" presented in [34], [33, pp. 12–14,63–64]. Note, however, that these postulates were designed to combine two score-based preference relations, whereas our setting is more general—the "faithfulness" postulate in [33] is already guaranteed by our definition of preference combination operator.

Theorem 4 *Let $KB = (O, P, M, \otimes_t^{gen})$ be a PP-Datalog+/– ontology, Q be a DAQ, $k \geqslant 0$, and $\succ_t^* = \otimes_t^{gen}(\succ_P, \succ_M)$, with $t \in [0, 1]$. For every atom a that is an answer to Q but does not belong to any k-rank answer to Q over KB relative to $\succ_t^*$, there exists a probabilistic model M' and $t' \in [0, 1]$ such that a belongs to some k-rank answer to Q over $KB' = (O, P, M', \otimes_{t'}^{gen})$ relative to SPO $\succ'_{t'} = \otimes_{t'}^{gen}(\succ_P, \succ_{M'})$.*

Proof By assumption, there exist atoms $a_1, \ldots, a_n$ such that $a_i \succ_t^* a$. Now, let $t' = 0$, and set the probabilities in M' such that $\Pr_{M'}(a) > \Pr_{M'}(a_i)$, for $1 \leqslant i \leqslant n$. By construction, $a \succ'_{t'} a_i$, and the statement follows. □



Order	Edge	$t = 0.1$	$t = 0.05$
1	(c, d)	No (t)	Yes
2	(e, d)	No (cycle)	Yes
3	(e, c)	No (t)	Yes
4	(a, c)	Yes	No (cycle)
5	(a, d)	Yes	Yes
6	(a, b)	Yes	No (cycle)
7	(c, b)	Yes	No (cycle)
8	(e, b)	Yes	No (cycle)

Fig. 4 User preferences for Example 5 (*top*) and the results of applying Algorithm ComPrefsGen for two different values of t (*bottom*). In the table, "yes" and "no" mean whether or not the edge is reversed in the result—in the negative cases, the reason is given in parenthesis

4.1.2 An Egalitarian Operator Based on Rank

In this section, we study a different kind of combination operator, which is based on a tradeoff between generality and the properties that can be proved about the result.

The following example shows a shortcoming of the $\otimes_t^{gen}$ operator in that it fails to exhibit a kind of monotonicity that might be expected when the two input relations disagree on a given pair (a_1, a_2). In particular, if the operator chooses the order imposed by $\succ_M$, then it may not continue to do so for smaller values of t. On the other hand, if the operator chooses the order imposed by $\succ_P$, then it may also fail to do so for greater values of t.

Example 5 Consider the SPO in Fig. 4 (top), where the directed edges between nodes describe an SPO $\succ_P$ and the numbers beside each node define score-based SPO $\succ_M$. The table at the bottom specifies what happens when the two are input to ComPrefsGen, and the edges are inspected in the specified order, for two values of t. Note row 4, where edge (a, c) becomes (c, a) under $t = 0.1$, but stays unaltered under $t = 0.05$, and row 6, where edge (a, b) stays unaltered under $t = 0.05$, but becomes (b, a) under $t = 0.1$. ■

As shown in Example 5, the cause of this unpredictability is the potential presence of cycles when merging SPOs. Algorithm 2 (Fig. 5) avoids this issue by combining the two relations into a score-based one using the rank of each atom

```
Algorithm 2: ComPrefsRank($\succ_P, \succ_M, f$)
Input: SPO $\succ_P$, score-based SPO $\succ_M$, both over $\mathcal{H}_{Ont}$,
       and integer binary function $f$ with $f(x,y) \in [x,y]$.
Output: Score-based SPO $\succ_f^{rank} \subseteq \mathcal{H}_{Ont} \times \mathcal{H}_{Ont}$.

1. Initialize $\succ_f^{rank}$ as an empty SPO over $\mathcal{H}_{Ont}$;
2. Initialize $rank_1$ and $rank_2$ as empty rankings over $\mathcal{H}_{Ont}$;
3. Compute the ranks of each ground atom $a$ relative to $\succ_P$
       and $\succ_M$, and store them in $rank_1$ and $rank_2$, respectively;
4. For every ground atom $a$ in $dom(rank_1)$ do
5.     add $a$ to $\succ_f^{rank}$ with score $f(rank_1(a), rank_2(a))$;
6. Return $transitiveClosure(\succ_f^{rank})$.
```

Fig. 5 Algorithm for a rank-based combination of an SPO with a score-based SPO

Table 1 Score-based SPOs produced by ComPrefsRank for two different input functions (*min* and *max*)

Atom	Rank ($\succ_P$)	Rank ($\succ_M$)	*min*	*max*
movie(m_2)	1	6	1	6
movie(m_4)	1	5	1	5
movie(m_6)	2	1	1	2
movie(m_3)	3	2	2	3
movie(m_1)	3	4	3	4
movie(m_5)	4	3	3	4

relative to each input relation. It takes as input an integer binary function that combines the two ranks and assigns it to the atom in the output relation—this function can be as simple as *min*, *max*, or *avg*, or a more complex function that takes into account how much the user would like to base the new rank on one relation or the other. A detailed treatment of such functions is outside the scope of this paper.

The following example shows the behavior of Algorithm ComPrefsRank over the running example.

Example 6 Consider again the running example. The results produced by ComPrefsRank for two different input functions (*min* and *max*) are shown in Table 1. The SPOs induced by the scores in the last two columns are shown in Fig. 6. ■

The following theorem states that ComPrefsRank implements a family of preference combination operators for certain functions; given function f, we denote the corresponding operator with $\otimes_f^{rank}$.

Theorem 5 *Let $\succ_P$ be an SPO, $\succ_M$ be a score-based SPO, f be an integer binary function such that $f(x, y) \in [x, y]$, and $\succ_f^{rank}$ = ComPrefsRank($\succ_P, \succ_M, f$). Then:*

(i) *$\succ_f^{rank}$ is an SPO; and*
(ii) *If $a_1 \succ_P a_2$, $a_1 \succ_M a_2$ and $f \in \{min, max, avg\}$, then $a_1 \succ_f^{rank} a_2$.*

Proof (i) Immediate, since $\succ_f^{rank}$ is a score-based SPO.
(ii) Let $r_1^P = rank(a_1, \succ_P)$, $r_2^P = rank(a_1, \succ_P)$, $r_1^M = rank(a_1, \succ_M)$, and $r_2^M = rank(a_1, \succ_M)$. The assumptions imply that $r_1^P < r_2^P$ and $r_1^M < r_2^M$. Clearly, $f(r_1^P, r_1^M) < f(r_2^P, r_2^M)$ for $f \in \{min, max, avg\}$, and thus $a_1 \succ_f^{rank} a_2$. □

Analyzing the proof of Theorem 5, we can see that the result also holds for variations of the functions considered—as long as the condition $f(r_1^P, r_1^M) < f(r_2^P, r_2^M)$ holds.

Another property of ComPrefsRank that can be shown is the analogous of Theorem 2; as the precondition implies that ranks relative to $\succ_M$ and $\succ_{M'}$ are equal, the result clearly holds. Finally, the following theorem discusses properties related to the postulates from [33] for this algorithm.

Theorem 6 *Let $KB = (O, P, M, \otimes_f^{rank})$ be a PP-Datalog+/– ontology, Q be a DAQ, $k \geqslant 0$, and $\succ_f^* = \otimes_f^{rank}(\succ_P, \succ_M)$. Then:*

(i) *Let $f \in \{min, max, avg\}$, M' be a probabilistic model such that for some ground atom a, we have $\Pr_{KB'}(a) \leqslant \Pr_{KB}(a)$ and $\Pr_{KB'}(a') = \Pr_{KB}(a')$ for every ground atom $a' \neq a$, where $KB' = (O, P, M', \otimes_f^{rank})$. If a does not belong to any k-rank answer to Q over KB relative to $\succ_f^*$, then a does not belong to any k-rank answer to Q over KB' relative to $\succ'_f = \otimes_f^{rank}(\succ_P, \succ_{M'})$.*
(ii) *Given the setup in (i), if a belongs to some k-rank answer to Q over KB relative to $\succ_f^*$ and $\Pr_{KB'}(a) \geqslant \Pr_{KB}(a)$, then a also belongs to some k-rank answer to Q over KB' relative to $\succ'_f = \otimes_f^{rank}(\succ_P, \succ_{M'})$.*
(iii) *For every atom a that is an answer to Q but does not belong to any k-rank answer to Q over KB relative to $\succ_f^*$, there exists a probabilistic model M' and f' such*

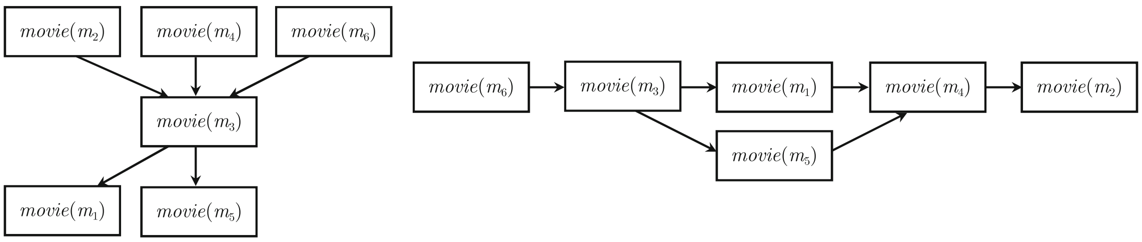


Fig. 6 Results of applying Algorithm ComPrefsRank using the functions *min* (*left side*) and *max* (*right side*)

```
Algorithm 3: ComPrefsPT($\succ_P, \succ_M, p$)
Input: SPO $\succ_P$, score-based SPO $\succ_M$, both over $\mathcal{H}_{Ont}$;
       probability threshold $p \in [0, 1]$.
Output: SPO $\succ_p^{pt} \subseteq \mathcal{H}_{Ont} \times \mathcal{H}_{Ont}$.
 1. Initialize $\succ_p^{pt}$ as a copy of $\succ_P$;
 2. For every atom $a$ appearing in $\succ_p^{pt}$ do
 3.     if $rank(a, \succ_M) < p$ then
 4.         remove element $a$ from $\succ_p^{pt}$;
 5. Return $transitiveClosure(\succ_p^{pt})$.
```

Fig. 7 Algorithm for performing a user-biased combination of an SPO with a score-based SPO, based on a given probability threshold

that a belongs to some k-rank answer to Q over $KB' = (O, P, M', \otimes_{f'}^{rank})$ *relative to* $\succ'_{f'} = \otimes_{f'}^{rank}(\succ_P, \succ_{M'})$.

Proof sketch (i) $\Pr_{KB'}(a) \leqslant \Pr_{KB}(a)$ implies that we have $rank(a, \succ_{M'}) \leqslant rank(a, \succ_M)$, and the result from Theorem 5 can be used to show the result.

(ii) Analogous to (i): $\Pr_{KB'}(a) \geqslant \Pr_{KB}(a)$ implies that $rank(a, \succ_{M'}) \geqslant rank(a, \succ_M)$.

(iii) Using $f' = min$, the rest of the proof is analogous to that of Theorem 4. □

4.2 A User-Biased Class of Combination Operators

The second class of operators that we study is *user-biased*. Their main characteristic is that they base the produced preference relation on the user's preferences and use the probabilistic model as a secondary source of "advice".

4.2.1 An Operator Based on Probability Thresholds

One way of obtaining a preference relation that is biased by the user's input is to remove from consideration any atom that has probability below a given threshold—all previously existing relationships between atoms remain the same. The following is an example of how this operator works.

Example 7 Returning to the running example, Fig. 8 shows the original SPO $\succ_P$ and the result of applying the preference combination operator implemented by Algorithm ComPrefsPT with threshold 0.75, called $\succ_{0.75}^{pt}$. Note that relations between remaining atoms that held in the original still hold; for instance, $movie(m_6) \succ_{0.7}^{pt} movie(m_5)$, which was true relative to $\succ_P$ due to transitivity.

The following theorem shows that ComPrefsPT (Fig. 7) satisfies the conditions stated in Definition 2 and thus implements a family of preference combination operators; given $p \in [0, 1]$, we denote the corresponding operator with $\otimes_p^{pt}$.

Theorem 7 *Let* $\succ_P$ *be an SPO,* $\succ_M$ *be a score-based SPO,* $p \in [0, 1]$*, and* $\succ_p^{pt} =$ ComPrefsPT$(\succ_P, \succ_M, p)$. *Then,*

(i) $\succ_p^{pt}$ *is an SPO; and*

(ii) *If* $a_1 \succ_P a_2$*,* $a_1 \succ_M a_2$*,* $rank(a_1, \succ_M) \geqslant p$*, and* $rank(a_2, \succ_M) \geqslant p$*, then* $a_1 \succ_p^{pt} a_2$.

Proof (i) Direct consequence, since $\succ_p^{pt} \subseteq \succ_P$.

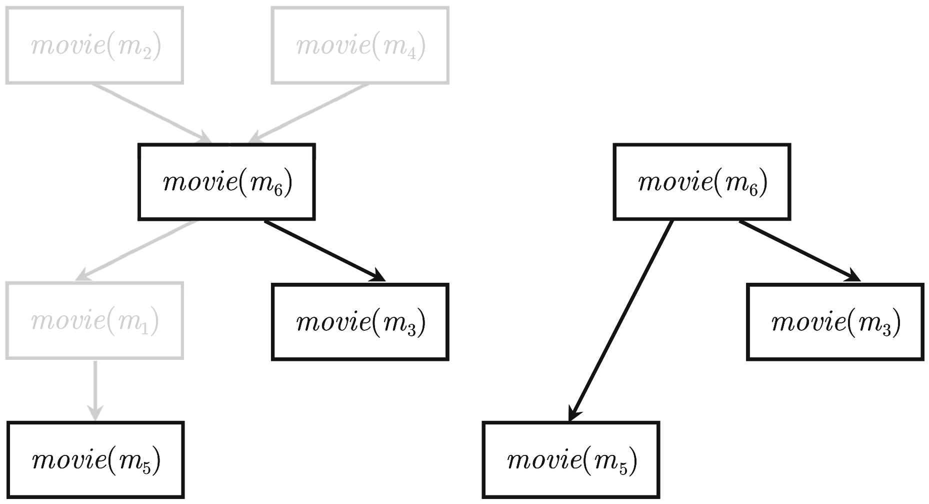
(ii) The two conditions $rank(a_1, \succ_M) \geqslant p$ and $rank(a_2, \succ_M) \geqslant p$ imply that a_1 and a_2 appear in $\succ_p^{pt}$; therefore, since $a_1 \succ_P a_2$ and $\succ_p^{pt} \subseteq \succ_P$, the statement follows.

We can also show that the properties described in Theorem 6 hold for ComPrefsPT.

Theorem 8 *Let* $KB = (O, P, M, \otimes_p^{pt})$ *be a PP-Datalog+/– ontology, Q be a DAQ,* $p \in [0, 1]$*, and* $\succ_p^* = \otimes_p^{pt}(\succ_P, \succ_M)$. *Then:*

(i) *Let M′ be a probabilistic model such that for some ground atom a, we have* $\Pr_{KB'}(a) \leqslant \Pr_{KB}(a)$ *and*

Fig. 8 The original SPO $\succ_P$ (*left side*), where atoms whose probabilities are less than 0.75 according to $\succ_M$ have been shaded, and the result of applying Algorithm ComPrefsPT (*right side*)



$\Pr_{KB'}(a') = \Pr_{KB}(a')$ for every ground atom $a' \neq a$, where $KB' = (O, P, M', \otimes_p^{pt})$. If a does not belong to any k-rank answer to Q over KB relative to $\succ_p^*$, then a does not belong to any k-rank answer to Q over KB' relative to $\succ'_p = \otimes_p^{pt}(\succ_P, \succ_{M'})$.

(ii) *Given the setup in (i), if a belongs to some k-rank answer to Q over KB relative to $\succ_p^*$ and $\Pr_{KB'}(a) \geqslant \Pr_{KB}(a)$, then a also belongs to some k-rank answer to Q over KB' relative to $\succ'_p = \otimes_p^{pt}(\succ_P, \succ_{M'})$.*

(iii) *For every atom a that is an answer to Q but does not belong to any k-rank answer to Q over KB relative to $\succ_p^*$, there exists a probabilistic model M' and $p' \in [0, 1]$ such that a belongs to some k-rank answer to Q over $KB' = (O, P, M', \otimes_{p'}^{pt})$ relative to $\succ'_{p'} = \otimes_{p'}^{pt}(\succ_P, \succ_{M'})$.*

Proof sketch (i) Clearly, if $\Pr_{KB'}(a) < p$ then the result holds. Otherwise, since a does not belong to any k-rank with respect to $\succ_p^*$, no other atom's status relative to the threshold changes, and ComPrefsPT does not use the probability value to produce its output, the statement follows.

(ii) The fact that a belongs to some k-rank answer implies that $\Pr_{KB}(a) \geqslant p$ and thus $\Pr_{KB'}(a) \geqslant p$. The rest of the proof is analogous to the second part of (i).

(iii) Using $p' = \Pr_{KB}(a)$, and changing the probabilities of all b such that $b \succ_P a$ to $\Pr_{KB}(a) - \epsilon$, for some $\epsilon \in [0, 1]$, it holds $rank(a, \succ'_{p'}) = 1$, and the statement follows. □

4.2.2 A Skyline Sorting Operator

Another possibility for a user-biased operator is to keep the same base structure given by the user and make use of the probability-based preference relation to remove to the furthest extent possible the indifference present among elements of the same rank, i.e., in each iterated skyline. Algorithm ComPrefsSort (see Fig. 9) implements such an operator. The following is an example of how the algorithm works.

Example 8 Consider again the running example. Figure 10 shows $\succ_P$ (left side) and the SPO resulting from ComPrefsSort (right side). Note that the two relations are very similar, except that the indifference between $movie(m_2)$ and $movie(m_4)$ and the one between $movie(m_1)$ and $movie(m_3)$ have been replaced by the ordering suggested by $\succ_M$. ■

The following theorem shows that ComPrefsSort indeed implements a preference combination operator, which we denote with $\otimes^{sort}$.

Theorem 9 *Let $\succ_P$ be an SPO, $\succ_M$ be a score-based SPO, and $\succ^{sort}$= ComPrefsSort$(\succ_P, \succ_M)$. Then:*

(i) *$\succ^{sort}$ is an SPO; and*

Algorithm 4: ComPrefsSort$(\succ_P, \succ_M)$
Input: SPO $\succ_P$, score-based SPO $\succ_M$, both over $\mathcal{H}_{Ont}$.
Output: SPO $\succ^{sort} \subseteq \mathcal{H}_{Ont} \times \mathcal{H}_{Ont}$.

```
1. Initialize $\succ^{sort}$ as a copy of $\succ_P$;
2. $k := 1$;
3. Repeat
4.     $curr\text{-}rank :=$ all elements of rank $k$ relative to $\succ^{sort}$;
5.     $k := k + 1$;
6.     Sort $curr\text{-}rank$ in decreasing order of $rank(\cdot, \succ_M)$
7.     For all $a_i, a_{i+1} \in curr\text{-}rank$ $(1 \leqslant i \leqslant |curr\text{-}rank| - 1)$ do
8.         add preference $a_i \succ^{sort} a_{i+1}$;
9. Until $curr\text{-}rank$ is empty;
10. Return $transitiveClosure(\succ^{sort})$.
```

Fig. 9 Algorithm for performing a user-biased combination of an SPO with a score-based SPO, based on sorting skylines

(ii) *If $a_1 \succ_P a_2$ and $a_1 \succ_M a_2$, then $a_1 \succ^{sort} a_2$.*

Proof (i) Since $\succ^{sort}$ is a copy of $\succ_P$ with the addition of preference pairs between elements that were previously unrelated, and there is no possibility of introducing cycles, the statement follows.

(ii) By construction, if $a_1 \succ_P a_2$, then also $a_1 \succ^{sort} a_2$. □

The following theorem states some properties of ComPrefsSort. Note that the property from Theorem 4 (also property (iii) in Theorems 6 and 8) does not hold for ComPrefsSort; this is because the algorithm does not allow the user's preferences to change so drastically depending on the probabilistic model.

Theorem 10 *Let $KB = (O, P, M, \otimes^{sort})$ be a PP-Datalog +/– ontology, Q be a DAQ, and $\succ^* = \otimes^{sort}(\succ_P, \succ_M)$. Then:*

(i) *Let M' be a probabilistic model such that for some ground atom a, we have $\Pr_{KB'}(a) \leqslant \Pr_{KB}(a)$ and $\Pr_{KB'}(a') = \Pr_{KB}(a')$ for every ground atom $a' \neq a$, where $KB' = (O, P, M', \otimes^{sort})$. If a does not belong to any k-rank answer to Q over KB relative to $\succ^*$, then a does not belong to any k-rank answer to Q over KB' relative to $\succ' = \otimes^{sort}(\succ_P, \succ_{M'})$.*

(ii) *Given the setup in (i), if a belongs to some k-rank answer to Q over KB relative to $\succ^*$ and $\Pr_{KB'}(a) \geqslant \Pr_{KB}(a)$, then a also belongs to some k-rank answer to Q over KB' relative to $\succ' = \otimes^{sort}(\succ_P, \succ_{M'})$.*

Proof sketch (i) Since the result of ComPrefsSort only differs from $\succ_P$ in that elements of the same rank relative to $\succ_P$ are ordered with respect to $\succ_M$, all elements dominating a in $\succ^*$ still do so in $\succ'$, and the statement follows.

(ii) Analogous to (i). □

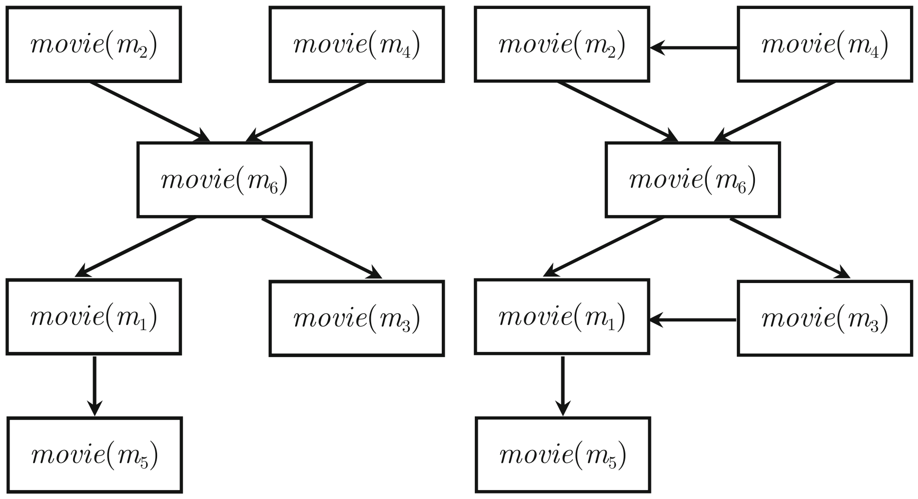


Fig. 10 The original SPO $\succ_P$ (*left side*) and the result of applying Algorithm ComPrefsSort (*right side*)

5 Complexity of Combination Algorithms

In this section, we analyze the complexity of all combination algorithms introduced above. We begin by identifying and discussing the main tasks that the algorithms perform. In the rest of this section, we assume that SPOs are represented as directed graphs; the size of such a graph G is determined by n (the number of nodes in the graph, i.e., the number of atoms in the SPO) and e (the number of edges in the graph, i.e., the number of pairs in the SPO).

Transitive closure All four algorithms compute as the last step the transitive closure of the relation that results from combination; this is required to return an SPO. The best known algorithms for computing the transitive closure are the ones from Warshall [32] and Warren [31], which have $O(n^3)$ as worst-case running time. Clearly, in all algorithms, the dominant term is $O(n^3)$, contributed by the transitive closure. However, as we will show in Sect. 8, if this last step is not required, then the running time of the algorithms in the worst case are quite different.

Cycle detection Algorithm ComPrefsGen checks for potential cycles every time an edge from the original SPO needs to be inverted. The procedure to find a cycle in a directed graph can be done with a depth-first search of the graph and checking for a back edge (edges that point from a node to one of its ancestors). The actual running time of this procedure depends on the representation of the graph, e.g., with adjacency lists, the worst-case running time is $O(n+e)$.

Computing probabilities The cost of computing probabilities relative to a probabilistic model M can range from polynomial time (such as in approximation algorithms, or tractable models like polytrees [18] or tractable Markov logic [9]) to #P-complete (such as in general Markov logic [28] or Bayesian networks). Furthermore, in some cases, it may be assumed that the probability of all ground atoms can be done offline and stored in a look-up table, which can be consulted in constant time. This is the case for our experimental evaluation, as scores from IMDB yield probabilities that can be easily computed and added to the database.

5.1 Egalitarian Combination Algorithms

In this section, we give an analysis of the running time of the egalitarian combination algorithms.

Theorem 11 *Let $\succ_P$ be an SPO and $\succ_M$ be a score-based SPO. Let S be the time required to compute $rank(a, \succ_M)$ for any ground atom a. Then,* ComPrefsGen *and* ComPrefsRank *run in time $O(n \cdot S + n^3 + e^2)$ and $O(n \cdot S + n^3)$, respectively, where n (resp., e) is the number of nodes (resp., edges) of $\succ_P$ represented as a directed graph.*

Proof Observe first that computing the probability of each node in M is possible in $O(n \cdot S)$.

As for ComPrefsGen, we then inspect every edge in $\succ_P$ by computing the values of the probabilities of the endpoints and checking for potential cycles, which takes time $O(e \cdot (n + e))$. Finally, we compute the transitive closure of the updated graph that represents the new combined relation, which can be done in $O(n^3)$. Overall, ComPrefsGen can be done in time $O(n \cdot S + n^3 + e^2)$.

As for ComPrefsRank, the ranks of each atom relative to both SPOs are computed. For $\succ_P$, this can be done in time $O(n + e)$; for $\succ_M$, this can be done in time $O(n \cdot \log(n))$ by sorting the nodes by probability values and then traversing that sorted list. Then, we compute for each atom a score based on a function that combines the previously computed ranks of that atom; we assume that this function can be computed in constant time—thus, this adds an $O(n)$ term. Finally, the transitive closure adds a cost of $O(n^3)$. Overall, ComPrefsRank is possible in time $O(n \cdot S + n^3)$. □

As $\succ_P$ describes a preference relation over the consequences of an ontology, we have the following corollary, stating the data tractability of the above algorithms.

Corollary 1 *Let $\succ_P$ be an SPO and $\succ_M$ a score-based SPO, both given by a guarded PP-Datalog+/– ontology KB $= (O, P, M, \otimes)$, with $O = (D, \Sigma)$. Let $\succ'_P$ be the restriction of $\succ_P$ to all (a, b) with $O \models a, b$. Let $\Pr_M(a)$ be computable (exactly or approximately) in polynomial time in the data complexity for any ground atom a with $O \models a$. Then,* ComPrefsGen *and* ComPrefsRank *on $\succ'_P$ and $\succ_M$, using as input KB, run in polynomial time in the data complexity.*

Proof In the guarded case, deciding whether O entails a ground atom is possible in polynomial time in the data complexity [6]. The statement then follows from Theorem 11 and the assumption that probabilities are computable in polynomial time in the data complexity.

5.2 User-Biased Combination Algorithms

In this section, we provide an analysis of the running time of the user-biased combination algorithms.

Theorem 12 *Let $\succ_P$ be an SPO and $\succ_M$ be a score-based SPO. Let S be the time required to compute rank$(a, \succ_M)$ for any ground atom a. Then,* ComPrefsPT *and* ComPrefsSort *run in time $O(n \cdot S + n^3)$, where n is the number of nodes of $\succ_P$ represented as a directed graph.*

Proof Let $e = |\succ_P|$. Observe first that computing the probability of each node in M is possible in $O(n \cdot S)$.

Then, as for ComPrefsPT, this algorithm inspects every node, checking if its associated probability given by M is below the input threshold, and if so, it deletes the node and all of its incoming and outgoing edges. Since we assume that the cost of deleting edges is constant, this operation is possible in time $O(n + e)$. Finally, the transitive closure is computed, which is possible in $O(n^3)$. It thus follows that ComPrefsPT overall runs in time $O(n \cdot S + n^3)$.

As for ComPrefsSort, we compute the rank of each ground atom relative to $\succ_P$ by iteratively computing a skyline of $\succ_P$. Additionally, we sort the elements in each skyline relative to $score_M$. As argued in the proof of Theorem 11, the former can be done in time $O(n+e)$, assuming suitable data structures, while the latter can be done in time $O(n \cdot \log(n))$, as sorting the whole set of nodes also produces a sorting for each skyline. Finally, the transitive closure is possible in time $O(n^3)$, and thus ComPrefsSort can overall be done in time $O(n \cdot S + n^3)$. □

Analogous to our analysis for egalitarian operators in Corollary 1, we have the following corollary stating data tractability, which can be proved analogously to Corollary 1.

Corollary 2 *Let $\succ_P$ be an SPO and $\succ_M$ a score-based SPO, both given by a guarded PP-Datalog+/– ontology KB $= (O, P, M, \otimes)$, with $O = (D, \Sigma)$. Let $\succ'_P$ be the restriction of $\succ_P$ to all (a, b) with $O \models a, b$. Let $\Pr_M(a)$ be computable (exactly or approximately) in polynomial time in the data complexity for any ground atom a with $O \models a$. Then,* ComPrefsPT *and* ComPrefsSort *on $\succ'_P$ and $\succ_M$, using as input KB, run in polynomial time in the data complexity.*

6 Answering k-Rank Queries

As discussed above, in this paper, we are interested in computing the rank of the answers to queries by means of the iterated computation of its skyline answers [4]. We now present a general algorithm to do so, and then analyze its correctness as well as its running time when used in conjunction with either the ComPrefsGen or ComPrefsRank algorithms.

The Algorithm k-Rank (see Fig. 11) begins by computing the combination of the two SPOs in the PP-Datalog+/– ontology and the necessary finite part C of the chase relative to Q. The main while loop iterates through the process of computing the skyline answers to Q relative to this new relation using a *computeSkyline* subroutine (which can be implemented by means of a linear-time scan of C), updating the result by appending these answers in arbitrary order and removing the atoms in the result from C. Once the loop is finished, the algorithm returns the first k results, as the last iteration may add superfluous elements.

Example 9 Consider again the running example and the query $Q = movie(X)$, and $k = 4$. Using the results shown in the various examples above, we can obtain the follow-

```
Algorithm 5: $k$-Rank$(KB, Q, k)$
Input: Guarded PP-Datalog+/– ontology $KB = (O, P, M, \otimes)$,
       DAQ $Q(\mathbf{X})$, and $k \geqslant 0$.
Output: $k$-rank answer $\langle a_1, \ldots, a_{k'} \rangle$ to $Q$, with $k' \leqslant k$.
 1. Initialize $Res$ as an empty vector of ground atoms;
 2. $C := computeChase(O, Q)$;
 3. $\succ^* := \otimes(\succ'_P, \succ_M)$, where $\succ'_P$ is the restriction of $\succ_P$
                          to all $(a, b)$ with $O \models a, b$;
 4. $i := k$;
 5. While $i > 0$ do begin
 6.     $S := computeSkyline(C, Q, \succ^*)$;
 7.     Append $S$ to $Res$ (in arbitrary order);
 8.     Remove $S$ from $C$;
 9.     $i := i - |S|$;
10. End;
11. Return $truncate(Res, k)$.
```

Fig. 11 An algorithm for computing a k-rank answer to DAQ Q relative to the composition of $\succ_P$ and $\succ_M$

ing k-rank answers to Q (in atom form) depending on the preference combination algorithms used:

- ComPrefsGen with $t = 0.3$: $\langle movie(m_4), movie(m_6), movie(m_2), movie(m_1)\rangle$;
- ComPrefsRank with $f = max$: $\langle movie(m_6), movie(m_3), movie(m_1), movie(m_5)\rangle$;
- ComPrefsPT with $p = 0.75$: $\langle movie(m_6), movie(m_3), movie(m_5)\rangle$; and
- ComPrefsSort: $\langle movie(m_4), movie(m_2), movie(m_6), movie(m_3)\rangle$.

Analyzing the differences among the different answers, we can see some characteristics of each operator. For instance, both ComPrefsGen with $t = 0.3$ and ComPrefsSort include one of the user's favorite movies in the first position, though ComPrefsSort includes both of the top elements with respect to $\succ_P$ first and ComPrefsGen puts $movie(m_6)$ before $movie(m_2)$, given the latter's lower probability score. Consider next the results obtained by ComPrefsRank with *max* and ComPrefsPT with $p = 0.75$. The former takes a pessimistic stance, and therefore the very low rank of the user's top choices pushes them outside the final rank. Similarly, the low scores of these elements make them unavailable to be chosen (no matter what value k has) by ComPrefsPT, since they do not surpass the threshold value. ■

The following theorem proves the correctness of the k-Rank algorithm and shows that under certain conditions, it runs in polynomial time for the above four operators $\otimes$.

Theorem 13 *Let $KB = (O, P, M, \otimes)$, with $O = (D, \Sigma)$, be a PP-Datalog+/– ontology, where $\otimes$ is any of the above four combination operators, let Q be a DAQ, and $k \geqslant 0$. Then,*

(i) *Algorithm k-Rank correctly computes a k-rank answer to Q over KB; and*
(ii) *if O is guarded, then Algorithm k-Rank can be done in polynomial time in the size of D and S, where S is the cost of computing $\Pr_M(a)$ for any ground atom a with $O \models a$.*

Proof (i) Correctness follows immediately from the definition of k-rank answers: 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 11 ensures that at most k results are returned.

(ii) As O is guarded, the necessary finite part C of the chase of O relative to Q is of polynomial size and computable in polynomial time in the size of D [6]. By Theorems 11 and 12, all above four combination operators $\otimes$ are possible in polynomial time in the size of $\succ'_P$ and S. In line 6, *computeSkyline* is possible in polynomial time by a linear-time scan of C, assuming suitable data structures, and the results can be removed by another linear-time scan. □

As a corollary of Theorem 13 (ii), if probabilities in M can be computed (exactly or approximately) in polynomial time in the data complexity, then so can k-rank answers.

Corollary 3 *Let $KB = (O, P, M, \otimes)$, with $O = (D, \Sigma)$, be a guarded PP-Datalog+/– ontology, where $\otimes$ is any of the above four combination operators, let Q be a DAQ, and $k \geqslant 0$. Let $\Pr_M(a)$ be computable (exactly or approximately) in polynomial time in the data complexity for any ground atom a with $O \models a$. Then, Algorithm k-Rank can be done in polynomial time in the data complexity*

7 A Use Case

We now describe a real-world application of our formalism.

Data and probabilistic model The raw data were obtained from the Internet Movie Database (IMDB)[1]; the resulting database consists of 13,893 movies—the structure of the used ontology is described in Fig. 12. Furthermore, IMDB associates a rating with each movie (computed as the average opinion of the users who give their input). Table 2 shows a small subset of the movie database, which was selected for this use case. The ratings range from 0 to 1; probability values, which adequately model the opinions of the users about how good the movie is, were computed as the cumulative density function of a Gaussian distribution with mean 0.635 and standard deviation 0.126 (the population values corresponding to the entire database of 13,893 movies).

User preferences In the following, suppose a user has declared the following set of preferences, which we express as an ordered set of preference formulas (very similar to the approach introduced in [7] for preferences in relational databases). If a_1 and a_2 are atoms, a *preference formula pf* is of the form "$a_1 \succ a_2$ *if* $C(a_1, a_2)$", where $C(a_1, a_2)$ is a first-order formula. We call $C(a_1, a_2)$ the *condition* of *pf*, denoted *cond*(*pf*). Suppose the following preference formulas have been acquired for a given user:

$$\begin{aligned} pf_1: \quad & movie(M_1) \succ movie(M_2) \\ & \textit{if } drama(M_1) \wedge drama(M_2) \\ & \wedge hasActor(M_2, A_2) \wedge actor(A_2, brendan, fraser) \\ & \wedge \neg hasActor(M_1, A_2). \end{aligned}$$

Drama movies without Brendan Fraser are preferred over those with him.

[1] http://www.imdb.com.

Movie Ontology Predicates:

$\mathit{movie}(I)$: Denotes a movie with identifier I — same schema for predicates corresponding to movie categories (such as “action”, “animation”, etc.).

$\mathit{actor}(I, F, L, G)$: Denotes an actor with identifier I, first name F, last name L, and gender G.

$\mathit{director}(I, F, L)$: Denotes a director with identifier I, first name F, and last name L.

$\mathit{hasActor}(M, A)$: Denotes that the movie with identifier M has the actor with identifier A.

$\mathit{hasDirector}(M, D)$: Denotes that the movie with identifier M has the director with identifier D.

Tuple-Generating Dependencies:

$$\begin{array}{llll}
\Sigma_T = \{ & \mathit{movie}(M) \rightarrow \exists D\, \mathit{hasDirector}(M, D), & \mathit{movie}(M) \rightarrow \exists A\, \mathit{hasActor}(M, A), & \mathit{actor}(A, F, L, G) \rightarrow \exists M\, \mathit{hasActor}(M, A), \\
& \mathit{director}(D, F, L) \rightarrow \exists M\, \mathit{hasActor}(M, D), & \mathit{action}(I) \rightarrow \mathit{movie}(I), & \mathit{adventure}(I) \rightarrow \mathit{movie}(I), \\
& \mathit{animation}(I) \rightarrow \mathit{movie}(I), & \mathit{biography}(I) \rightarrow \mathit{movie}(I), & \mathit{comedy}(I) \rightarrow \mathit{movie}(I), \\
& \mathit{crime}(I) \rightarrow \mathit{movie}(I), & \mathit{documentary}(I) \rightarrow \mathit{movie}(I), & \mathit{drama}(I) \rightarrow \mathit{movie}(I), \\
& \mathit{family}(I) \rightarrow \mathit{movie}(I), & \mathit{fantasy}(I) \rightarrow \mathit{movie}(I), & \mathit{film\_noir}(I) \rightarrow \mathit{movie}(I), \\
& \mathit{history}(I) \rightarrow \mathit{movie}(I), & \mathit{horror}(I) \rightarrow \mathit{movie}(I), & \mathit{music}(I) \rightarrow \mathit{movie}(I), \\
& \mathit{musical}(I) \rightarrow \mathit{movie}(I), & \mathit{mystery}(I) \rightarrow \mathit{movie}(I), & \mathit{romance}(I) \rightarrow \mathit{movie}(I), \\
& \mathit{sci\_fi}(I) \rightarrow \mathit{movie}(I), & \mathit{sport}(I) \rightarrow \mathit{movie}(I), & \mathit{thriller}(I) \rightarrow \mathit{movie}(I), \\
& \mathit{war}(I) \rightarrow \mathit{movie}(I), & \mathit{western}(I) \rightarrow \mathit{movie}(I)\}. &
\end{array}$$

Fig. 12 Description of the movie ontology used in the use case and experimental evaluation

Table 2 Small subset of movies taken from the database to illustrate the use case

ID	Title	Actors	Director	Year	Genre	Rating	Probability
m_1	X-Men: First Class	M. Fassbender, J. McAvoy	B. Singer	2011	Action/Sci-fi	0.80	0.90
m_2	Haywire	E. McGregor, M. Fassbender	S. Soderbergh	2011	Action/Thriller	0.60	0.39
m_3	Gravity	G. Clooney, S. Bullock	A. Cuarón	2013	Action/Drama	0.85	0.96
m_4	Batman & Robin	G. Clooney, A. Schwarzenegger	J. Schumacher	1997	Action/Fantasy	0.40	0.03
m_5	The Prestige	S. Johansson, H. Jackman	C. Nolan	2006	Mystery	0.80	0.90
m_6	Crash	S. Bullock, B. Fraser	P. Haggis	2004	Drama	0.60	0.39
m_7	Elysium	M. Damon, J. Foster	N. Blomkamp	2013	Sci-fi/Action	0.68	0.64
m_8	Savages	B. Lively, T. Kitsch	O. Stone	2012	Adventure/Crime	0.70	0.69
m_9	2001: A Space Odyssey	K. Dullea, G. Lockwood	S. Kubrick	1968	Sci-fi	0.80	0.90
m_{10}	Eyes Wide Shut	T. Cruise, N. Kidman	S. Kubrick	1999	Drama	0.70	0.69
m_{11}	A Clockwork Orange	M. MacDowell, P. Magee	S. Kubrick	1971	Sci-fi	0.90	0.98
m_{12}	Ghost Busters II	B. Murray, D. Aykroyd	I. Reitman	1989	Action/Sci-fi	0.60	0.39
m_{13}	Forrest Gump	T. Hanks, R. Wright	R. Zemeckis	1994	Drama/Romance	0.90	0.98
m_{14}	Dragnet	T. Hanks, D. Aykroyd	T. Mankiewicz	1987	Comedy	0.60	0.39
m_{15}	Inception	L. DiCaprio, J. Gordon-Levitt	C. Nolan	2010	Action/Sci-fi	0.90	0.98

$$\begin{array}{ll}
\mathit{pf}_2: & \mathit{movie}(M_1) \succ \mathit{movie}(M_2) \\
& \mathit{if}\ \mathit{actor}(A_1, \mathit{matt}, \mathit{damon}) \wedge \mathit{actor}(A_2, \mathit{george}, \mathit{clooney}) \\
& \wedge (\mathit{hasActor}(M_1, A_1) \vee \mathit{hasActor}(M_1, A_2)) \\
& \wedge \neg(\mathit{hasActor}(M_2, A_1) \vee \mathit{hasActor}(M_2, A_2)).
\end{array}$$

Movies starring George Clooney or Matt Damon are preferred over movies without them.

$$\begin{array}{ll}
\mathit{pf}_3: & \mathit{movie}(M_1) \succ \mathit{movie}(M_2) \\
& \mathit{if}\ \mathit{hasActor}(M_1, A) \wedge \mathit{hasActor}(A, \mathit{michael}, \mathit{fassbender}) \\
& \wedge \neg \mathit{hasActor}(M_2, A).
\end{array}$$

Movies starring Michael Fassbender are preferred over all others.

$$\begin{array}{ll}
\mathit{pf}_4: & \mathit{movie}(M_1) \succ \mathit{movie}(M_2) \\
& \mathit{if}\ \mathit{hasActor}(M_1, A_1) \wedge \mathit{hasActor}(A_1, \mathit{scarlett}, \mathit{johansson}) \\
& \wedge \mathit{hasActor}(M_2, A_2) \wedge \mathit{hasActor}(A_2, \mathit{leonardo}, \mathit{dicaprio}).
\end{array}$$

Movies starring Scarlett Johansson are preferred over those starring Leonardo DiCaprio.

$$\begin{array}{ll}
\mathit{pf}_5: & \mathit{movie}(M_1) \succ \mathit{movie}(M_2) \\
& \mathit{if}\ (\mathit{action}(M_1) \vee \mathit{sci\_fi}(M_1)) \wedge \mathit{drama}(M_2).
\end{array}$$

Action and science fiction movies are preferred over dramas.

$$\begin{array}{ll}
\mathit{pf}_6: & \mathit{movie}(M_1) \succ \mathit{movie}(M_2) \\
& \mathit{if}\ \mathit{sci\_fi}(M_1) \wedge \mathit{sci\_fi}(M_2) \\
& \wedge \mathit{hasDirector}(M_1, D_1) \wedge \mathit{director}(D_1, \mathit{stanley}, \mathit{kubrick}) \\
& \wedge \mathit{hasDirector}(M_2, D_2) \wedge \mathit{director}(D_2, \mathit{ivan}, \mathit{reitman}).
\end{array}$$

Science fiction movies directed by Stanley Kubrick are preferred over those directed by Ivan Reitman.

$$\begin{array}{ll}
\mathit{pf}_7: & \mathit{movie}(M_1) \succ \mathit{movie}(M_2) \\
& \mathit{if}\ \mathit{hasDirector}(M_1, D_1) \wedge \mathit{director}(D_1, \mathit{stanley}, \mathit{kubrick}) \\
& \wedge \mathit{hasDirector}(M_2, D_2) \wedge \mathit{director}(D_2, \mathit{oliver}, \mathit{stone}).
\end{array}$$

Movies directed by Stanley Kubrick are preferred over those directed by Oliver Stone.

To avoid cyclic preferences, we assume that these preference formulas are applied in the given order, and only apply if no cycles are introduced. Figure 13 shows the induced preference relation over the set of movies in Table 2.

Results To illustrate the different behaviors of the preference combination operators of Sect. 4, we study the answers to the query $Q(X) = movie(X)$ obtained via Algorithm ComPrefsRank with each one—the results are shown in Table 3. For each rank i from 1 to 13, the table shows the set of IDs corresponding to movies (cf. Table 2) that have rank i in the graph resulting from merging the original user SPO with the one induced by the probabilistic model using several combination operators and parameter values.

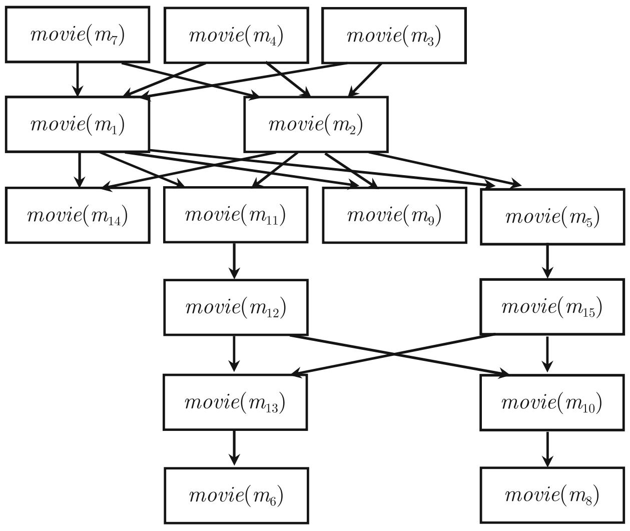


Fig. 13 The user's SPO modeled by preference formulas pf_1 to pf_7

For example, using the combination operator from Algorithm 2 with $f = min$, the table shows that movies m_1 and m_2 have rank 2 in the combined SPO. One possible 5-rank answer to $Q(X)$ using Algorithm 2 with function $f = min$ is $\langle m_3, m_4, m_7, m_{13}, m_{15}\rangle$.

This table shows interesting differences in the results obtained using each combination operator with different parameters. For instance, movie m_3 receives rank 1 in all cases, since it is one of the user's favorites, and also has a very high probability of 0.96. However, note that for m_4, which is another of the user's favorites, the behavior of each algorithm is quite different. Algorithm 1 (with both $t = 0.15$ and $t = 0.30$) assigns rank 3 to this movie, which is a compromise between the fact that the user ranked it very high, but its probability is almost zero. On the other hand, Algorithm 2 has two variants depending on the input function f: with min, the algorithm makes optimistic choices and ranks high movies that are either ranked high by the user or have high probability—in this case, m_4 is assigned the top rank. With $f = max$, the algorithm behaves pessimistically by assigning the worst rank possible between the user and the probability—since m_4 is the lowest ranking movie relative to probability, it receives the worst rank in the result.

Algorithm 3 uses the probability values to purge the user's SPO, using as a result the structure arising from the deletion of the movies that do not surpass the threshold. Note that movie m_4 thus disappears from all possible answers. Another interesting observation is that with $p = 0.95$, only the highest probability choices are kept—however, the user's preferences are still visible in the resulting structure. For instance, movie m_{15} climbed a position to join m_{11}, because the ones dominating it (m_9 and m_5) were removed; however, m_{13} is still less preferable than m_{15} and m_{11}.

Table 3 Answers to the query $Q(X) = movie(X)$ for several different combinations of operators and parameter values, using the set of movies shown in Table 2 and the SPO in Fig. 13

Rank	Alg. 1 ($t = 0.15$)	Alg. 1 ($t = 0.3$)	Alg. 2 ($f = min$)	Alg. 2 ($f = max$)	Alg. 3 ($p = 0.4$)	Alg. 3 ($p = 0.95$)	Alg. 4
1	$\{m_3\}$	$\{m_3, m_7\}$	$\{m_3, m_4, m_7, m_{11}, m_{13}, m_{15}\}$	$\{m_3\}$	$\{m_7, m_3\}$	$\{m_3\}$	$\{m_3\}$
2	$\{m_1\}$	$\{m_1\}$	$\{m_1, m_2\}$	$\{m_1, m_5, m_9, m_{11}\}$	$\{m_1\}$	$\{m_{11}, m_{15}\}$	$\{m_7\}$
3	$\{m_7, m_4\}$	$\{m_4\}$	$\{m_5, m_{14}, m_9\}$	$\{m_{15}\}$	$\{m_5, m_9, m_{11}\}$	$\{m_{13}\}$	$\{m_4\}$
4	$\{m_9, m_{11}\}$	$\{m_9, m_{11}, m_5\}$	$\{m_8, m_{10}, m_{12}\}$	$\{m_7\ m_{10}, m_{13}\}$	$\{m_{15}\}$		$\{m_1\}$
5	$\{m_5\}$	$\{m_2\}$	$\{m_6\}$	$\{m_2, m_6, m_8, m_{12}, m_{14}\}$	$\{m_{13}, m_{10}\}$		$\{m_2\}$
6	$\{m_2\}$	$\{m_{14}, m_{15}\}$		$\{m_4\}$	$\{m_8\}$		$\{m_{11}\}$
7	$\{m_{14}, m_{15}\}$	$\{m_{13}\}$					$\{m_9, m_5\}$
8	$\{m_{10}, m_{13}\}$	$\{m_{12}\}$					$\{m_{14}, m_{15}\}$
9	$\{m_{12}\}$	$\{m_{10}, m_6\}$					$\{m_{12}\}$
10	$\{m_8, m_6\}$	$\{m_8\}$					$\{m_{13}\}$
11							$\{m_{10}\}$
12							$\{m_8\}$
13							$\{m_6\}$

The answers are shown according to their rank in the graph resulting from merging the original SPO with the probability-based SPO

Finally, the behavior of Algorithm 4 is quite clear: it uses the probability values to sort the iterated skylines of the user's preferences—the only non-singleton sets in the rank assignments thus only arise when ties occur in probability values. This kind of answer is perhaps most useful when the user desires to have a fully ordered list of choices; on the other hand, the other algorithms may be most applicable to the presentation of answers in a layered fashion, such as an online system that shows movie suggestions by page.

8 Experimental Evaluation

In this section, we evaluate and analyze the running time of our algorithms over a combination of real-world and synthetic data. We first describe the experimental setup and then continue to discuss the obtained results.

8.1 Implementation and Hardware

We implemented our framework and algorithms by extending the Datalog+/– query answering engine in [13], which supports FO-rewritable fragments of the Datalog+/– family of ontology languages. All graph operations were implemented using the JGraphT[2] library, which provides efficient data structures for the representation of graph structures, as well as efficient implementations of operations such as reachability, transitive closure, and cycle detection. The whole code was written in Java

All runs were done on a laptop with an Intel Core i5 processor at 2.6 GHz and 16GB RAM, under the Windows 7 Professional (64-bit) SP1 operating system (Build 7601) and a Sun JVM Standard Edition with maximum heap size set to 14 GB RAM. To minimize experimental variation, all results are averages of (between 5 and 10) independent runs.

8.2 Experimental Setup

Inputs to our system consist of tuples $\langle Q, KB, k\rangle$, where Q is a query, $KB = (O, P, M, \otimes)$ is a PP-Datalog+/– ontology, and k is the number of query results to be sorted according to the user's preferences. The preference graph is a labeled directed graph (N, E), where N is the node set (the atoms), and E is the edge set (the SPO). A preference-augmented chase is then used for obtaining the answer to the query.

- *Data* All runs were carried out using the ontology built on the basis of the IMDB dataset, as described in Sect. 7. To test our algorithms on instances of different sizes, we considered different subsets of the dataset, varying this parameter from 1,000 to 13,000 nodes (movies). Finally, the database for this ontology was stored in a PostgreSQL 9.3 database.
- *User preferences* User preference graphs (SPOs) were randomly generated by first creating a set of nodes of the required size and then adding a certain number of edges. For the latter, we used different values of a *density* parameter to set the size of E—this simply refers to the number of edges as a percentage of $|N| \cdot (|N|-1)/2$ (the maximum possible number of edges in a DAG). For transitively closed graphs (see discussion below), we used a seed density manually tuned to obtain the target number of edges after the transitive closure. The averages and standard deviations for each set of independent runs yielding a data point are reported in Tables 4 and 5.
- *Probabilistic model* As described in Sect. 7, the IMDB dataset provides a numerical rating ranging from 0 to 10; to obtain a probability, these values were normalized using a Gaussian distribution with population values of 0.635 (mean) and 0.126 (standard deviation).
- *Query* We used the query $Q(X) = movie(X)$ for all runs. This query represents a user requesting a set of k movies sorted according to a combination of own preferences and the probability that each movie is good (according to the reviews available on IMDB).

Finally, in contrast to the general analysis of the worst-case running times in Sect. 5, these experiments focus on the special case of computing k-rank query answers. As such,

Table 4 Information on the input graphs for Experiment 1

# of nodes	Avg. # of edges	St. dev.	Avg. density (%)	Seed density (%)
1,000	940.8	604.01	0.19	0.1
2,000	10,520.8	943.89	0.53	0.1
3,000	73,728.0	6,091.80	1.64	0.1
4,000	275,681.2	10,503.84	3.45	0.1
5,000	781,634.8	38,602.54	6.25	0.1
6,000	1,672,068.0	35,328.19	9.29	0.1
7,000	3,167,134.0	84,153.84	12.93	0.1

Table 5 Information on the input graphs for Experiment 2

# of nodes	Avg. # of edges	St. dev.	Avg. density (%)	Seed density (%)
2,000	92,918.9	7,728.17	4.64	0.20
2,000	178,938.9	9,917.13	8.95	0.25
2,000	271,800.7	13,082.45	13.60	0.30
2,000	356,077.3	15,205.12	17.81	0.35
2,000	463,820.2	15,216.49	23.20	0.40

[2] http://jgrapht.org/.

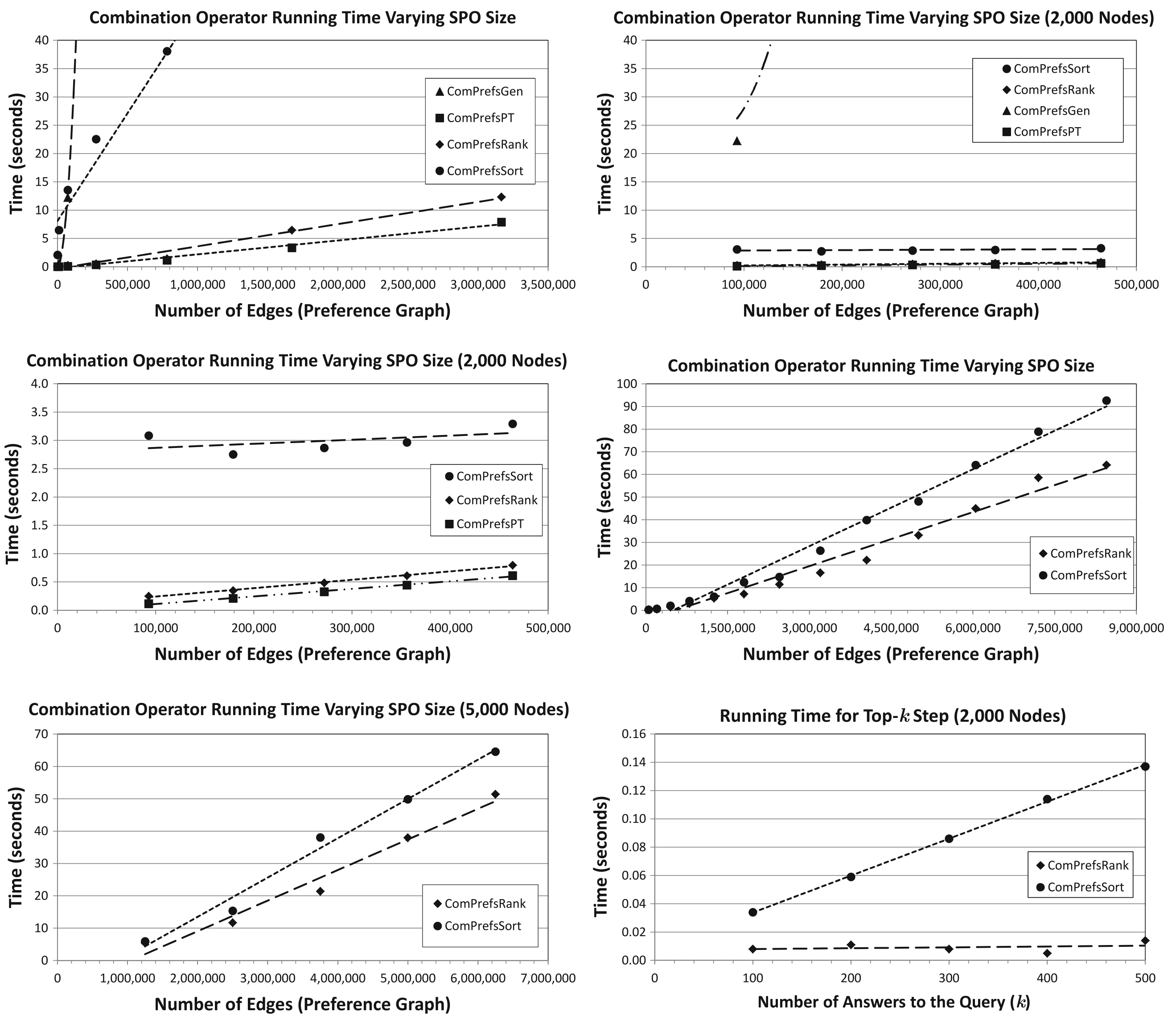


Fig. 14 Experimental results over the IMDB dataset and randomly generated SPOs. Each data series is augmented with the corresponding polynomial trend line with respect to the values in the x axis (cf. Sect. 4)

we may resort to a simple (but in practice very effective) optimization of the operators that involves not computing the transitive closure as the final step. Though this is necessary in theory to obtain an SPO as the result of the combination operation, the information added by the transitive closure is superfluous for Algorithm k-Rank.

8.3 Results

To test the performance of our algorithms, we carried out experiments varying several parameters, namely, (i) SPO size and (ii) the number of answers (k). As for (i), since the size of an SPO represented as a directed graph is given both by the number of nodes and the number of edges, our experiments show the effects of varying these two parameters.

Experiment 1: Varying the number of nodes in the SPO Fig. 14 (top, left side) shows the running times of all four combination operators when varying the number of nodes from 1,000 to 13,000 with seed density of 0.1% and k fixed at 200—the graph is plotted with number of edges on the x axis, since this parameter is the one that most directly affects the performance of the algorithms. Table 4 shows, for each data point, the number of nodes, average number of edges, standard deviation, average density, and seed density. Clearly, ComPrefsGen is greatly outperformed by the rest of the algorithms, while ComPrefsSort is the next worse. This is due to the fact that ComPrefsGen involves a pairwise comparison of all nodes (ground atoms) in the graph and, in each case, a check for potential cycles. On the other hand, ComPrefsSort suffers when the number of nodes increases, as it involves sorting each iterated skyline, which were quite large for these relatively low-density graphs.

Experiment 2: Varying the edge density in the SPO. Next, we fixed the number of nodes in the SPO at 2,000, varied the density parameter from 0.2 to 0.4 %, and fixed k to 200; Table 5 shows the rest of the information for this experiment. The results can be seen in Fig. 14 (top, right side). As in the previous experiment, we can see that ComPrefsGen is outperformed by the rest; however, in this case, ComPrefsSort performs comparably to ComPrefsRank and ComPrefsPT, since the number of nodes to sort is fixed in this case. Figure 14 (middle, left side) shows the comparison without ComPrefsGen to illustrate the differences among the other three operators more clearly.

Experiment 3: Varying the number of nodes in the SPO (high edge density) (a) Fig. 14 (middle, right side) corresponds to the same setup of Experiment 1 with one important difference: the SPOs generated as inputs to the operators are not transitively closed. Though, by definition, this is required, algorithms ComPrefsRank and ComPrefsSort do not actually use this information—it is highly redundant and greatly hinders the performance of the algorithms. In this plot, we can see that these two operators remain quite scalable even for graphs with over 8M edges, which represents a much larger graph if it were transitively closed.
(b) As before, we also ran another experiment in which the number of nodes was set to 5,000, and the density varied from 10% to 50% (also without transitive closure)—the results are shown in Fig. 14 (bottom, left side).

Tables 6 and 7 show, for each data point, the number of nodes, number of edges, density, and seed density.

Table 6 Information on the input graphs for Experiment 3 (a)

# of nodes	# of edges	Density (%)	Seed density (%)
1,000	49,950	10	10
2,000	199,900	10	10
3,000	449,850	10	10
4,000	799,800	10	10
5,000	1,249,750	10	10
6,000	1,799,700	10	10
7,000	2,449,650	10	10
8,000	3,199,600	10	10
9,000	4,049,550	10	10
10,000	4,999,500	10	10
11,000	6,049,450	10	10
12,000	7,199,400	10	10
13,000	8,449,350	10	10

Table 7 Information on the input graphs for Experiment 3 (b)

# of nodes	# of edges	Density (%)	Seed density (%)
5,000	1,249,750	10	10
5,000	2,499,500	20	20
5,000	3,749,250	30	30
5,000	4,999,000	40	40
5,000	6,248,750	50	50

Experiment 4: Varying k Fig. 14 (bottom, right side) shows the running time for the computation of the k-rank answers without taking into account the combination operation; the number of nodes was fixed at 2,000 and density at 20%. For this experiment, we only show ComPrefsRank and ComPrefsSort, as they are mostly affected by the density of edges in the SPO. If the graph is too sparse, the first skyline usually contains all k-rank answers—in the runs corresponding to the previous lower-density experiments, the running time for the k-rank computation was negligible.

9 Related Work

Preferences have received much attention in many areas of study such as philosophy, choice theory, and certain areas of the social sciences (such as social choice). In computer science, the most relevant to our work is their incorporation into query answering mechanisms. To our knowledge, the current state of the art in this respect is centered around relational databases, and no other work to date combines general preferences with those induced by probability assignments.

The seminal work in preference-based query answering is that of [19], in which the SQL language is extended to incorporate user preferences, showing that the resulting formalism can be translated into the domain relational calculus. In [7], preference formulas are introduced as a logical formalism that allows an embedding of preference specifications into SQL through a *winnow* operator parameterized by a preference formula; the winnow operator is a generalization of the *skyline* operator, first introduced in [4]. Perhaps closest to our approach (except for the probabilistic model) is that of preference Datalog programs [15], which are a restriction of preference logic programs [14] that contain no uninterpreted function symbols and extend classical Datalog with constructs for determining which predicates must be optimized along with the optimization criteria (i.e., the set of preferences). For a recent survey on preference-based query answering, see [30].

With respect to probabilistic preferences, there are several works that have been developed in the last few years.

Probabilistic skylines were introduced in [26] (and later also studied in [1]; the related *stochastic* skyline is introduced in [20]), where the authors tackle the problem of computing the probability of an element belonging to the skyline, rather than using the probability values for the purpose of ranking, as proposed in our work. In [29,34], the authors focus on a more specific version of our problem, since they assume that tuples receive both a score and a probability value, and thus the two preference relations in question are score-based. Furthermore, the approach adopted in their work is based on possible worlds, whereas here we focus on the use of probabilities solely as vehicles for ranking.

Preference revision is also a closely related area. The work of [16] tackles the problem of modeling preference change, studying axioms and postulates for the revision and contraction of a set of sentences specifying preferences by an input preference, in the style of belief revision theory; addition and subtraction of elements are also studied. The main difference to our work is that the author does not study algorithms or complexity of deriving a revised relation, but rather focuses on a set of postulates based on those from belief revision and centered on obtaining a relation that incorporates the new preference with minimal change. The work of [8] is also related in that it addresses the problem of *query modification* in preference-based query answering in relational DBs. The focus is, however, on three specific combination mechanisms: union, prioritized, and Pareto composition, and on the study of the preservation of properties of different kinds of relations under these combinations.

Social choice theory [11] is also relevant, since it seeks to combine preferences to produce a new preference relation; methods range from those using score-based relations (e.g., approval voting) to ones using more general ones (e.g., ranked pairs). In particular, the work of [27] studies results with respect to possibility and impossibility, which generalize properties such as Arrow's theorem to contemplate cases in which incomparable elements exist.

10 Conclusion

In this work, we have presented an extension of the Datalog+/– family of ontology languages for preference-based query answering under uncertainty. This task has recently attained central importance due to its relation with the Social (Semantic) Web. The main focus of this work has been on defining preference combination operators, which produce a preference relation, given a general SPO and a score-based SPO. We have proposed four specific algorithms for such an operator and analyzed their semantic and computational properties. Finally, we have studied a basic algorithm for answering k-rank queries and showed that under certain conditions, k-rank queries can be answered in polynomial time in the data complexity, which is the same complexity as answering traditional preference-based queries in relational DBs. We have also reported on an implementation and experimental results. Future work involves developing other combination operators and studying specific preference specification mechanisms and probabilistic models.

Acknowledgments 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 paper and of its ODBASE-2013 preliminary version [24] for their useful feedback, as well as to Giorgio Orsi for his help with the Datalog+/– query answering engine.

References

1. Atallah MJ, Qi Y (2009) Computing all skyline probabilities for uncertain data. In: Proceedings of PODS. ACM Press, New York, pp 279–287
2. Beeri C, Vardi MY (1987) The implication problem for data dependencies. In: Proceedings of ICALP. Springer, Berlin, pp 73–85
3. Berners-Lee T, Hendler J, Lassila O (2001) The semantic web. Sci Am 284(5):34–43
4. Börzsönyi S, Kossmann D, Stocker K (2001) The skyline operator. In: Proceedings of ICDE. IEEE Computer Society, Los Alamitos, pp 421–430
5. Calì A, Gottlob G, Kifer M (2008) Taming the infinite chase: query answering under expressive relational constraints. In: Proceedings of KR. AAAI Press, Menlo Park, pp 70–80
6. Calì A, Gottlob G, Lukasiewicz T (2012) A general Datalog-based framework for tractable query answering over ontologies. J Web Sem 14:57–83
7. Chomicki J (2003) Preference formulas in relational queries. ACM Trans Database Syst 28(4):427–466
8. Chomicki J (2007) Database querying under changing preferences. Ann Math Artif Intell 50(1/2):79–109
9. Domingos P, Webb WA (2012) A tractable first-order probabilistic logic. In: Proceedings of AAAI. AAAI Press, Menlo Park, pp 1902–1909
10. Finger M, Wassermann R, Cozman FG (2011) Satisfiability in $\mathcal{EL}$ with sets of probabilistic ABoxes. In: Proceedings of DL
11. Gaertner W (2009) A primer in social choice theory: revised edition. Oxford University Press, Oxford
12. Gottlob G, Lukasiewicz T, Martinez MV, Simari GI (2013) Query answering under probabilistic uncertainty in Datalog+/- ontologies. Ann Math Artif Intell 69(1):37–72
13. Gottlob G, Orsi G, Pieris A (2011) Ontological queries: Rewriting and optimization. In: Proceedings of ICDE. IEEE Computer Society, Washington, DC, pp 2–13
14. Govindarajan K, Jayaraman B, Mantha S (1995) Preference logic programming. In: Proceedings of ICLP. MIT Press, Cambridge, pp 731–745
15. Govindarajan K, Jayaraman B, Mantha S (2001) Preference queries in deductive databases. New Generat Comput 19(1):57–86
16. Hansson SO (1995) Changes in preference. Theory Decis 38:1–28
17. Jung JC, Lutz C (2012) Ontology-based access to probabilistic data with OWL QL. In: Proceedings of ISWC. Springer, Berlin, pp 182–197

18. Kim JH, Pearl J (1983) A computational model for causal and diagnostic reasoning in inference systems. In: Proceedings of IJCAI. William Kaufmann, Karlsruhe, pp 190–193
19. Lacroix M, Lavency P (1987) Preferences: putting more knowledge into queries. In: Proceedings of VLDB. Morgan Kaufmann, Burlington, pp 1–4
20. Lin X, Zhang Y, Zhang W, Cheema MA (2011) Stochastic skyline operator. In: Proceedings of ICDE. IEEE Computer Society, pp 721–732
21. Lukasiewicz T, Martinez MV, Orsi G, Simari GI (2012) Heuristic ranking in tightly coupled probabilistic description logics. In: Proceedings of UAI. AUAI, Edinburgh, pp 554–563
22. Lukasiewicz T, Martinez MV, Simari GI (2012) Consistent answers in probabilistic Datalog+/- ontologies. In: Proceedings of RR. Springer, Berlin, pp 156–171
23. Lukasiewicz T, Martinez MV, Simari GI (2013) Preference-based query answering in Datalog+/- ontologies. In: Proceedings of IJCAI. AAAI Press / IJCAI, Menlo Park, pp 1017–1023
24. Lukasiewicz T, Martinez MV, Simari GI (2013) Preference-based query answering in probabilistic Datalog+/- ontologies. In: Proceedings of ODBASE. Springer, Berlin, pp 501–518
25. Noessner J, Niepert M (2011) ELOG: A probabilistic reasoner for OWL EL. In: Proceedings of RR. Springer, Berlin, pp 281–286
26. Pei J, Jiang B, Lin X, Yuan Y (2007) Probabilistic skylines on uncertain data. In: Proceedings of VLDB. ACM Press, New York, pp 15–26
27. Pini MS, Rossi F, Venable KB, Walsh T (2009) Aggregating partially ordered preferences. J Log Comput 19(3):475–502
28. Richardson M, Domingos P (2006) Markov logic networks. Mach Learn 62(1/2):107–136
29. Soliman MA, Ilyas IF, Chen-Chuan Chang K (2007) Top-k query processing in uncertain databases. In: Proceedings of ICDE. IEEE Computer Society, pp 896–905
30. Stefanidis K, Koutrika G, Pitoura E (2011) A survey on representation, composition and application of preferences in database systems. ACM Trans Database Syst 36(3):19:1–19:45
31. Warren HS Jr (1975) A modification of Warshall's algorithm for the transitive closure of binary relations. Commun ACM 18(4):218–220
32. Warshall S (1962) A theorem on Boolean matrices. J ACM 9(1):11–12
33. Zhang X (2010) Probabilities and sets in preference querying. Ph.D. thesis, University at Buffalo, State University of New York
34. Zhang X, Chomicki J (2009) Semantics and evaluation of top-k queries in probabilistic databases. Distrib Parallel Dat 26:67–126