Preferential Regular Path Queries

Gösta Grahne[1], Alex Thomo[2], and William Wadge[2]

[1] Concordia University, Montreal, Canada, grahne@cs.concordia.ca
[2] University of Victoria, Victoria, Canada, {thomo,wwadge}@cs.uvic.ca

Abstract. In this paper, we introduce preferential regular path queries. These are regular path queries whose symbols are annotated with preference weights for "scaling" up or down the intrinsic importance of matching a symbol against a (semistructured) database edge label. Annotated regular path queries are expressed syntactically as annotated regular expressions. We interpret these expressions in a uniform semiring framework, which allows different semantic interpretations for the same syntactic annotations. For our preference queries, we study three important aspects: (1) (progressive) query answering (2) (certain) query answering in LAV data-integration systems, and (3) query containment and equivalence. In all of these, we obtain important positive results, which encourage the use of our preference framework for enhanced querying of semistructured databases.

1 Introduction

Regular path queries are one of the basic building blocks of virtually all the mechanisms for querying *semistructured data*, commonly found in information integration applications, Web and communication networks, biological data management, etc. Semistructured data is conceptualized as edge-labeled graphs, and regular path queries are in essence regular expressions over the edge symbols. The answer to a regular path query on a given graph (database) is the set of pairs of objects, which are connected by paths spelling words in the language of the regular path query.

Seen from a different angle, regular path queries provide the user with a simple way of expressing *preferences* for navigating database paths. Let us take an example from road network databases. Suppose that the user wants to retrieve all the pairs of objects preferentially connected by highways, and tolerating up to k provincial roads or city streets. Clearly, such preferences can easily be captured by the regular path query

$$Q = \mathit{highway}^* \parallel (\mathit{road} + \mathit{street} + \epsilon)^k,$$

where $\parallel$ is the shuffle operator (see e.g. [18]).

It is exactly this ability of regular expressions to capture pattern preferences that has made them very popular, starting from the early days of computers. However, let us take a more careful look at the above example. It surely captures the user preferences, but in a "Boolean" way. A pair of objects will be produced as an answer if there exists a path between them satisfying the user query. In other words, there is just a "yes" or "no" qualification for the query answers. But, the answers are not equally good! A pair of objects connected by a *highway* path with only 1 intervening *road* is obviously a "better" answer than a pair of objects connected by a *highway* path with 5 intervening *road*s.

Clearly, preferences beyond the "Boolean" ones cannot be captured by simple regular path queries.

In this paper, we introduce *preferentially annotated regular path queries*, which are regular path queries (regular expressions) with a very simple syntactic addition: the user can annotate the symbols in the regular expressions with "markers" (typically natural numbers), which "strengthen" or "weaken" her (pattern) preferences. For example, she can write

$$Q = (\mathit{highway} : 0)^* \parallel (\mathit{road} : 1 + \mathit{street} : 2 + \epsilon)^k,$$

to express that she ideally prefers highways, then roads, which she prefers less, and finally she can tolerate streets, but with an even lesser preference. Given such a query, the system should produce first the pairs of objects connected by highways, then the pairs of objects connected by highways intervened by 1 road, and so on.

The above "so on" raises some important semantical questions. Is a pair of objects connected by a highway path intervened by two roads equally good as another pair of objects connected by a highway path intervened by one street only? Indeed, in this example, it might make sense to consider them equally good, and "concatenate" weights by summing them up.

However, let us consider another example regarding travel itineraries. Assume that the preferentially annotated user query is

$$Q = (viarail : 0)^* \parallel (greyhound : 1 + aircanada : 2 + \epsilon)^k.$$

Is now a pair of objects connected by a path with two *greyhound* segments equally preferable as a pair of objects connected with one *aircanada* segment? Here the answer is not clear anymore. If the user is afraid of flying, she might want to "concatenate" edge-weights by choosing the maximum of the weights. Then an itinerary with no matter how many *greyhound* segments is preferable to an itinerary containing only one flight segment.

We say that in the first case the preference semantics are "quantitative," while in the second case they are "qualitative." We study both semantics for regular path queries, and leave the choice as an option specified by the user during query time.

We also consider another choice of semantics, which is a hybrid between the quantitative and qualitative semantics. Continuing the travel itinerary example, by following a purely qualitative approach, *greyhound* itineraries are always preferable to itineraries containing *aircanada* segments, while these itineraries are equally preferable, no matter how many lags the flight has. Although, there might be applications where such qualification is all what is needed, in the particular example we need to distinguish among itineraries on the same "level of discomfort." Namely, we should be able to (quantitatively) say for example that a direct *aircanada* route is preferable to an *aircanada* route with a stop-over, which again is preferable to an *aircanada* route with three lags. Notably, such user preferences can concisely be captured by our hybrid semantics.

In total, from all the above, we have four kind of preference semantics: Boolean, quantitative, qualitative, and hybrid. Other semantics can also be proposed, tailored to specific applications. In all these semantics, we aggregate ("concatenate") preference markers or weights along edges of the paths, and then we aggregate path preferences when there are multiple paths connecting a pair of objects. Hence, we regard the preference annotations as elements of a semiring, with two operations: the "plus" and "times." The "times" aggregates the preferences along edges of a path, while the "plus" aggregates preferences among paths.

An interesting feature of our preference framework is that for all new semantics (quantitative, qualitative, and hybrid), the syntactic user interface (*i.e.* annotated regular expressions) is exactly the same. After the user writes the query, she also specifies which semantics the system should assume for answering the query. It is straightforward for the user to preferentially annotate regular path queries, and moreover, such annotation can be easily facilitated by system default values.

In this paper, we study three important aspects of our preferentially annotated queries. First, we focus on query answering and design a progressive algorithm, which produces the answer tuples in order of their "goodness" with respect to the user preferences. Notably, answering annotated regular path queries is computationally no more difficult than the answering of classical regular path queries. In both cases, a database object is accessed at most once.

Second, we turn our attention to query answering in data integration systems, in which we have only incomplete information about databases. Such systems have been the focus of many studies (cf. [6–8][1]) and reasoning about query answering in this setting is a very important technology.

[1] For the semistructured data case.

We introduce a technique, which we call "query sphering" and show how to progressively compute answer tuples in this variant of incomplete information.

Third, we study query containment and equivalence of preferential regular path queries. We show that containment is undecidable for the quantitative and the hybrid semantics and decidable for qualitative semantics. Then, we present an important class of queries for which the containment is decidable for both quantitative and hybrid semantics.

We would like to note that we introduce the semiring framework in order to unify the formalization of preferences in each case. In this way, we are able to create a template for formally defining new preferences depending on the application. We derive both general results (whenever possible) and specialized results for each case.

The rest of the paper is organized as follows. In Section 2, we overview related work. In Section 3, we introduce the semiring framework for preferentially annotated regular path queries. In Section 4, we give a progressive algorithm for computing the answer to an annotated query. In Section 5, we define and reason about the certain answer to annotated queries in LAV data integration systems. In Section 6, we introduce the concept of query spheres and give a characterization of the certain answer in terms of query spheres. Then, in Section 7, we present algorithms for computing query spheres under the different preference semantics. Section 8 is devoted to containment and equivalence of preferential regular path queries. Finally, Section 9 concludes the paper.

2 Related Work

In relational databases, the most important work on preferences is by Chomicki in a series of papers. One of his recent papers, which gives a detailed overview of the field, is [9]. However, in Chomicki's work, the preference framework is about reasoning on fixed-arity tuples of attribute values. In contrast, here we define "structural" preferences, in the sense that they apply to the paths used for obtaining query answers. Because of this difference, the meaning of our "quantitative" and "qualitative" adjectives is different from the ones mentioned in [9].

Preferences for XML are studied by [19]. These preferences are aimed at comparing attribute values of XML elements rather than structure of (parts of) documents. As characterized by [9], the preferences of [19] seem to largely conform to the relational paradigm.

Regarding our qualitative preferences, they are similar in spirit with constraints in the framework of Infinitesimal Logic studied in [29]. However, [29] focuses on the relational case only.

In [10], weighted path queries are introduced. Syntactically, such queries are the same as our preferentially annotated queries. However, [10] does not give preferential semantics to queries. Notably, in [10] an algorithm is given for answering weighted regular path queries with weights from an arbitrary semiring. This algorithm is based on the Floyd-Warshall algorithm for computing all-pairs shortest paths. On the other hand, we show that our preference semirings have some particular properties that make possible using best-first approaches, which are more efficient for large databases, and which produce the answers in order of their rank.

The quantitative preference semantics in this paper are similar to the distortion semantics in [13]. Regarding query answering (on a given database), one can also use, assuming quantitative semantics only, the algorithm of [13] for queries under distortions. In that paper, there are also some technical results, which can be adapted to help in some of our derivations for the quantitative case. However, we do not do this, due to the high computational complexity of constructs in [13]. Rather, we devise new and better constructs, which are original and can contribute in research regarding formal languages as well. Such research will be mentioned in relevant places during the exposition of the paper.

Finally, [32] and [33] deal with distributed evaluation of weighted regular path queries. However, the algorithms of [32] and [33] apply to quantitative semantics only. We believe that they can also be adapted for other semantics as well, and thus, [32, 33] should be considered to nicely complement this work regarding query answering on distributed databases.

3 Databases and Preferential Regular Path Queries

Databases and classical regular path queries. We consider a database to be an edge-labeled graph. Intuitively, the nodes of the database graph represent objects and the edges represent relationships between the objects.

Formally, let Δ be an alphabet. Elements of Δ will be denoted $r, s, \ldots$. As usual, Δ^* denotes the set of all finite words over Δ. Words will be denoted by $u, w, \ldots$. We also assume that we have a universe of objects, and objects will be denoted $a, b, c, \ldots,$. A *database* DB is then a graph (V, E), where V is a finite set of objects and $E \subseteq V \times \Delta \times V$ is a set of directed edges labeled with symbols from Δ.

Before introducing preferentially annotated regular path queries, it will help to first review the classical regular path queries.

A *regular path query* (RPQ) is a regular language over Δ. For the ease of notation, we will blur the distinction between regular languages and regular expressions that represent them. Let Q be an RPQ and $DB = (V, E)$ a database. Then, the *answer* to Q on DB is defined as

$$Ans(Q, DB) = \{(a, b) \in V : \text{ for some } w \in Q,\ a \xrightarrow{w} b \text{ in } DB\},$$

where $\xrightarrow{w}$ denotes a path spelling the word w in the database.

We note that there exists also another variant of semantics for regular path queries, which asks for simple paths only (cf. [26, 10]). In this paper, we consider the first variant. We will return to the second variant in a future paper.

Semirings and annotated regular path queries. By a *semiring* we mean a tuple $\mathscr{R} = (R, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ such that

1. $(R, \oplus, \mathbf{0})$ is a commutative monoid with 0 as the identity element for $\oplus$.
2. $(R, \otimes, \mathbf{1})$ is a monoid with 1 as the identity element for $\otimes$.
3. $\otimes$ distributes over $\oplus$: for all $x, y, z \in R$,

$$(x \oplus y) \otimes z = (x \otimes z) \oplus (y \otimes z)$$
$$z \otimes (x \oplus y) = (z \otimes x) \oplus (z \otimes y).$$

4. $\mathbf{0}$ is an annihilator for $\otimes$: $\forall x \in R,\ x \otimes \mathbf{0} = \mathbf{0} \otimes x = \mathbf{0}$.

The *natural order* $\preceq$ on R is defined as: $x \preceq y$ if and only if $x \oplus y = x$. We interpret $x \preceq y$ to mean that x is better than y i.e. x brings less discomfort than y. Equivalently this means that x is *preferred* over y.

> For the rest of this paper, we require for semirings of preferences that their natural order $\preceq$ is a *total* order.

All the preference semirings mentioned in Introduction have this property.[2] We call such semirings *naturally linearly ordered* (NLO). Observe that $\mathbf{0}$ is the "biggest" element of the semiring, and it corresponds to the "infinitely worst" preference weight. Also, since $x \preceq x$ for each $x \in R$, in particular, addition is idempotent.

Now, let $\mathscr{R} = (R, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ be a semiring as above. An $\mathscr{R}$-annotated language Q over Δ is a function

$$Q : \Delta^* \to R.$$

Such functions are also called formal power series (cf. [5, 23, 30]).

[2] We want to note here that for database paths, it is difficult to find intuitively plausible preference semantics, which would ask for a non-total natural order $\preceq$.

We will call such Q's *annotated queries* for short. Frequently, we will write $(w, x) \in Q$ instead of $Q(w) = x$. When such annotated queries are given by "annotated regular expressions," we have *annotated regular path queries* (ARPQ's). Computationally, ARPQ's are represented by "annotated automata" defined as follows.

An *annotated automaton* $\mathcal{A}$ is a quintuple $(P, \Delta, \mathscr{R}, \tau, p_0, F)$, where τ is a subset of $P \times \Delta \times R \times P$, $p_0 \in P$, and $F \subseteq P$. Each annotated automaton $\mathcal{A}$ defines the annotated language (query) $[\![\mathcal{A}]\!]$ defined by

$$[\![\mathcal{A}]\!] = \{(w, x) \in \Delta^* \times R : w = r_1 r_2 \dots r_n, \\ x = \oplus \{\otimes_{i=1}^{n} x_i : (p_{i-1}, r_i, x_i, p_i) \in \tau, \text{ for } i = 1, \dots, n, \text{ and } p_n \in F\}\}.$$

The reader is referred to [4] for efficient algorithms translating annotated (weighted) regular expressions into annotated (weighted) finite automata. In this paper, we only use annotated finite automata.

Given a database DB, and a query Q, annotated over a semiring $\mathscr{R} = (R, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ we define the preferentially *weighted answer* of Q on DB as

$$Ans(Q, DB, \mathscr{R}) = \{(a, b, x) \in V \times V \times R : x = \oplus\{(y : (w, y) \in Q \text{ and } a \xrightarrow{w} b \text{ in } DB) \cup (\mathbf{0})\}.$$

Intuitively, for the semirings that we consider, we have $(a, b, \mathbf{0})$ as an answer to Q, if there is no path in DB spelling some word in Q.

Let us now discuss each of the preference semirings that we mentioned in Introduction. The *Boolean* semiring is

$$\mathscr{B} = (\{T, F\}, \vee, \wedge, F, T),$$

where T and F stand for "true" and "false" respectively, and $\vee$, $\wedge$ are the usual "or," and "and" Boolean operators. ARPQ's in the Boolean semiring correspond exactly to classical RPQ's. The user does not annotate explicitly the regular expression symbols by T or F. By default, all the symbols present in the query are assumed to be annotated with T. Also, the system produces only the "T-ranked" answers. In general, for any semiring it only makes sense to produce the answers, which are not ranked by the $\mathbf{0}$ of the semiring. In practice, a $\mathbf{0}$-ranked answer means in fact "no answer." For the $\mathscr{B}$ semiring, we formally have that

$$Ans(Q, DB, \mathscr{B}) = \{(a, b, T) : (a, b) \in Ans(Q, DB)\} \cup \{(a, b, F) : (a, b) \notin Ans(Q, DB)\}.$$

It is easy to see that a Boolean annotated automaton $\mathcal{A} = (P, \Delta, \mathscr{B}, \tau, p_0, F)$ is indeed an "ordinary" finite state automaton $(P, \Delta, \tau, p_0, F)$.

In the case of *quantitative preferences* we have

$$\mathscr{N} = (\mathbb{N} \cup \{\infty\}, min, +, \infty, 0),$$

where min and $+$ are the usual operators for integers. This semiring is also known as the *tropical semiring* in the literature (cf. [31]). The user annotates query symbols by natural numbers.

In the case of *qualitative preferences*, we have

$$\mathscr{F} = (\mathbb{N} \cup \{\infty\}, min, max, \infty, 0).$$

This semiring is also known as the *fuzzy semiring* in the literature. Similarly to the quantitative case, the user annotates query symbols by natural numbers. This is however, only syntactically "the same" as the quantitative case. The semantics of the two cases are different. The numbers here represent the "level of discomfort" for traversing database edges. As we mentioned in Introduction, it is the choice of the user to specify the semantics that she desires.

Finally, for hybrid preferences, the user again uses the same query syntax as for the quantitative and qualitative case. That is, the user annotates the query symbols with natural numbers. However,

here the set $\mathbb{N}$ is just the "user interface." In fact the support set of the semiring $\mathscr{H}$, for hybrid preference semantics is

$$R = \{0, 1, 1^{(2)}, \ldots, 2, 2^{(2)}, \ldots\} \cup \{\infty\},$$

where the symbolic ingredients, n and i, of a semiring element $n^{(i)}$ are natural numbers. [Elements $1, 2, \ldots$ are shorthand for $1^{(1)}, 2^{(1)}, \ldots$.] Intuitively, n represents the level of discomfort, while i represents how many times a user is "forced to endure" that level of discomfort. While the subset $\{0, 1, 2, \ldots\}$ is the user interface for annotating queries, set R is richer in elements in order to allow for a finer ranking of query answers.

Regarding the semiring operations, we introduce

$$n^{(i)} \oplus m^{(j)} = \begin{cases} n^{(i)} & \text{if } n < m \\ m^{(j)} & \text{if } n > m \\ n^{(min\{i,j\})} & \text{if } n = m, \end{cases} \qquad n^{(i)} \otimes m^{(j)} = \begin{cases} n^{(i)} & \text{if } n > m \\ m^{(j)} & \text{if } n < m \\ n^{(i+j)} & \text{if } n = m \end{cases}$$

and for these we have $\mathbf{0} = \infty$, $\mathbf{1} = 0$. It is easy to verify that the semiring axioms are satisfied.

Reiterating, the user, the same as before, annotates query symbols with natural numbers representing her preferences. However, semantically the queries will be different from both the quantitative and qualitative case, while bearing similarities to both of them. Similarly to the qualitative semantics, only database edges matched by transitions annotated with the "worst" level of discomfort will really count in computing a preferential weight for a traversed path. On the other hand, differently from the qualitative semantics, and similarly with the quantitative semantics, paths with the same "worst-level of discomfort" are comparable. Namely, the best path will be the one with the fewest "worst-level of discomfort" edges.

4 Answering Preferentially Annotated RPQ's

Our goal here is to not only compute preferentially weighted answers to a query, but to compute the answers in a progressive way, *i.e.* to compute the best answers first.

First, we will review the well-known method for the evaluation of classical RPQ's (cf. [1]). In essence, the evaluation proceeds by creating state-object pairs from the query automaton and the database. For this, let $\mathcal{A}$ be an ϵ-free NFA that accepts an RPQ Q. Starting from an object a of a database DB, we first create the pair (a, p_0), where p_0 is the initial state in $\mathcal{A}$. Then, we create all the pairs (b, p) such that there exist an edge from a to b in DB and a transition from p_0 to p in $\mathcal{A}$, and furthermore the labels of the edge and transition match. Also, we introduce an edge from (a, p_0) to (b, p). In the same way, we continue to create new pairs and edges, until we are not anymore able to do so. In essence, what is happening is a lazy construction of a Cartesian product graph of the query automaton with the database graph. Of course, only a small (hopefully) part of the Cartesian product is really constructed depending on the selectivity of the query.

After obtaining the above Cartesian product graph, producing query answers becomes a question of computing reachability of nodes (b, p), where p is a final state, from (a, p_0), where p_0 is the initial state. Namely, if (b, p) is reachable from (a, p_0), then (a, b) is a tuple in the query answer.

Now, when having instead an annotated query automaton, we can modify the classical matching algorithm to build an annotated (or weighted) Cartesian product graph. This can be achieved by assigning to the edges of this graph the corresponding (automaton) transition annotations (or weights). It is not difficult to see that, in order to compute preferentially weighted answers, we have to find, in the Cartesian product graph, the (semiring) shortest paths from (a, p_0) to all the nodes (b, p), where p is a final state in the query automaton $\mathcal{A}$.

In order to compute shortest paths in the above Cartesian product, we can run the "all-pairs shortest paths" algorithm of Floyd-Warshall (cf. [18]). For a detailed discussion and complexity analysis of this algorithm when answering weighted regular path queries see [10]. The Floyd-Warshall algorithm has the advantage that it works for any semiring of weights. On the other hand, it cannot

generate the query answers in an incremental and progressive fashion and might be expensive for large databases ([10]).

Notably, regarding our preference NLO semirings, we observe that they posses the following useful property.

Property 1. $\mathbf{1} \preceq x$ for each element $x \in R$.

In other words, we have that each element is "between" $\mathbf{1}$ and $\mathbf{0}$, with $\mathbf{1}$ being the "best" preferred element, and $\mathbf{0}$ being the "worst" preferred element. Based on this property, we have that

Lemma 1. *For all $x, y \in R$, $x \preceq x \otimes y$.*

Proof. From Property 1, we have that for each element y, $\mathbf{1} \preceq y$, i.e. $\mathbf{1} \oplus y = \mathbf{1}$.

If we multiply by x both sides of $\mathbf{1} \oplus y = \mathbf{1}$, we get $x \otimes (\mathbf{1} \oplus y) = x \otimes \mathbf{1}$, which by the distributive property of $\otimes$ over $\oplus$, becomes $(x \otimes \mathbf{1}) \oplus (x \otimes y) = x$, i.e. $x \oplus (x \otimes y) = x$, that is $x \preceq x \otimes y$. □

The above lemma says that x is better than $x \otimes y$. Since we aggregate the weights along a path using the $\otimes$ operator, we never get a lesser weight by expanding the path further with new edges. This allows us to employ a best-first strategy for computing query answers in the order of their preference rank.

In our first algorithm, we, in a similar spirit with [1], lazily build the above mentioned Cartesian product. However, we also compute "on the fly" shortest paths needed for preferentially weighting the answer tuples. Our algorithm, being a (graph) best-first search algorithm, maintains two structures, the "fringe" organized as a priority queue, and the "history-list" (or "closed-list") organized as a hashmap (cf. [34] for details on these artifacts). The fringe contains search-states to be expanded, while the history-list remembers the expansions so far. In our case, the elements of the fringe and history-list are of the form [(object, state), weight], e.g. $[(b, p), x)]$. The meaning of such an element $[(b, p), x)]$ is that node (b, p) of the Cartesian product is reachable from some "origin node" (a, p_0), and the weight of the best path known so far to reach (b, p) is x.

First, we describe how to progressively compute the $(a, \_, \_)$ answers starting from a database object a. Then, we generalize our algorithm to progressively compute all the query answers.

In order to compute the $(a, \_, \_)$ answers, the algorithm initializes first the fringe to have a single element $[(a, p_0), \mathbf{1}]$, where p_0 is the initial state of the query automaton, and $\mathbf{1}$ is the neutral element with respect to $\otimes$ operator. Also, the algorithm initializes the history-list to be empty. Then, the algorithm performs in a loop, until the fringe becomes empty, the following steps: (a) dequeue an element from the fringe, (b) check to see whether a similar element is in the history-list, (c) if not, then possibly generate a query answer, and "expand" the element obtaining new elements that are inserted in the fringe.

In order to better clarify the algorithm and the subsequent proofs, we formally define the Cartesian product $\mathcal{C}$ of a database DB and a query automaton $\mathcal{A}$ as the graph with

- nodes (b, p), where b is an object in DB and p is a state in $\mathcal{A}$, and
- edges $[(b, p), r, x, (c, q)]$, such that there exists an edge (b, r, c) (i.e. $b \xrightarrow{r} c$) in DB and a transition (p, r, x, q) in $\mathcal{A}$.

Based on this definition, we have that

Theorem 1. *$(a, b, x) \in Ans(Q, DB, \mathscr{R})$ if and only if there exists some path from (a, p_0) to (b, p) in $\mathcal{C}$, p is a final state in $\mathcal{A}$ and x is the weight of a cheapest[3] (with respect to $\preceq$) path.*

Proof. By the construction of $\mathcal{C}$, we have that:

1. For every path π_1 in DB matching some weighted transition path π_2 in $\mathcal{A}$, there exists some path π in $\mathcal{C}$ spelling the same word as π_1 (and π_2) and annotated by the same weights as π_2.

[3] There might be more than one such cheapest path.

2. For every path π in $\mathcal{C}$ there exist paths π_1 in DB and π_2 in $\mathcal{A}$, which match and spell the same word as π, and furthermore π_2 is annotated with the same weights as π.

Now, our claim is a direct consequence of these facts, and the definition of weighted query answers (see Section 3). □

We reiterate that $\mathcal{C}$ is only lazily built.
The detailed algorithm is as follows.

Algorithm 1

Input:

1. An ϵ-free[4] automaton $\mathcal{A}$ for an $\mathscr{R}$-annotated query Q
2. A database DB
3. An element a in DB

Output: All the $(a, \_, \_)$ answer tuples.

Method:

1. Initialize priority queue F (fringe) to $\{[(a, p_0), \mathbf{1})]\}$.
   Initialize hashmap H (history-list) to $\emptyset$.
2. Repeat (a), (b), and (c) until F becomes empty.
   (a) Dequeue an element, say $[(b, p), x]$, from priority queue F, such that x is the smallest weight of such elements, i.e.
   $$x = \oplus\{y : [(b, p), y] \in F\}.$$
   (b) If there is a $[(b, p), \_]$ element in H discard $[(b, p), x]$.
   (c) Otherwise,
      i. Insert $[(b, p), x]$ in H.
      ii. If p is a final state in $\mathcal{A}$, report (a, b, x) as a query answer.
      iii. Expand $[(b, p), x]$ as follows.
      For each transition (p, r, y, q) in $\mathcal{A}$ and edge $b \xrightarrow{r} c$ in DB add $[(c, q), x \otimes y]$ to F.

□

We remark that step 2.c.iii, in essence, incrementally grows the Cartesian product $\mathcal{C}$. For showing the correctness of Algorithm 1, we present a series a lemmas which culminate with Theorem 2.

Lemma 2. *Let $[(b, p), x]$ and $[(b', p'), x']$ be the dequeued elements in iteration i and $i + 1$, respectively. Then, $x \preceq x'$.*

Proof. Let F_i and F_{i+1} be the fringe contents in iteration i and $i + 1$, respectively. From steps 2.a and 2.c.iii, we have that
$$F_{i+1} = F^1_{i+1} \cup F^2_{i+1},$$
where
$$F^1_{i+1} = (F_i \setminus \{[(b, p), x]\}) \text{ and } F^2_{i+1} = \{[(c, q), x \otimes y] : \text{ for one or more } y\text{'s in } R\}.$$

[4] The procedure of [4] for building weighted automata from weighted regular expressions generates ϵ-free automata.

If $[(b', p'), x'] \in F^1_{i+1}$ then we have that $x \preceq x'$. This is because $F^1_{i+1} \subset F_i$ and x was the best in F_i, i.e. better than any element in F^1_{i+1}.

On the other hand, if $[(b', p'), x'] \in F^2_{i+1}$ then $x' = x \otimes y$ for some $y \in R$. Now by Lemma 1, $x \preceq x \otimes y = x'$ thus proving our claim. □

Lemma 3. *If there exists a path from (a, p_0) to (b, p) in $\mathcal{C}$, then there will be some (not necessarily unique) element $[(b, p), \_]$ that will be eventually inserted into F.*

Before giving the proof, we remark that this lemma does not imply that Cartesian product $\mathcal{C}$ is built in its entirety. There could be many "isolated islands" in the full $\mathcal{C}$, and clearly, such islands are never built.

Proof. Suppose that there exists a path π from (a, p_0) to (b, p) in $\mathcal{C}$, but the algorithm, during its execution, never inserts some $[(b, p), \_]$ into F. Let π be the edge sequence $((b_0, p_0), r_1, x_1, (b_1, p_1)), \ldots, ((b_{n-1}, p_{n-1}), r_n, x_n, (b_n, p_n))$, where $n \geq 1$, $b_0 = a$, $b_n = b$, and $p_n = p$. Clearly, $[(b_0, p_0), \mathbf{1}]$ will be enqueued and then subsequently dequeued and expanded. Let $k \in [1, n-1]$ be the number for which we have that for all $h \in [0, k-1]$ there is some $[(b_h, p_h), \_]$ inserted at some point into F, but there is never a $[(b_k, p_k), \_]$ inserted into F.

First observe that, since there is some $[(b_{k-1}, p_{k-1}), \_]$ inserted at some point into F, there is some (in fact only one) $[(b_{k-1}, p_{k-1}), x]$ expansion in step 2.c.iii.

Now, as (b_{k-1}, p_{k-1}) and (b_k, p_k) are consecutive nodes in π, there exists at least one edge connecting them; (at least) the edge in π. [Recall that an edge in $\mathcal{C}$ represents a match of a database edge and automaton transition.] Since, (b_{k-1}, p_{k-1}) and (b_k, p_k) are connected in $\mathcal{C}$, upon expansion of a $[(b_{k-1}, p_{k-1}), \_]$ element, some $[(b_k, p_k), \_]$ will be certainly produced and inserted in F, and this contradicts our supposition.

Thus, for all the nodes (b_i, p_i) in π, where $i \in [0, n]$, we have that some $[(b_i, p_i), \_]$ element will be certainly inserted (at some point) in F. This applies to $(b_n, p_n) = (b, p)$ as well, and so, some $[(b, p), \_]$ element will be eventually inserted in F. □

From the above lemma and steps 2.b and 2.c.i, we have that

Corollary 1. *If there exists a path from (a, p_0) to (b, p) in $\mathcal{C}$, then there will be an element $[(b, p), \_]$ that will be eventually inserted into H.*

Clearly, there is only one such insertion in H. Now we show that

Lemma 4. *Let $[(b, p), x]$ be an element in H. Then, x is the weight of a cheapest path going from (a, p_0) to (b, p) in $\mathcal{C}$.*

Proof. From Lemma 2, the first $[(b, p), \_]$ to be dequeued from F will have the smallest weight not only among those $[(b, p), \_]$ elements that could currently be in F, but in general among all $[(b, p), \_]$ elements that would ever be enqueued in and then dequeued from F during the whole execution of the algorithm. Let $[(b, p), y]$ be this (first) $[(b, p), \_]$ tuple. From steps 2.b and 2.c.i, $[(b, p), y]$ is inserted in H and never overwritten.

Now, let π, with a weight z, be a cheapest path from (a, p_0) to (b, p) in $\mathcal{C}$. Then, we claim that $[(b, p), z]$ will be eventually inserted into H. From this, our claim will follow as there can be only one $[(b, p), \_]$ element in H, i.e. x will have to be equal to z.

Let π be the edge sequence $((b_0, p_0), r_1, z_1, (b_1, p_1)), \ldots, ((b_{n-1}, p_{n-1}), r_n, z_n, (b_n, p_n))$, where $n \geq 1$, $b_0 = a$, $b_n = b$, $p_n = p$ and $\otimes_{i=1}^{n} z_i = z$.

Clearly, $[(b_0, p_0), \mathbf{1}]$ will be enqueued and then subsequently dequeued and inserted into H. Suppose now that $[(b_{h-1}, p_{h-1}), \otimes_{i=1}^{h-1} z_i]$, for some $h \in [1, n]$, is in H. By steps 2.b, 2.c.i and 2.c.iii, we have that $[(b_{h-1}, p_{h-1}), \otimes_{i=1}^{h-1} z_i]$ has been in F, dequeued from F and then expanded. The expansion of $[(b_{h-1}, p_{h-1}), \otimes_{i=1}^{h-1} z_i]$ brings in F the element $[(b_h, p_h), \otimes_{i=1}^{h} z_i]$ (see step 2.c.iii). Then, since the algorithm continues until F gets completely empty, there will be a moment when $[(b_h, p_h), \otimes_{i=1}^{h} z_i]$

is dequeued and tested to see whether or not there already exists some other element $[(b_h, p_h), y]$ in H. If there is no such element in H, then we insert $[(b_h, p_h), \otimes_{i=1}^{h} z_i]$ in H.

Suppose that there exists such $[(b_h, p_h), y]$ in H. By the discussion in the begining of this proof, we know that $y \preceq \otimes_{i=1}^{h} z_i$. This means that there exists a path, say π'_h, from (a, p_0) to (b_h, p_h) which is cheaper than subpath π_h of π going from $(b_0, p_0) = (a, p_0)$ to (b_h, p_h). This means in turn that we could replace π_h with π'_h in π and obtain a cheaper path than π going from (a, p_0) to $(b_n, p_n) = (b, p)$, which is a contradiction. Thus, $[(b_h, p_h), \otimes_{i=1}^{h} z_i]$ will be inserted into H.

Inductively, we have that $[(b_n, p_n), \otimes_{i=1}^{n} z_i] = [(b, p), z]$ will be inserted into H and this completes our proof. □

Based on all the above, we have that

Theorem 2. *Algorithm 1 is sound and complete.*

Proof. "*soundness.*" From step 2.c.ii we have that the output produced by the algorithm is in fact

$$\{(a, b, x) : [(b, p), x] \in H \text{ and } p \text{ is final state in } \mathcal{A}\}.$$

Now, let $[(b, p), x]$ be an element in H and (a, x, b) be the corresponding produced answer to the given query. From Lemma 4, we have that x is the weight of a cheapest path in $\mathcal{C}$ connecting (a, p_0) with (b, p). From Theorem 1, (a, x, b) is an answer to the given query.

"*completeness.*" Let (a, b, x) be an answer to the given query. From Theorem 1, there exists some path from (a, p_0) to (b, p) in $\mathcal{C}$, and x is the weight of a cheapest path. From Corollary 1, the existence of some path from (a, p_0) to (b, p) in $\mathcal{C}$ means that an element $[(b, p), \_]$ will be eventually inserted into H. From Lemma 4, the exact weight of this element will equal the weight of a cheapest path from (a, p_0) to (b, p) in $\mathcal{C}$, i.e. x. Thus, (a, b, x) will be produced as an answer by the algorithm. □

Observe that a tuple (a, b, x) can be produced as output as soon as it is discovered and inserted in H. The answers are generated in order of their preference weight. However, this order is with respect to the subset $(a, \_, \_)$ of query answers. Thus, the question is: How can we generate the answers in the *absolute* order of their preference weight?

For this, we modify Algorithm 1 to run from each potential Cartesian product node (a, p_0). Such potential nodes can be easily identified by using a classical index on the database edge labels. Now, in order to generate the answers in the absolute order of their preference weight, we use a *common for all* fringe, but different history-lists. The elements of the fringe have also an additional component for recording the relevant [to the element] source. Formally, the modified algorithm is as follows.

Algorithm 2

Input:

1. An ϵ-free automaton $\mathcal{A}$ for an $\mathscr{R}$-annotated query Q
2. A database DB
3. A set S of start elements in DB

Output: $Ans(Q, DB, \mathscr{R})$

Method:

1. Initialize priority queue F (fringe) to $\{[a, (a, p_0), \mathbf{1}] : a \in S\}$.
   For each element $a \in S$, initialize a hashmap H_a (history-list) to $\emptyset$.
2. Repeat (a), (b), and (c) until F becomes empty.

(a) Dequeue an element, say $[a, (b, p), x]$, from priority queue F, such that x is the smallest weight of such elements, i.e.

$$x = \oplus\{y : [a, (b, p), y] \in F\}.$$

(b) If there is a $[(b, p), \_]$ element in H_a discard $[a, (b, p), x]$.

(c) Otherwise,
   i. Insert $[(b, p), x]$ in H_a.
   ii. If p is a final state in $\mathcal{A}$, report (a, b, x) as a query answer.
   iii. Expand $[a, (b, p), x]$ as follows.
   For each transition (p, r, y, q) in $\mathcal{A}$ and edge $b \xrightarrow{r} c$ in DB add $[a, (q, c), x \otimes y]$ to F.

□

The soundness and completeness of Algorithm 2 follows from the soundness and completeness of Algorithm 1. This is because Algorithm 2, in essence, synchronizes the parallel runs of a set of instances of Algorithm 1.

Regarding the complexity, observe that in both algorithms 1 and 2, the most expensive step is the expansion of a triple. This is because for an expansion, we need to access a database object. Let $|V_c|$ be the number of nodes in the relevant (to the query) Cartesian product. Then, there are $\mathcal{O}(|V_c|)$ and $\mathcal{O}(|V_c|^2)$ expansions for algorithms 1 and 2 respectively. Note that this data complexity is the same as for classical RPQ's (cf. [1]), and we are not adding extra work because of the preference semantics.

5 Preferentially Ranked Answers on Possible Databases

In a semistructured LAV data integration system (cf. [6, 7, 24, 8]), we do not have a database in the classical sense. Instead what we have is incomplete information, which is in the form of a set of "data-sources," characterized by an algebraic definition over a "global schema."

Each data-source also has a name, and the set of these names constitutes the "local schema." The LAV system also has a set of tuples over the local schema. The queries are formulated on the "integrated" global schema. Since the data exists in the local schema only, a translation from the global to the local schema has to be performed in order to be able to compute query answers.

When the user gives an ARPQ, the question is: What does it mean to preferentially answer such a query in a LAV system?

Formally, let Δ be the *global schema*. Let $\mathbf{S} = \{S_1, \ldots, S_n\}$ be a set of *data-source definitions*, with each S_i being a regular language over the global schema Δ. Associated with each data-source is a name s_i, for $i = 1, \ldots, n$. The *local schema* is the set $\Omega = \{s_1, \ldots, s_n\}$ of all the data-source names. There is a natural mapping between the local and global schema: for each $s_i \in \Omega$, we set $def(s_i) = S_i$. The mapping or substitution[5] *def* associates with each data-source name s_i the definition language S_i. The substitution *def* is applied to words, languages, and regular expressions in the usual way (see e. g. [18]).

Let $\Omega = \{s_1, \ldots s_n\}$ be the local schema as before. Then, a *source collection* $\mathcal{S}$ over $(\mathbf{S}, \Omega)$ is a database over (D, Ω), i.e. a database with objects from D and edges labeled by symbols on Ω.

As mentioned earlier, in a LAV system, the user formulates queries on the global schema, *i.e.* Δ, and the system has to compute the answer on the data available in the local schema, *i.e.* Ω. For this, we have to reason about hypothetical databases over (D, Δ) that a database over (D, Ω) could possibly represent.

A source collection $\mathcal{S}$ defines a set $poss(\mathcal{S})$ of databases over (D, Δ) as follows:

$$poss(\mathcal{S}) = \{DB \ : \ \text{there exists a path } a \xrightarrow{w \in S_i} b \text{ in } DB \text{ for each } (a, s_i, b) \in \mathcal{S}\}.$$

[5] In a language theoretic terminology.

This definition reflects the intuition about the connection of an edge (a, s_i, b) in $\mathcal{S}$ with paths between a and b in hypothetical DB's.

For classical regular path queries, what we usually compute is the *certain answer* using $\mathcal{S}$, which is the set of all tuples, which are in the query answer on each possible database.

Consider a classical regular path query as a preferentially annotated query over the Boolean semiring $\mathscr{B}$. In a semiring terminology, what we do is an "$\wedge$" aggregation of query answers on the possible databases. Also, let us overload $\wedge$ operator to work for answer tuples and sets as follows: $(a, b, x) \wedge (a, b, y) = (a, b, x \wedge y)$, and

$$Ans(Q, DB_1, \mathscr{B}) \wedge Ans(Q, DB_2, \mathscr{B}) = \\ \{(a, b, x \wedge y) : (a, b, x) \in Ans(Q, DB_1, \mathscr{B}) \text{ and } (a, b, y) \in Ans(Q, DB_2, \mathscr{B})\}.$$

Then, the certain answer w.r.t. $\mathcal{S}$ and "weighted" over $\mathscr{B}$ is

$$CAns(Q, \mathcal{S}, \mathscr{B}) = \bigwedge_{DB \in poss(\mathcal{S})} Ans(Q, DB, \mathscr{B}),$$

It is easy to verify that this definition is equivalent with the definition of the certain answer given in other works as for example [6, 7].

In fact, $\wedge$ for aggregating the answers on possible databases is the "dual operator" of $\vee$ used for aggregating paths when computing answers on databases.[6] Generalizing, in order to define the certain answer for other semirings, we introduce the $\odot$ operator, which is the dual of the path aggregation operator $\oplus$. Namely,

$$x \odot y = \begin{cases} x & \text{if } x \oplus y = y \\ y & \text{if } x \oplus y = x. \end{cases}$$

This is possible since $\oplus$ induces a total order, and so, $x \oplus y$ is equal to either x or y. Clearly, $\wedge$ is the dual of $\vee$ according to this definition. Observe also that the operator $\odot$ induces the reverse order (with respect to $\oplus$) among the elements of the semiring. Hence $\oplus$ is the "minimum" and $\odot$ the "maximum" operation.

Similarly with the above overloading of $\wedge$, we overload $\odot$ to work with answer tuples and sets. Now, for a query Q, annotated over a preference semiring $\mathscr{R}$, we define the certain answer as

$$CAns(Q, \mathcal{S}, \mathscr{R}) = \bigodot_{DB \in poss(\mathcal{S})} Ans(Q, DB, \mathscr{R}).$$

In the above definition, a tuple (a, b, x), with $x \neq \mathbf{0}$, will belong to $CAns(Q, \mathcal{S}, \mathscr{R})$ iff for each $DB \in poss(\mathcal{S})$ there exists $y \preceq x$ such that $(a, b, y) \in Ans(Q, DB, \mathscr{R})$ and x is the "smallest" element with this property. This definition reflects the *certainty* that objects a and b are always connected with paths, which are preferentially weighted not more than x. As an example, consider the query

$$Q = (highway : 0)^* \,\|\, (road : 1 + \epsilon)^*,$$

and a source collection (consisting of single source with a single tuple) $\mathcal{S} = \{(a, s, b)\}$, with definition

$$S = highway^* \,\|\, (road + \epsilon)^5.$$

The possible databases for $\mathcal{S}$ are all those databases, which have at least a path (between a and b) labeled by highways intervened by at most 5 roads. Now let us discuss the certain answer considering the semirings for the quantitative, qualitative, and hybrid preference semantics.

[6] The fact that this operator $\wedge$ is the same as the "multiplication" operator of the Boolean semiring for aggregating edge-weights along paths, is just a coincidence.

In the quantitative case, $\odot$ is max, and we have $(a, b, 5)$ as a certain answer. The weight of 5 states exactly our certainty that in any possible database, there is a path from a to b, whose preferential weight w.r.t. the given query is not more than 5. Also, there exists a possible database in which the best path between a and b is exactly 5.

In the qualitative case, $\odot$ is again max. However, we have now $(a, b, 1)$ as a certain answer. The weight of 1 states our certainty that in any possible database, there is a path from a to b, and the level of discomfort (w.r.t. the query) for traversing that path is not more than 1.

Finally, in the hybrid case, $\odot$ is as follows

$$n^{(i)} \odot m^{(j)} = \begin{cases} m^{(j)} & \text{if } n < m \\ n^{(i)} & \text{if } n > m \\ n^{(max\{i,j\})} & \text{if } n = m. \end{cases}$$

We have that $(a, b, 1^{(5)})$ as a certain answer. This is because although the level of discomfort of the best path connecting a with b in any possible database is 1, in the worst case (of such best paths), we need to endure up to 5 times such discomfort (w.r.t. the query). Of course $1^{(5)}$ is infinitely better than 2.

6 Certain Answers via Query Spheres

In [6], there is given an algorithm, which computes the certain answer of a classical RPQ Q given a source collection $\mathcal{S}$. This translates into having available an algorithm for computing $CAns(Q, \mathcal{S}, \mathscr{B})$.

Now, let Q be an ARPQ with annotations over a preference semiring $\mathscr{R}$. In this section, we cast computing tuples in $CAns(Q, \mathcal{S}, \mathscr{R})$ into computing tuples in $CAns(Q, \mathcal{S}, \mathscr{B})$, which is the Boolean certain answer of Q, after "collapsing" all the annotations in Q into element T of the Boolean semiring.

For this, we introduce the notion of "query spheres." We formally define the *y-sphere* of Q, where $y \in R$, as

$$Q^y = \{(w, x) \in \Delta^* \times R : (w, x) \in Q \text{ and } x \preceq y\}.$$

Let $\mathcal{A}$ be an annotated automaton recognizing Q. Then, Q^y will be the query recognized by the automaton $\mathcal{A}^y$ obtained from $\mathcal{A}$ by retaining only (transition) paths weighted by some x, which is no more than y. We show in the next section how to obtain such automata for the different preference semirings that we consider.

Clearly, $Q^x \subseteq Q^y$ for $x \preceq y$.[7]

For semirings in which the notion of the "next" element is well defined, we give a necessary and sufficient condition for a tuple (a, b, y) to belong to $CAns(Q, \mathcal{S}, \mathscr{R})$. We give the following definition about the "next element" property of a semiring.

A preference semiring $\mathscr{R} = (R, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is said to be *discrete* iff for each $x \neq \mathbf{0}$ in R there exists y in R, such that (a) $x \prec y$, and (b) there does not exist z in R, such that $x \prec z \prec y$. The element y is called the *next element after x*.

Notably, all our preference semirings are discrete. We show that

Theorem 3. *Let Q be a query annotated over a discrete NLO semiring $\mathscr{R} = (R, \oplus, \otimes, \mathbf{0}, \mathbf{1})$. Also, let y be the next element after some element x in $\mathscr{R}$. Then,*

$$(a, b, y) \in CAns(Q, \mathcal{S}, \mathscr{R}) \text{ iff } (a, b, T) \in CAns(Q^y, \mathcal{S}, \mathscr{B}) \text{ and } (a, b, F) \in CAns(Q^x, \mathcal{S}, \mathscr{B}),$$
$$(a, b, \mathbf{1}) \in CAns(Q, \mathcal{S}, \mathscr{R}) \text{ iff } (a, b, T) \in CAns(Q^{\mathbf{1}}, \mathcal{S}, \mathscr{B}).$$

[7] It is this property that motivates the use of "query spheres."

Proof. We only show here the first claim. The second claim can be shown similarly.

"If." Let (a, b, T) be a tuple in $CAns(Q^y, \mathcal{S}, \mathscr{B})$. Then, (a, b, T) will be in the answer of Q^y on each possible database with respect to the source collection $\mathcal{S}$. Let, DB be such a possible database. For DB, we have $(a, b, T) \in Ans(Q^y, DB, \mathscr{B})$. This means that there exists a path from a to b in DB, which spells a word w in Q^y (considering it as a classical RPQ; see beginning of this section). However, Q^y is an ARPQ and by its definition, what we have in fact is that (for this word w) $(w, y') \in Q^y$ for $y' \preceq y$. Thus, $(a, b, y') \in Ans(Q, DB, \mathscr{R})$. In other words, if $(a, b, T) \in CAns(Q^y, \mathcal{S}, \mathscr{B})$, then a corresponding $\mathscr{R}$-weighted tuple will be in the certain answer of the original query Q on each possible database, and furthermore its weight on any such database will not be greater than y.

Now, if $(a, b, T) \in CAns(Q^y, \mathcal{S}, \mathscr{B})$ but $(a, b, F) \in CAns(Q^x, \mathcal{S}, \mathscr{B})$, then we claim that there exists a possible database DB, for which we have $(a, b, y) \in Ans(Q, DB, \mathscr{R})$, *i.e.* the weight of such a tuple is exactly y. If not, then we would have that for each possible DB, $(a, b, y') \in Ans(Q, DB, \mathscr{R})$, where y' is strictly $\prec y$. This means that in each possible DB there exist some path spelling a word, say w, such that $(w, y') \in Q$ and $y' \prec y$ (strict $\prec$). From the next element property, the above implies that $y' \preceq x$. So, all these (w, y') belong to Q^x. This implies that $(a, b, T) \in Ans(Q^x, DB, \mathscr{B})$, for each possible DB, *i.e.* $(a, b, T) \in CAns(Q^x, \mathcal{S}, \mathscr{B})$, which is a contradiction.

Hence, if $(a, b, T) \in CAns(Q^y, \mathcal{S}, \mathscr{B})$ but $(a, b, F) \in CAns(Q^x, \mathcal{S}, \mathscr{B})$, then $(a, b, y) \in CAns(Q, \mathcal{S}, \mathscr{R})$.

"Only if." By the definition of $\odot$ operator, we can observe that if $(a, b, y) \in CAns(Q, \mathcal{S}, \mathscr{R})$, then $(a, b, y') \in Ans(Q, DB, \mathscr{R})$, where $y' \preceq y$ for each possible DB. This means that there exists $(w, y') \in Q$, where $y' \leq y$, and there exists a path from a to b in DB, spelling w. Clearly, $(w, y') \in Q^y$, and so we have that $(a, b, T) \in Ans(Q^y, DB, \mathscr{B})$. Since DB was an arbitrary possible database, we get that $(a, b, T) \in CAns(Q^y, \mathcal{S}, \mathscr{B})$.

Additionally, $(a, b, y) \in CAns(Q, \mathcal{S}, \mathscr{R})$ also means that there exists a possible database DB, for which $(a, b, y) \in Ans(Q, DB, \mathscr{R})$. That is, the minimum weight of Q words, which are spelled by paths in DB is exactly y, *i.e.* no word in Q with weight $y' \prec y$ (strict) can match some path from a to b in DB. So, we have that $(a, b, F) \in CAns(Q^x, \mathcal{S}, \mathscr{B})$. □

From the above theorem, we conclude that if we are able to compute Q^y for each y (relevant to the query), then we could generate all the y-ranked tuples (a, b, y) of $CAns(Q, \mathcal{S}, \mathscr{R})$ by computing with the algorithm of [6] $CAns(Q^y, \mathcal{S}, \mathscr{B})$ and $CAns(Q^x, \mathcal{S}, \mathscr{B})$, and then taking the set difference of their T-tuples.

We present in the next section algorithms, which for a given y compute Q^y, for the different preference semirings that we study.

Now the question is, for what y's to apply the method suggested by Theorem 3 for generating (a, b, y) tuples of the certain answer? For this, let $z = \odot\{x : (w, x) \in Q\}$.

Equivalently, $x \preceq z$, for all x such that $(w, x) \in Q$, and there is no $z' \prec z$ satisfying the same property. We have the following theorem.

Theorem 4. $Q^z = Q$.

Proof. Since Q^z is a query sphere, by definition, $Q^z \subseteq Q$.

Now let $(w, x) \in Q$. By the definition of z, we have that $x \preceq z$. This in turn, by the definition of a query sphere of index z, means that $(w, x) \in Q^z$. □

For the quantitative and qualitative semirings, the existence of a $z \prec \mathbf{0}$ (strict $\prec$) guarantees a terminating procedure for ranking all the tuples in the certain answer. Simply, one has to repeat the method of Theorem 3 starting with y equal to $\mathbf{1}$ and continuing for up to y equal to z. On the other hand, for the hybrid semiring a "global" (upper bound) z is not enough. Rather, we need to reason about "level-wise" z's, as we explain later in this section.

Quantitative case. Interestingly, determining whether there exists such a $z \prec \mathbf{0}$ coincides with deciding the "limitedness" problem for "distance automata." The latter problem is widely known and positively solved in the literature (cf. for example [14, 25, 31, 16]).

A recent paper that elegantly solves the problem of limitedness for distance desert automata (which are a generalization of distance automata) is [20]. We remark that the most efficient algorithms for deciding the limitedness problem are those by [25] and [20], running in exponential time in the size of the automaton. [The problem is PSPACE-complete.]

If the query automaton is limited in distance, and this limit is z, then we need to compute query spheres up to Q^z, which will be equivalent to Q. The question is: How many such query spheres might be needed to compute in the worst case? Hashiguchi in [16] showed that such a z limit can be up to single exponential in the automaton size. Hence, we need to compute an exponential (in Q) number of Q^y query spheres. However, this complexity is absorbed by the complexity of computing $CAns(Q^y, \mathcal{S}, \mathscr{B})$ and $CAns(Q^x, \mathcal{S}, \mathscr{B})$ with the method of [6], which is exponential in Q.

On the other hand, if the query automaton is not limited in distance, we can still apply the same procedure utilizing query spheres for ranking the tuples in the certain answer. However, the ranking in this case is only *eventually computable*. This means that in such a case, we can continue trying bigger and bigger query spheres with the hope of being able to include each of the $(a, b, \_)$ tuples in some query sphere answer and be able to rank them. However, there is no guarantee that such a Q^y sphere would exist for $y \neq \mathbf{0}$. Note that by computing $CAns(Q, \mathcal{S}, \mathscr{B})$ with the method of [6], we do know all the $(a, b, \_)$ tuples in $CAns(Q, \mathcal{S}, \mathscr{R})$, but without the third, weight component needed for ranking.

In practice, the user might provide beside the query, also an upper bound z' on the preferential weight of the answers that she is interested to retrieve. In such a case, we need to compute not more than z' query spheres in order to return all the tuples weighted less or equal to z' in $CAns(Q, \mathcal{S}, \mathscr{R})$.

Qualitative case. Here, the existence of $z \prec \infty$ (semiring $\mathbf{0}$) is guaranteed. This is because z will be less or equal to the biggest transition weight in the query automaton.

Hybrid case. In this case, the existence of a global $z \prec \infty$ does not guarantee the ability to rank all the tuples in the certain answer. Rather we need for this the existence of the level-wise z's. Namely, we define the *upper bound for level* n as

$$z_n = \odot\{x : (w, x) \in Q \text{ and } x \prec n + 1\}$$

(strict $\prec$, and recall $n + 1$ is a shorthand for $(n + 1)^{(1)}$). If there exists $k \in \mathbb{N}$, such that $z_n = n^{(k)}$, we say that z_n is *finite*.

Now, if z_n is finite, then for determining the exact weight of the "n-range" tuples $(a, b, n^{(\_)})$ in the certain answer, we need to compute query spheres from $Q^{n^{(1)}}$ up to $Q^{n^{(k)}}$.

If $z_n = z_m$ for $m \prec n$ (strictly), then there cannot be any n-range tuple in the certain answer.

On the other hand, if $z_n > n^{(i)}$ for each $i \in \mathbb{N}$, then the exact weight of the "n-range" tuples is only eventually computable.

Hence the question is how can we determine the existence of a finite z_n?

For this, we first introduce the generalized $n^{(\infty)}$*-query sphere*

$$Q^{n^{(\infty)}} = \bigcup_{i=0}^{\infty} Q^{n^{(i)}}.$$

Equivalently,

$$Q^{n^{(\infty)}} = \left\{ \left(w, m^{(i)} \right) : m, i \in \mathbb{N} \text{ and } m \leq n \right\}.$$

We will soon derive a necessary and sufficent condition for the finiteness of z_n based on automata for $Q^{n^{(i)}}$. So, how can we build an automaton for $Q^{n^{(\infty)}}$? In essence, we want to extract the paths in a query automaton $\mathcal{A} = (P, \Delta, \mathbb{N}, p_0, \tau_{\mathcal{A}}, F)$, which are weighted strictly less than $n + 1$. Such paths cannot recognize words weighted more or equal to $n + 1$. Performing this extraction is easy; we only retain the transitions of $\mathcal{A}$ which are weighted strictly less than $n + 1$ and remove all the other transitions. Let $\mathcal{B}_n$ be this new weighted automaton. We show that

Theorem 5. *$\mathcal{B}_n$ accepts exactly $Q^{n^{(\infty)}}$.*

Proof. Since $\mathcal{B}_n$ has only transitions weighted strictly less than $n+1$, the words that $\mathcal{B}_n$ accepts are also weighted strictly less than $n+1$. Thus, these words belong to $Q^{n^{(\infty)}}$.

Conversely, let $(w,x) \in Q^{n^{(\infty)}}$. By the definition of $Q^{n^{(\infty)}}$, $(w,x) \in Q$ and $x \prec n+1$. Thus, there exists a path π in $\mathcal{A}$ (automaton recognizing Q) whose weight is x. As such, π will have only transitions weighted stricly less than $n+1$. By the construction of $\mathcal{B}_n$, π "survives," and thus, π is an accepting path in $\mathcal{B}_n$, i.e. (w,x) is accepted by $\mathcal{B}_n$. □

From automaton $\mathcal{B}_n$, we can build automaton $\mathcal{B}'_n$ over the tropical (quantitative) semiring. Recall that the weights of the transitions in $\mathcal{B}_n$ are from $\{0,1,\ldots,n\}$. Now, $\mathcal{B}'_n$ will be identical to $\mathcal{B}_n$, but with the weights of transitions changed to be 0 and 1 when the weights of the corresponding transitions in $\mathcal{B}_n$ are less than n and exactly n, respectively. We have that

Lemma 5. *$\left(w, n^{(i)}\right)$ is accepted by $\mathcal{B}_n$* if and only if *(w,i) is accepted by $\mathcal{B}'_n$.*

Proof. If. The fact that (w,i) is accepted by $\mathcal{B}'_n$ says that a cheapest accepting path, say π, spelling w in $\mathcal{B}'_n$ has i transitions marked by 1's. By the construction of $\mathcal{B}'_n$, we have that π is also a cheapest accepting path spelling w in $\mathcal{B}_n$. Path π in $\mathcal{B}_n$ will have exactly i occurences of n, thus $\mathcal{B}_n$ accepts $\left(w, n^{(i)}\right)$.

Only. The fact that $\left(w, n^{(i)}\right)$ is accepted by $\mathcal{B}_n$ says that a cheapest accepting path, say π, spelling w in $\mathcal{B}_n$ has i transitions marked by n's. By the construction of $\mathcal{B}'_n$, we have that π is also a cheapest accepting path spelling w in $\mathcal{B}'_n$. Path π in $\mathcal{B}'_n$ will have exactly i occurences of 1, thus $\mathcal{B}_n$ accepts (w,i). □

Now, we show that

Theorem 6. *z_n is finite* if and only if *$\mathcal{B}'_n$ is limited in distance, that is, there exists a $k \in \mathbb{N}$, such that every word accepted by $\mathcal{B}_n$ is weighted less or equal to k.*

Proof. If. Since $\mathcal{B}'_n$ is limited in distance, there exists $k \in \mathbb{N}$, such that for all (w,h) accepted by $\mathcal{B}'_n$, we have that $h \leq k$. From this and Lemma 5, we have that $n^{(h)} \preceq n^{(k)}$ for all $\left(w, n^{(h)}\right)$ accepted by $\mathcal{B}_n$. From the definition of z_n, we have that $z_n = n^{(k)}$, thus z_n is finite.

Only. If z_n is finite, then there exists $k \in \mathbb{N}$, such that $z_n = n^{(k)}$. By the definition of z_n, for each $\left(w, n^{(h)}\right)$ in $Q^{n^{(\infty)}}$ (i.e. accepted by $\mathcal{B}_n$), $n^{(h)} \preceq n^{(k)}$. From this and Lemma 5, we have that $h \leq k$ for all (w,h) accepted by $\mathcal{B}'_n$, thus k is a bound for $\mathcal{B}'_n$. □

Here again, the user can practically specify an upper bound k on the preferential weight of the tuples in each range that she is interested to exactly rank. Such a bound will serve as an accuracy index. By computing query spheres up to $Q^{n^{(k)}}$, we accurately rank the n-range tuples having a weight, which is not more than $n^{(k)}$.

Finally, we can "inaccurately" derive the rest of n-range tuples, by computing the whole $CAns(Q^{n^{(\infty)}}, \mathcal{S}, \mathscr{B})$. By "inaccurately" we mean that for the n-range tuples weighted more than $n^{(k)}$, we only know that their weight is from $n^{(k)}$ to $n+1$ exclusive.

7 Computing Query Spheres

7.1 Quantitative Case

In this section we present an algorithm, which for any given number $k \in \mathbb{N}$ constructs the k-th sphere Q^k of an ARPQ Q.

For this, we build a mask automaton $\mathcal{M}_k$ on the alphabet $K = \{0,1,\ldots,k\}$, which formally is as follows: $\mathcal{M}_k = (P_k, K, \tau_k, p_0, F_k)$, where $P_k = F_k = \{p_0, p_1, \ldots, p_k\}$, and

$$\tau_k = \{(p_i, h, p_{i+h}) : 0 \leq i \leq k, \text{ and } 0 \leq h \leq k-i\}.$$

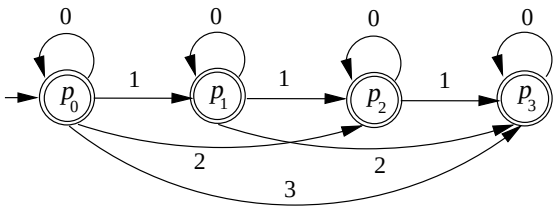

Fig. 1. Automaton $\mathcal{M}_3$

As an example, we give $\mathcal{M}_3$ in Fig. 1. Automaton $\mathcal{M}_k$ has a nice property. It captures all the possible number sequences with cumulative weight less or equal to k. We show that

Theorem 7. *$\mathcal{M}_k$ accepts* all *and* only *the number sequences σ with weight$(\sigma) \leq k$.*

Proof. We will show this by induction. Clearly, our claim is true for $\mathcal{M}_0$ and $\mathcal{M}_1$, and suppose that it is true for all $\mathcal{M}_h$ automata up to $\mathcal{M}_{k-1}$, for some $k \geq 2$. We want to show now that the claim is true for $\mathcal{M}_k$.

By construction, $\mathcal{M}_k$ will contain all of $\mathcal{M}_h$, for $0 \leq h \leq k-1$, as subautomata, and since all the states are final, $L(\mathcal{M}_h) \subset L(\mathcal{M}_k)$ for all $0 \leq h \leq k-1$. Thus, $\mathcal{M}_k$ will accept all the number sequences with a weight of less or equal to $k-1$.

Due to the construction of τ_k, for a given $h \in [1, k-1]$, all subautomata on states $\{p_i, \ldots, p_{i+h}\}$, for $i \in [0, k-h]$, are identical to $\mathcal{M}_h$. More specifically, what we mean here is that, the subautomaton using states $\{p_i, \ldots, p_{i+h}\}$ is identical to the subautomaton using states $\{p_j, \ldots, p_{j+h}\}$, for $i, j \in [0, k-h]$, and all such subautomata are identical to $\mathcal{M}_h$.

From this, and the general construction of $\mathcal{M}_k$, we have that

$$L(\mathcal{M}_k) = \left\{ \bigcup_{h=1}^{k-1} L(\mathcal{M}_h) \cdot L(\mathcal{M}_{k-h}) \right\} \cup \{k\}.$$

By the inductive hypothesis, for a given $h \in [1, k-1]$, $\mathcal{M}_h$ and $\mathcal{M}_{k-h}$ do accept all the possible number sequences of weight h and $h-k$, respectively. Thus, the union $\bigcup_{h=1}^{k-1} L(\mathcal{M}_h) \cdot L(\mathcal{M}_{k-h})$ gives us all the sequences that can be obtained by concatenating two sequences of weight h and $k-h$, respectively, for all $h \in [1, k-1]$.

These sequences, together with the single number sequence k, comprise *all* the possible sequences with a cumulative weight of k.

For the *only* part of our claim, again from the above equality and the inductive hypothesis, we can see that no sequence having a weight of more than k can be accepted by $\mathcal{M}_k$. □

Now using $\mathcal{M}_k$, we can extract from a weighted automaton $\mathcal{A}$ for Q all the accepting transition paths with a weight less or equal to k, giving so an effective procedure for computing the k-th sphere $Q^{(k)}$.

For this, let $\mathcal{A} = (P_{\mathcal{A}}, \Delta, \tau_{\mathcal{A}}, q_0, F_{\mathcal{A}})$ be a weighted automaton for Q. We construct a Cartesian product automaton

$$\mathcal{C}_k = \mathcal{A} \times \mathcal{M}_k = (P_{\mathcal{A}} \times P_k, \Delta, \tau, (q_0, p_0), F_{\mathcal{A}} \times F_k),$$

where $\tau = \{((q,p), r, n, (q',p')) : (q, r, n, q') \in \tau_{\mathcal{A}} \text{ and } (p, n, p') \in \tau_k\}$. We have that

Theorem 8. *The weighted automaton $\mathcal{C}_k$ accepts exactly the k-th sphere $Q^{(k)}$ of query Q.*

Proof. Straightforward from the properties of the Cartesian product. □

We can observe that the size of automaton $\mathcal{M}_k$ is $\mathcal{O}(k^2)$. Thus, the above algorithm for computing $Q^{(k)}$ through $\mathcal{C}_k$ is in fact exponential in k, since k is represented in a binary format. However, as we show by the next theorem, this is the best one could do unless $P = NP$. In fact, our suggested incremental computation of the certain answer is a parametrically optimal procedure.

Theorem 9. *Our algorithm for computing $Q^{(k)}$ is optimal, under the assumption that $P \neq NP$.*

Proof. First we show the following lemma.

Lemma 6. *Given k, to decide whether or not there exists $(w, k) \in Q$ is NP-hard.*

Proof. We show this via a reduction from the *subset sum* problem 13 of [11], which is as follows. Given natural numbers n_1, ..., n_m, and n, determine whether or not $n = n_{i_1} + \ldots + n_{i_k}$, for some $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, m\}$.

To decide this, we build a weighted query automaton $\mathcal{A}_m = (P, \Delta, \tau, p_0, \{p_m\})$, where $P = \{p_0, p_1, \ldots, p_m\}$, $\Delta = \{r_1, \ldots, r_m\}$, and

$$\tau = \{(p_{i-1}, r_i, n_i, p_i) : 1 \leq i \leq m\} \cup \{(p_{i-1}, r_i, 0, p_i) : 1 \leq i \leq m\}.$$

Observe that, for each word w accepted by $\mathcal{A}_m$ there is only one path spelling w in $\mathcal{A}_m$.

Then, to decide whether or not there exists $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, m\}$, for which $n = n_{i_1} + \ldots + n_{i_k}$, is equivalent with deciding whether or not there exists a word w, such that $(w, n) \in [\mathcal{A}_m]$.

We remark that our reduction is similar to the reduction used in [27] for showing NP-hardness for the exact path length problem. □

Let us now return to the proof of the theorem. Suppose we can generate Q^k in polynomial time in k. Now, let us take a source collection $\mathcal{S} = \{(a, s, b)\}$ with definition $S = \{w : (w, n) \in Q^k\}$. It is easy to see that $(a, b, k) \in \mathit{CAns}(Q, \mathcal{S}, \mathscr{R})$ iff there exists $(w, k) \in Q$. From this, and the above lemma, it follows that it is unlikely (unless $P = NP$) that there exists a polynomial in k algorithm for deciding whether or not $(a, b, k) \in \mathit{CAns}(Q, \mathcal{S}, \mathscr{R})$. Finally, from all the above and Theorem 3, our claim follows. □

7.2 Qualitative Case

Here, computing a Q^k query sphere coincides with computing Q^{k^∞} in the hybrid case (see previous section). The procedure for computing query spheres is repeated as many times as the number of different annotations in the query automaton, *i.e.* the number of repetitions does not depend on k. Hence, we conclude that to compute the certain answer is polynomial in k for the qualitative case.

7.3 Hybrid Case

For computing a query sphere Q^y, where $y = n^{(k)}$, for $n, k \in \mathbb{N}$, we need to extract from a query automaton all the paths (not necessary simple) with (a) any number of transitions weighted strictly less than n, and (b) not more than k transitions weighted exactly n.

For this, we build a mask automaton $\mathcal{M}_{n,k}$ as follows:

$$\mathcal{M}_{n,k} = (P_{n,k}, \{0, 1, \ldots, n\}, \tau_{n,k}, p_0, F_{n,k}),$$

where $P_{n,k} = F_{n,k} = \{p_0, p_1, \ldots, p_k\}$, and

$$\begin{aligned}\tau_{n,k} = &\{(p_i, m, p_i) : 0 \leq m < n \text{ and } 0 \leq i \leq k\} \cup \\ &\{(p_i, n, p_{i+1}) : 0 \leq i < k\}.\end{aligned}$$

As an example, we give $\mathcal{M}_{3,2}$ in Fig. 2. We show that

Theorem 10. *$\mathcal{M}_{n,k}$ accepts* all *and* only *the number sequences with cumulative weight $\preceq n^{(k)}$.*

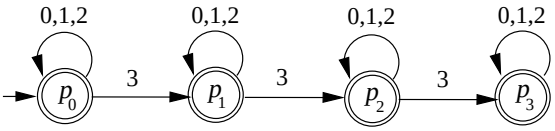

Fig. 2. Automaton $\mathcal{M}_{3,2}$

Proof. Due to its construction, $\mathcal{M}_{n,k}$ recognizes the language of number sequences having not more than k occurrences of n, each surrounded by any number of 0, 1, ..., $n-1$ numbers. Clearly, any sequence in this language has a weight which is $\preceq n^{(k)}$.

Now, let σ be a number sequence with $weight(\sigma) \preceq n^{(k)}$. Let $h \leq k$ be the number of occurrences of n in σ. Due to its construction, $\mathcal{M}_{n,k}$ does not change state upon reading any of 0, 1, ..., $n-1$ numbers. On the other hand, $\mathcal{M}_{n,k}$ jumps to the next higher numbered state upon reading an n. Thus, when reading σ, which has h occurrences of n, $\mathcal{M}_{n,k}$ will finally be in state p_h, i.e. $\mathcal{M}_{n,k}$ does not get stuck when reading σ. Since all the states are final, $\mathcal{M}_{n,k}$ accepts σ, and this concludes our proof. □

Now using $\mathcal{M}_{n,k}$, similarly with the quantitative case, we can extract from an automaton $\mathcal{A}$ for Q all the transition paths weighted less or equal to $n^{(k)}$, giving so an effective procedure for computing the $Q^{n^{(k)}}$ query sphere.

Observe that the above algorithm for computing $Q^{n^{(k)}}$ is polynomial in n, but unfortunately exponential in k (due to a binary representation of n). It is open whether or not $Q^{n^{(k)}}$ can be computed in better time with respect to k.

8 Query Containment and Equivalence

In this section, we focus on query containment and equivalence, which are very important reasoning services employed in a multitude of applications.

For two queries Q_1 and Q_2 annotated over a preference semiring $\mathscr{R}$, we say that Q_1 is *contained* in Q_2, and denote this by $Q_1 \sqsubseteq Q_2$, iff for each database DB, $(a, b, x) \in Ans(Q_1, DB, \mathscr{R})$ implies that there exists $y \preceq x$ such that $(a, b, y) \in Ans(Q_2, DB, \mathscr{R})$. If both $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$ are true, we say that Q_1 is equivalent with Q_2 and write $Q_1 \equiv Q_2$. For the Boolean case, the above correspond to the classical query containment and equivalence. This is because for $\mathscr{B}$, we have $T \prec F$.

On the other hand, we have the language-semiring theoretic notion of containment and equivalence. Namely, we say that Q_1 is $\mathcal{L}$-*smaller* than Q_2, and denote this by $Q_1 \sqsubseteq_{\mathcal{L}} Q_2$ iff $(w, x) \in Q_1$ implies $(w, y) \in Q_2$ and $y \preceq x$. If both $Q_1 \sqsubseteq_{\mathcal{L}} Q_2$ and $Q_2 \sqsubseteq_{\mathcal{L}} Q_1$ are true, we say that Q_1 is equivalent with Q_2 and write $Q_1 \equiv_{\mathcal{L}} Q_2$. We are able to show the following characterization theorem.

Theorem 11.

1. $Q_1 \sqsubseteq Q_2$ iff $Q_1 \sqsubseteq_{\mathcal{L}} Q_2$
2. $Q_1 \equiv Q_2$ iff $Q_1 \equiv_{\mathcal{L}} Q_2$.

Proof. We only show (1) as (2) directly follows from (1).

"If." This direction is easy to verify.

"Only if." Let $(w, x) \in Q_1$, where $w = r_1 \dots r_n$. Construct a ("linear") database DB as $a_1 \xrightarrow{r_1} a_2 \xrightarrow{r_2} \dots \xrightarrow{r_{n-1}} a_n \xrightarrow{r_n} a_{n+1}$. Clearly, $(a_1, a_{n+1}, x) \in Ans(Q_1, DB, \mathscr{R})$. Since in this (proof) direction we assume $Q_1 \sqsubseteq Q_2$, we have $(a_1, a_{n+1}, y) \in Ans(Q_2, DB, \mathscr{R})$ and $y \preceq x$. From the definition of query answers, and from the fact that there is only one path from a_1 to a_{n+1} in DB, and this path spells w, we get that $(w, y) \in Q_2$ (where $y \preceq x$). □

Unfortunately, to decide whether or not $Q_1 \sqsubseteq_{\mathcal{L}} Q_2$ is undecidable under quantitative preference semantics. This follows from the undecidability of the equality of rational series in the tropical semiring (cf. [22]). From this, Krob showed that the equivalence problem for distance automata is undecidable. Recall from Section 6 that our annotated query automata for the quantitative case coincide with distance automata. [Equivalence of two distance automata $\mathcal{A}_1$ and $\mathcal{A}_2$, in our framework, means exactly $[\mathcal{A}_1] \equiv_{\mathcal{L}} [\mathcal{A}_2]$.] Thus, we get undecidability of query containment and equivalence for quantitative semantics.

The undecidability is true even for the class of distance automata with 0 and 1 weights only (cf. [22]). On the other hand, it is easy to see that the quantitative semantics coincide with hybrid semantics for queries with 0 and 1 annotations only. Thus, we get undecidability for the containment and equivalence of queries under hybrid semantics as well.

Fortunately, we are able to decide query containment for the case of qualitative preferences as it will become clear soon.

Given a query (language) Q annotated over a semiring $\mathscr{R} = (R, \oplus, \otimes, \mathbf{0}, \mathbf{1})$, we define

1. The *base language* of Q as $\lfloor Q \rfloor = \{w : (w, \_) \in Q\} \subseteq \Delta^*$.
2. The *x-stripe* of Q as $Q^{\dot{x}} = \{(w, x) : (w, x) \in Q\}$, for $x \in R$.
3. The *x-outersphere* of Q as $Q^{\breve{x}} = \{(w, y) : (w, y) \in Q \text{ and } x \preceq y\}$, for $x \in R$.

After these definitions, we give the following general theorem,

Theorem 12. *Let Q_1 and Q_2 be two queries annotated over an NLO semiring $\mathscr{R}$. Then, $Q_1 \sqsubseteq_{\mathcal{L}} Q_2$ iff*

1. $\lfloor Q_1 \rfloor \subseteq \lfloor Q_2 \rfloor$,
2. $\lfloor Q_2^{\dot{x}} \rfloor \cap \lfloor Q_1 \rfloor \subseteq \lfloor Q_1^{\breve{x}} \rfloor$, *for each element x of $\mathscr{R}$.*

Proof. If. Let $(w, x) \in Q_1$. By condition (1), $w \in \lfloor Q_2 \rfloor$, and thus, there exists y in $\mathscr{R}$, such that $(w, y) \in Q_2$. Now, we want to show that $y \preceq x$. For this, observe that $w \in \lfloor Q_2^{\dot{y}} \rfloor$ and since also $w \in \lfloor Q_1 \rfloor$, we have $w \in \lfloor Q_2^{\dot{y}} \rfloor \cap \lfloor Q_1 \rfloor$. By condition (2), $\lfloor Q_2^{\dot{y}} \rfloor \cap \lfloor Q_1 \rfloor \subseteq \lfloor Q_1^{\breve{y}} \rfloor$, i.e. $w \in \lfloor Q_1^{\breve{y}} \rfloor$. The latter means that $(w, x) \in Q_1^{\breve{y}}$, i.e. $y \preceq x$.

Only. If $Q_1 \sqsubseteq_{\mathcal{L}} Q_2$, then, clearly, condition (1) directly follows. Now, let $w \in \lfloor Q_2^{\dot{y}} \rfloor \cap \lfloor Q_1 \rfloor$, for some y in $\mathscr{R}$. From this, we have that $(w, y) \in Q_2$ and $(w, x) \in Q_1$ for some x in $\mathscr{R}$. By the fact that $Q_1 \sqsubseteq_{\mathcal{L}} Q_2$, $y \preceq x$. Thus, $(w, x) \in Q_1^{\breve{y}}$, which in turn means that $w \in \lfloor Q_1^{\breve{y}} \rfloor$. Since y was arbitrary, we have that condition (2) is satisfied as well. □

By the definitions of Q^x, $Q^{\breve{x}}$ and $Q^{\dot{x}}$, we have that

$$\lfloor Q^{\breve{x}} \rfloor = (\lfloor Q \rfloor \setminus \lfloor Q^x \rfloor) \cup \lfloor Q^{\dot{x}} \rfloor.$$

Notably, for all the discrete semirings (which have the *next/previous element* property), such as the semirings we study in this paper, we can also compute $Q^{\dot{x}}$ by computing $\lfloor Q^x \rfloor \setminus \lfloor Q^u \rfloor$, where u is the previous element before x. [Initially, $\lfloor Q^{\mathbf{i}} \rfloor = \lfloor Q^{\mathbf{1}} \rfloor$.]

Observe that that conditions (1) and (2) of Theorem 12 are about containment checks of pure regular languages that we obtain if we ignore the weight of the words in the corresponding annotated languages. Notice however, that Theorem 12 does not give a decision procedure for query containment under quantitative and hybrid preference semantics. This is because the number of query spheres might be infinite for these semantics.

Interestingly, the above theorem gives an effective procedure for deciding query containment under qualitative semantics. This is true because under such semantics, the number of query spheres is finite; this number is bounded by the number of transitions in the query automaton.

8.1 Queries Represented by Deterministic Automata

Quantitative Case. Although as we explained above, in general, the equivalent problem of containment (and equivalence) for distance automata is undecidable, as Hashiguchi et. al. in [17] show, the problem becomes decidable for the class of deterministic (w.r.t. Δ) automata. What they show can be translated as the fact that for deterministic automata

> $[\mathcal{A}_1] \sqsubseteq_{\mathcal{L}} [\mathcal{A}_1]$ iff for all words w on Δ with length at most $2 \times |P_1| \times |P_2|$, where $|P_1|$ and $|P_2|$ are the number of states for $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively, we have that if $(w, x) \in [\mathcal{A}_1]$ then $(w, y) \in [\mathcal{A}_2]$ for $y \preceq x$.

This surely gives a procedure for deciding the containment of queries represented by deterministic annotated automata. Such queries are in fact an important class because as other related work show (cf. [3] and [21]), large classes of weighted NFA's can successfully be determinized.

The complexity of deciding the containment, by naively testing for each word shorter or equal to $2 \times |P_1| \times |P_2|$, is exponential in the size of query automata (Hashiguchi et. al. do not provide a more efficient algorithm).

Let $\mathcal{A}$ be an annotated automaton with n as its the biggest transition weight. Clearly, according to $\mathcal{A}$, any word shorter or equal to $2\times|P_1|\times|P_2|$ cannot have a weight of more than $n\cdot(2\times|P_1|\times|P_2|)$. Now, based on Theorem 12 and the bound by Hashiguchi et. al., we can show that

Theorem 13. *Under quantitative preference semantics, for Q_1 and Q_2 represented by deterministic annotated automata, $Q_1 \sqsubseteq_{\mathcal{L}} Q_2$ if and only if*

1. $\lfloor Q_1 \rfloor \subseteq \lfloor Q_2 \rfloor$,
2. $\lfloor Q_2^{\dot{k}} \rfloor \cap \lfloor Q_1 \rfloor \subseteq \lfloor Q_1^{\check{k}} \rfloor$, *for* $0 \leq k \leq n \cdot (2 \times |P_1| \times |P_2| + 1)$ *where n is the biggest weight among the transition weights in deterministic automata for Q_1 and Q_2.*

Proof. The *only* direction follows directly from Theorem 12.

if. Let $(w, k) \in Q_1$, where $|w| \leq 2 \times |P_1| \times |P_2| + 1$. As such, $k \leq n \cdot (2 \times |P_1| \times |P_2| + 1)$. By condition (1), $w \in \lfloor Q_2 \rfloor$, so $(w, h) \in Q_2$ for some $h \leq n \cdot (2 \times |P_1| \times |P_2| + 1)$.

We want to show that $h \leq k$. By condition (2), (as in the proof of Theorem 12) $\lfloor Q_2^{\dot{h}} \rfloor \cap \lfloor Q_1 \rfloor \subseteq \lfloor Q_1^{\check{h}} \rfloor$, i.e. $w \in \lfloor Q_1^{\check{h}} \rfloor$. The latter means that $(w, k) \in Q_1^{\check{h}}$, i.e. $h \leq k$.

Recall, that the above is true for any $(w, k) \in Q_1$, where $|w| \leq 2 \times |P_1| \times |P_2| + 1$. Hence, by the result of Hashiguchi et. al. (stated above in quote), we have that $Q_1 \sqsubseteq Q_2$. □

Recall that $\lfloor Q^{\ddot{x}} \rfloor = (\lfloor Q \rfloor \setminus \lfloor Q^x \rfloor) \cup \lfloor Q^{\dot{x}} \rfloor$. For discrete semirings, this equals $\lfloor Q^{\ddot{x}} \rfloor = \lfloor Q \rfloor \setminus \lfloor Q^u \rfloor$, where u is the previous element before x. [In particular, $\lfloor Q^{\check{\mathbf{1}}} \rfloor = \lfloor Q \rfloor$.] Also recall that for discrete semirings, $Q^{\dot{x}} = \lfloor Q^x \rfloor \setminus \lfloor Q^u \rfloor$, where u is the previous element before x. [In particular, $\lfloor Q^{\dot{\mathbf{1}}} \rfloor = \lfloor Q^{\mathbf{1}} \rfloor$.]

This means that computing $\lfloor Q^{\dot{x}} \rfloor$ and $\lfloor Q^{\ddot{x}} \rfloor$ amounts to taking set differences with base language $\lfloor Q^u \rfloor$.

Fortunately, if a query Q is represented by a deterministic automaton $\mathcal{A}$, then each query sphere, generated by our mask automaton of the previous section, is deterministic as well. To see this, suppose we are computing sphere Q^k using mask automaton $\mathcal{M}_k$. Let (p_1, r, j, p_2), where $j \leq k$, be a transition in $\mathcal{A}$. This transition will match the j-labeled transitions in $\mathcal{M}_k$. Recall that $\mathcal{M}_k$ is deterministic with respect to its (number) alphabet. Let (q_1, j, q_2) be a j-labeled transition in M_k. Now, in Q^k automaton, we will have only one r-labeled transition outgoing from state (p_1, q_1) because there is only one r-labeled transition outgoing from p_1 in $\mathcal{A}$, and there is only one j-labeled transition outgoing from q_1 in $\mathcal{M}_k$. Similar reasoning applies for each other state of Q^k automaton.

The importance of having deterministic (with respect to Δ) query spheres is that the above differences with query spheres can be computed in polynomial time. Moreover the differences themselves will be represented by deterministic automata, and thus, their containments can be checked in polynomial time as well. Based on all the above, we formally state the following theorem.

Theorem 14. *Deciding containment of deterministic queries under quantitative preference semantics can be done in PTIME.*

Hybrid Case. Regarding the containment of queries under hybrid preference semantics, we show that for the class of deterministic queries, the problem is decidable in pseudo-polynomial time.

For this, we first define the $[n, n^\infty]$-*stripe* of a query Q under hybrid semantics as

$$Q^{[n,n^\infty]} = \{(w, x) \in Q : n \preceq x \prec n + 1\}.$$

The $[n, n^\infty]$-stripe of Q can be obtained by intersecting Q with mask automaton

$$\mathcal{M}_{n,n^\infty} = (\{p_0, p_1\}, \{0, 1, \ldots, n\}, \tau_n, p_0, \{p_1\}),$$

where

$$\tau_n = \{(p_0, m, p_0) : 0 \leq m < n\} \cup \{(p_0, n, p_1)\} \cup \{(p_1, m, p_1) : 0 \leq m \leq n\}.$$

As an example, we give $\mathcal{M}_{3,3^\infty}$ in Fig. 3.

The two-state automaton $\mathcal{M}_{n,n^\infty}$ accepts all the number sequences with numbers which are less or equal to n and having at least one occurrence of n in them.

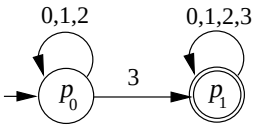

Fig. 3. Automaton $\mathcal{M}_{3,3^\infty}$

We also define the *intersection* of an annotated language Q with a pure language L on Δ as

$$Q \cap L = \{(w, x) : (w, x) \in Q \text{ and } w \in L\}.$$

If $\mathcal{A} = (P^{\mathcal{A}}, \Delta, \mathscr{R}, \tau^{\mathcal{A}}, p_0^{\mathcal{A}}, F^{\mathcal{A}})$ (annotated) and $\mathcal{B} = (P^{\mathcal{B}}, \Delta, \tau^{\mathcal{B}}, p_0^{\mathcal{B}}, F^{\mathcal{B}})$ (regular) are automata for Q and L, respectively, then an (annotated) automaton for $Q \cap L$ can be easily obtained by taking the Cartesian product of $\mathcal{A}$ and $\mathcal{B}$ as $(P^{\mathcal{A}} \times P^{\mathcal{B}}, \Delta, \mathscr{R}, \tau, (p_0^{\mathcal{A}}, p_0^{\mathcal{B}}), F^{\mathcal{A}} \times F^{\mathcal{A}})$, where

$$\tau = \{((p, q), r, x, (p', q')) : (p, r, x, p') \in \tau^{\mathcal{A}} \text{ and } (q, r, q') \in \tau^{\mathcal{B}}\}.$$

Now, we show that

Theorem 15. *Let Q_1 and Q_2 be two queries, and $\mathcal{A}_1$ and $\mathcal{A}_2$ be two automata recognizing Q_1 and Q_2, respectively, Let $n \in \mathbb{N}$ be the biggest weight that occurs among transitions of $\mathcal{A}_1$ and $\mathcal{A}_2$.*

We have that $Q_1 \sqsubseteq_{\mathcal{L}} Q_2$, if and only if, for each $m \in \mathbb{N}$ such that $m \leq n$

1. $\left\lfloor Q_1^{m^\infty} \right\rfloor \subseteq \left\lfloor Q_2^{m^\infty} \right\rfloor$
2. $Q_1^{[m,m^\infty]} \cap \left\lfloor Q_2^{[m,m^\infty]} \right\rfloor \sqsubseteq_{\mathcal{L}} Q_2^{[m,m^\infty]}$.

Proof. First observe that, by the definition of hybrid semantics, Q_1 and Q_2 cannot have words with a weight equal or more than $n + 1$.

If. Let $(w, x) \in Q_1$, where $x = m^k$, for $m, k \in \mathbb{N}$ and $m \leq n$. As such, $w \in \left\lfloor Q_1^{m^\infty} \right\rfloor$, and from condition (1), we have $(w, y) \in Q_2^{m^\infty}$ for some y. We want to show that $y \preceq x$.

If $w \notin \left\lfloor Q_2^{[m,m^\infty]} \right\rfloor$, then $y \prec m \preceq x$, and our claim follows. On the other hand, if $w \in \left\lfloor Q_2^{[m,m^\infty]} \right\rfloor$, then $(w, x) \in Q_1^{[m,m^\infty]} \cap \left\lfloor Q_2^{[m,m^\infty]} \right\rfloor$, and by condition (2), we have that $(w, y) \in \left\lfloor Q_2^{[m,m^\infty]} \right\rfloor$ and $y \preceq x$.

Only. Condition (1) follows straightforwardly from $Q_1 \sqsubseteq Q_2$ and the definitions of $Q_1^{m^\infty}$ and $Q_2^{m^\infty}$. Now, let $(w, x) \in Q_1^{[m,m^\infty]} \cap \left\lfloor Q_2^{[m,m^\infty]} \right\rfloor$, for some $m \leq n$. We have $x = m^{(k)}$ for some $k \in \mathbb{N}$, and there exists $y = m^{(l)}$, for some $l \in \mathbb{N}$, such that $(w, y) \in Q_2^{[m,m^\infty]}$. From $Q_1 \sqsubseteq_{\mathcal{L}} Q_2$, we have that $x \preceq y$, which concludes the proof in this direction. □

From Section 6, we know how to obtain generalized query spheres Q^{n^∞}. Namely, this was done by removing all the transitions weighted by $n+1$ or more from an automaton for Q. The obtained automata are deterministic if the query automata are such.

Also, the mask automata $\mathcal{M}_{n,n^\infty}$ are deterministic. Hence, similarly as for the quantitative case, we are able to formally state our last theorem regarding deterministic queries under hybrid preference semantics.

Theorem 16. *Deciding containment of deterministic queries under hybrid preference semantics can be done in pseudo-polynomial time.*

Proof. From the above discussion, we have that all the automata for the languages mentioned in Theorem 15 are deterministic if the query automata are deterministic. Furthermore, their sizes are polynomial in the size of Q_1 or Q_2.

Thus, condition (1) can be checked in polynomial time in the size of Q_1 and Q_2. Regarding condition (2), observe that $Q_1^{[m,m^\infty]} \cap \left\lfloor Q_2^{[m,m^\infty]} \right\rfloor$ and $Q_2^{[m,m^\infty]}$ are annotated regular languages, whose words are weighted by weights $m^{(k)}$, where $k \in \mathbb{N}$. By the nature of the hybrid semiring, when we consider only one range, say m, of elements, what we get is the tropical semiring. By Theorem 13, we know that we can decide containment of deterministic queries annotated over the tropical semiring in polynomial time.

Hence, both conditions (of Theorem 15) can be checked in polynomial time in the size of Q_1 and Q_2.

Now, we have to check these conditions n times. This gives us an algorithm which runs in pseudo-polynomial time (due to a binary representation of n). □

9 Conclusions

We have introduced a general semiring framework for defining preferential regular path queries. Using this framework, we have formalized three useful preference semantics, namely the quantitative, qualitative, and hybrid semantics. We have derived general results (whenever possible) and specialized results for each preference semantics. For all three semantics, we were able to devise methods for progressive query answering on a given database, as well as on a given set of sound views. Furthermore, we showed that for a large class of queries, the query containment is indeed decidable.

All these encourage the use of preferential regular path queries. Naturally, a possible direction for future work is to investigate other application-dependent preference semantics. We believe that our general semiring framework would unify and facilitate such work.

Acknowledgment. We would like to thank the ICDT'07 reviewers, a reviewer for Fundamenta Informaticae, and the corresponding editor Thomas Schwentick, for their constructive comments.

References

1. Abiteboul S., P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann, San Francisco, CA, 1999.

2. Aho A., J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.
3. Allauzen C., M. Mohri. An optimal pre-determinization algorithm for weighted transducers. *Theoretical Computer Science* 328 (1–2): 3–18, 2004.
4. Allauzen C., M. Mohri. A Unified Construction of the Glushkov, Follow, and Antimirov Automata. *Proc. of 31st International Symposium on Mathematical Foundations of Computer Science (MFCS'06)*, Lecture Notes in Computer Science 4162: 110–121, Springer, 2006.
5. Berstel J. Jr. and C. Reutenauer. *Rational Series and Their Languages.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, New York, NY, 1988.
6. Calvanese D., G. Giacomo, M. Lenzerini, and M. Y. Vardi. Answering Regular Path Queries Using Views. *Proc. of 16th International Conference on Data Engineering (ICDE'00)*: 389–398, IEEE, 2000.
7. Calvanese D., G. Giacomo, M. Lenzerini, and M. Y. Vardi. View-Based Query Processing and Constraint Satisfaction. *Proc. of 15th Annual IEEE Symposium on Logic in Computer Science (LICS'00)*: 361–371. IEEE, 2000.
8. Calvanese D., G. Giacomo, M. Lenzerini, and M. Y. Vardi. View-based Query Processing: On the Relationship between Rewriting, Answering and Losslessness. *Proc. of 10th International Conference on Database Theory (ICDT'05)*: 321–336, Lecture Notes in Computer Science 3363, Springer, 2005.
9. Chomicki J. Preference formulas in relational queries. *ACM Transactions on Database Systems* 28 (4): 427–466, ACM, 2003.
10. Flesca S., F. Furfaro, and S. Greco. Weighted Path Queries on Semistructured Databases. *Information and Computation*, 204 (5): 679–696, 2006.
11. Garey M. R., and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, NY, 1979.
12. Grahne G., and A. Thomo. Query Containment and Rewriting Using Views for Regular Path Queries under Constraints. *Proc. of 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'03)*: 111–122, ACM, 2003.
13. Grahne G., and A. Thomo. Regular Path Queries under Approximate Semantics. *Annals of Mathematics and Artificial Intelligence*, 46 (1-2): 165–190, 2006.
14. Hashiguchi K. Limitedness Theorem on Finite Automata with Distance Functions. *Journal of Computer and System Sciences* 24 (2): 233–244, 1982.
15. Hashiguchi K. Improved Limitedness Theorems on Finite Automata with Distance Functions. *Theoretical Computer Science* 72 (1): 27–38, 1990.
16. Hashiguchi K. New Upper Bounds to the Limitedness of Distance Automata. *Theoretical Computer Science* 233 (1-2): 19–32, 2000.
17. Hashiguchi K., K. Ishiguro, and S. Jimbo. Finitely Ambiguous Finance Automata. *International Journal of Algebra and Computation* 12 (3): 445–461, 2002.
18. Hopcroft J. E., and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.
19. Kiesling W., B. Hafenrichter, S. Fischer, and S. Holland. Preference XPATH – A query language for E-commerce. *Proc. of 5th Intern. Konf. für Wirtschaftsinformatik*: 425–440, Augsburg, Germany, 2001.
20. Kirsten D. Distance Desert Automata and the Star Height Problem. *Theoretical Informatics and Applications* 39 (3): 455–510, 2005.
21. Kirsten D., and I. Mäurer. On the Determinization of Weighted Automata. *Journal of Automata, Languages and Combinatorics* 10 (2–3): 287–312, 2005.
22. Krob D. The Equality Problem for Rational Series with Multiplicities in the Tropical Semiring is Undecidable. *International Journal of Algebra and Computation* 4 (1): 405–425, 1994.
23. Kuich W., and A. Salomaa. *Semirings, Automata, Languages.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, London, UK, 1985.
24. Lenzerini M. Data Integration: A Theoretical Perspective. *Proc. of 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'02)*: 233-246, ACM, 2002.
25. Leung H. Limitedness Theorem on Finite Automata with Distance Functions: An Algebraic Proof. *Theoretical Computer Science* 81 (1): 137–145, 1991.
26. Mendelzon A. O., and P. T. Wood. Finding Regular Simple Paths in Graph Databases. *SIAM Journal on Computing* 24 (6): 1235–1258, 1995.
27. Nykänen M., and E. Ukkonen. The Exact Path Length Problem. *Journal of Algorithms* 42 (1): 41–53, 2002.

28. Rondogiannis P., and W. W. Wadge. Minimum Model Semantics for Logic Programs with Negation-as-failure. *ACM Transactions on Computational Logic* 6 (2): 441–467, 2005.
29. Ruchi A. A Framework for Expressing Prioritized Constraints Using Infinitesimal Logic *Master Thesis*, University of Victoria, BC, Canada, 2005.
30. Salomaa A., and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series.* Texts and Monographs in Computer Science. Springer-Verlag, New York, NY, 1978.
31. Simon I. On Semigroups of Matrices over the Tropical Semiring. *Informatique Theorique et Applications* 28 (3-4): 277–294, 1994.
32. Stefanescu D. C., A. Thomo A, and L. Thomo. Distributed Evaluation of Generalized Path Queries. *Proc. of 20th ACM Symposium on Applied Computing (SAC'05)*: 610–616, ACM 2005.
33. Stefanescu D. C., A. Thomo. Enhanced Regular Path Queries on Semistructured Databases. *Proc. of 11th International Workshop on Foundations of Models and Languages for Data and Objects: "Query Languages and Query Processing" (QLQP'05), EDBT'06 Workshops*: 700–711, Lecture Notes in Computer Science 4254, Springer, 2006.
34. Russell J. S., P. Norvig. *Artificial Intelligence: A Modern Approach.* (2nd Edition) Prentice Hall, Upper Saddle River, NJ, 2002.
35. Ullman J. D. *Principles of Data and Knowledge Bases, Vol. II: The New Technologies*. W. H. Freeman & Co, New York, NY, 1990.
36. Vardi. M. Y. A Call to Regularity. *Proc. of PCK50 - Principles of Computing & Knowledge, Paris C. Kanellakis Memorial Workshop '03*: p. 11, ACM, 2003.
37. von Wright G. H. *The Logic of Preference.* Edinburgh University Press, 1963.