Preferentially Annotated 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 semantics specializations 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.

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

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

Is now a pair of objects connected by a path with two *greyhound* segments equally equally preferrable 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 preferrable 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 preferrable to itineraries containing *aircanada* segments, while these itineraries are equally preferrable, 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 preferrable 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. [2, 3][3]) and reasoning about query answering in this setting is a very important technology. 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 prefential 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.

Due to space constraints we omit this third part. The interested reader can find it in the full version of the paper available online (see [7]). Also, the full proofs of most of our results can be found at this online reference.

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

[3] For the semistructured data case.

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. Finally, in Section 7, we present algorithms for computing query spheres under the different preference semantics.

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 [4]. 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 [4].

Preferences for XML are studied by [9]. These preferences are aimed at comparing attribute values of XML elements rather than structure of (parts of) documents. As characterized by [4], the preferences of [9] 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 [11]. However, [11] focuses on the relational case only.

In [5], weighted path queries are introduced. Syntactically, such queries are the same as our preferentially annotated queries. However, [5] do not give any semantics on their queries. Technically, one can use their query answering algorithm to answer our queries on a given database. However, we carefully study some important details of query answering, which are not taken into consideration in [5]. Moreover, query answering on a given database is not our most important contribution in this paper.

Regarding query answering (on a given database), one can also use, assuming quantitative semantics only, the algorithm of [6] for queries under distortions. In that paper, there are also some technical results, which can be adapted to help in some of our derivations. However, we do not do this, due to the high computational complexity of constructs in [6]. 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, [12] and [13] deal with distributed evaluation of weighted regular path queries. However, the algorithms of [12] and [13] apply to quantitative semantics only. We believe that they can also be adapted for other semantics as well, and thus, [12, 13] 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.

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 anihilator 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 $x \oplus y = x$. It is easily verified that $\preceq$ is a partial order.

In this paper, we will in addition require for semirings of preferences to have a *total* natural order. All the preference semirings mentioned in Introduction posses such an order.[4] Observe that $\mathbf{0}$ is the "biggest" element of the semiring, and it corresponds to the "infinitely worst" preference weight.

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

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

[4] We want to note here that for database paths, it is difficult to find intuitively plausible preference semantics, which would ask for a partial order only.

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."

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

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

Given a 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

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

Intuitively, 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 "and," and "or" 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

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

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

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

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

In the case of *qualitative preferences*, we have

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

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 $1^{(1)}, 2^{(1)}, \ldots$.] Intutitively, 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 with both of them. Similarly with 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 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 NFA that accepts an RPQ Q. Starting from an object a of a database DB, we first create

the pair (p_0, a), where p_0 is the initial state in $\mathcal{A}$. Then, we create all the pairs (p, b) such that there exist a transition from p_0 to p in $\mathcal{A}$, and an edge from a to b in DB, and furthermore the labels of the transition and the edge match. In the same way, we continue to create new pairs from existing ones, 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 contructed depending on the selectivity of the query. The implicit assumption in [1] is that this part of the Cartesian product fits in main memory and each object is not accessed more than once in secondary storage.

After obtaining the above Cartesian product graph, producing query answers becomes a question of computing reachability of nodes (p, b), where p is a final state, from (p_0, a), where p_0 is the intial state. Namely, if (p, b) is reachable from (p_0, a), 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 (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 (p_0, a) to all the nodes (p, b), where p is a final state in the query automaton $\mathcal{A}$.

In our 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 is progressive, *i.e.* it computes answer tuples (w.r.t. each potential starting object a) in the order of their preference rank. For this, Dijkstra's algorithm is the best choice (compared to Flloyd-Warshall algorithm). It fits perfectly with the lazy strategy of constructing the Cartesian product graph, and it reaches the b objects in a "best first" fashion. Our general algorithm, which works with all the proposed preference semirings is as follows.

Algorithm 1
Input: An ϵ-free automaton $\mathcal{A}$ for an $\mathscr{R}$-annotated query Q, and a database DB.
Output: $Ans(Q, DB, \mathscr{R})$.
Method: For each *potential* start object a[5] compute the set $Reach_a$ as follows.

1. Initialize $Reach_a$ to $\{(p_0, a, \mathbf{1}, \mathit{false})\}$.
2. Repeat 3–5 until $Reach_a$ no longer changes.
3. Choose a quadruple $(p, b, x, \mathit{false}) \in Reach_a$, such that

$$x = \oplus\{y : (p, b, y, \mathit{false}) \in Reach\}.$$

Update $(p, b, x, \mathit{false})$ to (p, b, x, true).

[5] Finding potential start objects can be facilitated by classical indexes on the database edge labels.

4. If p is a final state, then insert (a, b, x) in $Ans(Q, DB, \mathscr{R})$.
5. If there is a transition (p, r, y, q) in $\mathcal{A}$ and there is an edge $b \xrightarrow{r} c$ in DB then add $(q, c, x \otimes y, \mathit{false})$ to $Reach_a$.

5 Preferentially Ranked Answers on Possible Databases

In a semistructured LAV data integration system (cf. [2, 10, 3]), 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 $\mathit{def}(s_i) = S_i$. The mapping or substitution[6] *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.

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). 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 $\mathit{poss}(\mathcal{S})$ of databases over (D, Δ) as follows:

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

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 $\wedge$

[6] In a language theoretic terminology.

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 [2].

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.[7] 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.

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 $(a, b, y) \in Ans(Q, DB, \mathscr{R})$. 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

[7] 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.

let us discuss the certain answer considering the semirings for the quantitative, qualitative, and hybrid preference semantics.

In the quantative 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 [2], 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

$$Q^y = \{(w, x) \in \Delta^* \times R : (w, x) \in Q \text{ and } x \preceq y, \text{ or otherwise } y = \mathbf{0}\}.$$

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 $x \preceq y$.[8]

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

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

$CAns(Q, \mathcal{S}, \mathscr{R})$. We give the following definition about the "next element" property of a semiring.

A 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. Let $\mathscr{R}$ be a discrete semiring. Also, let y (as above) be the next element after (some) x. We can show that

Theorem 1.

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

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 [2] $CAns(Q^y, \mathcal{S}, \mathscr{B})$ 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 1 for generating (a, b, y) tuples of the certain answer? For this, let $z = \odot\{x : (w, x) \in Q\}$. We state the following theorem, which can be easily verified.

Theorem 2. $Q^z = Q$.

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 1 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 later problem is widely known and positively solved in the literature (cf. for example [8]).

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. 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*.

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 $i \in \mathbb{N}$, such that $z_n = n^{(i)}$, 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 $Q^{(n^{(i)})}$.

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 query spheres $Q^{n^{(\infty)}} = \bigcup_{i=0}^{\infty} Q^{n^{(i)}}$. If z_n is finite, then there exists $j \in \mathbb{N}$, such that $Q^{n^{(\infty)}} = \bigcup_{i=0}^{j} Q^{n^{(i)}}$. But, the existence of such j can be found by deciding the limitedness of an automaton for $Q^{n^{(\infty)}}$. Thus, we state that

Theorem 3. *z_n is finite* iff *$Q^{n^{(\infty)}}$ is limited in distance.*

The question is, how can we compute $Q^{n^{(\infty)}}$? 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 $n + 1$. In order to perform this extraction, we build a one-state mask automaton $\mathcal{M}_n$ on the alphabet $\{0, 1, \ldots, n\}$. Let $\tau_{\mathcal{A}}$ be the transition relation of the query automaton $\mathcal{A}$. Then, $\mathcal{M}_n = (\{q\}, \{0, 1, \ldots, n\}, q, \tau_n, \{q\})$, where $\tau_n = \{(q, m, q) : (p, r, m, p') \in \tau_{\mathcal{A}} \text{ and } m \leq n\}$.

Finally, we construct a Cartesian product automaton

$$\mathcal{C}_n = \mathcal{A} \times \mathcal{M}_n = (P_{\mathcal{A}} \times \{q\}, \Delta, \tau, (p_0, q), F_{\mathcal{A}} \times \{q\}),$$

where $\tau = \{((p, q), r, n, (p', q)) : (p, r, m, p') \in \tau_{\mathcal{A}} \text{ and } (q, m, q') \in \tau_n\}$. It can be shown that

Theorem 4. *The weighted automaton $\mathcal{C}_n$ accepts exactly $Q^{n^{(\infty)}}$.*

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

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

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

The automaton $\mathcal{M}_k$ has a nice property. It captures all the possible paths (unlabeled with respect to Δ) with weight equal to k. Formally, we can show that

Theorem 5. *$\mathcal{M}_k$ contains* all *the possible paths π with weight$(\pi) \leq k$, and it does not contain any path with weight greater than k.*

Now by using $\mathcal{M}_k$, we can extract from a weighted automaton $\mathcal{A}$ for Q all the 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 can show that

Theorem 6. *The weighted automaton $\mathcal{C}_k$ accepts exactly the k-th sphere $Q^{(k)}$ of query Q.*

It can be easily seen 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. We can show that

Theorem 7. *Our algorithm for computing $Q^{(k)}$ is essentially optimal.*

Qualitative Case. Here the mask automaton is polynomial in k, and it coincides with the mask automaton for 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.

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

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

Formally, we can show that

Theorem 8. *$\mathcal{M}_k$ contains* all *the possible paths π with weight$(\pi) \leq n^{(k)}$, and it does not contain any path with weight greater than $n^{(k)}$.*

Now by using $\mathcal{M}_{n,k}$, similarly with the previous cases, 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.

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. Calvanese D., G. Giacomo, M. Lenzerini, and M. Y. Vardi. Answering Regular Path Queries Using Views. *Proc. of ICDE '00.*
3. Calvanese D., G. Giacomo, M. Lenzerini, and M. Y. Vardi. View-based Query Processing: On the Relationship between Rewriting, Answering and Losslessness. *Proc. of ICDT '05.*
4. Chomicki J. Preference formulas in relational queries. *ACM Trans. Database Syst.* 28 (4) : 427–466, 2003.
5. Flesca S., F. Furfaro, and S. Greco. Weighted Path Queries on Web Data. *Proc. of WebDB '01.*
6. Grahne, G., and A. Thomo. Query Answering and Containment for Regular Path Queries under Distortions. *Proc. of FoIKS '04.*
7. Grahne, G., A. Thomo, W. Wadge. Preferentially Annotated Regular Path Queries. `http://www.cs.uvic.ca/~thomo/papers/icdt2007full.pdf`
8. Hashiguchi K. Limitedness Theorem on Finite Automata with Distance Functions. *Journal of Computer and System Sciences* 24 (2) : 233–244, 1982.
9. Kiesling W., B. Hafenrichter, S. Fischer, and S. Holland. Preference XPATH – A query language for E-commerce. *Proc. of the 5th Int'l Conf. Wirtschaftsinformatik*, Augsburg, Germany, 2001.
10. Lenzerini M. Data Integration: A Theoretical Perspective. *Proc. of PODS '02.*
11. Ruchi A. A Framework for Expressing Prioritized Constraints Using Infinitesimal Logic *Master Thesis* University of Victoria, BC, Canada, 2005.
12. Stefanescu D. C., A. Thomo, and L. Thomo Distributed evaluation of generalized path queries. *Proc. of SAC '05.*
13. Stefanescu D. C., A. Thomo. Enhanced Regular Path Queries on Semistructured Databases. *Proc. of QLQP '05.*