Query Answering and Containment for Regular Path Queries under Distortions

Gösta Grahne[1] and Alex Thomo[2]

[1] Concordia University, Montreal, Canada, grahne@cs.concordia.ca
[2] Suffolk University, Boston, USA, thomo@mcs.suffolk.edu

Abstract. We give a general framework for approximate query processing in semistructured databases. We focus on regular path queries, which are the integral part of most of the query languages for semistructured databases. To enable approximations, we allow the regular path queries to be *distorted*. The distortions are expressed in the system by using weighted regular expressions, which correspond to weighted regular transducers. After defining the notion of weighted approximate answers we show how to compute them in order of their proximity to the query. In the new approximate setting, query containment has to be redefined in order to take into account the quantitative proximity information in the query answers. For this, we define approximate containment, and its variants *k-containment* and *reliable containment*. Then, we give an optimal algorithm for deciding the k-containment. Regarding the reliable approximate containment, we show that it is polynomial time equivalent to the notorious limitedness problem in distance automata.

1 Introduction

The semi-structured data model [ABS99] is now widely used as a foundation for reasoning about a multitude of applications, where the data is best formalized in terms of labeled graphs. Such data is usually found in Web information systems, XML data repositories, digital libraries, communication networks, and so on.

Regarding the query languages for semi-structured data, virtually all of them provide the possibility for the user to query the database through regular expressions. The design of query languages using regular expressions is based on the observation that many of the recursive queries, which arise in practice, amount to graph traversals. In essence, these queries are graph patterns, and the answers to the query are subgraphs of the database that match the given pattern [MW95,ABS99,C+99,C+00]. In particular, the (sub)queries expressed by regular expressions are called *regular path queries*.

For example, for answering the query

$$Q = \_^* \cdot article \cdot \_^* \cdot ref \cdot \_^* \cdot article.hopcroft$$

one should find all the paths having at some point an edge labeled *article*, followed by any number of other edges then by an edge *ref* followed at some point

by an edge *article* and immediately after concluding with an edge labeled with *hopcroft*.

However, we are often willing to live with structural information that is approximate. In other words, the semistructured data represented by a graph database can be an approximation of the real world, rather than an exact representation. On the other hand, the user herself can have an approximate idea and/or knowledge about the world, and this has as a consequence a need for non exact information to be extracted from the database. In both cases the conclusion is that we need to deal with approximate queries and databases, and give approximate answers to the user queries. As an example, suppose that the user gives the above query, but in the database we have edges labeled *papers* instead of *article* or we have recorded in the database only books by Hopcroft, and no papers authored by him. In both cases, the user would get an empty answer under exact query semantics, while it would be very desirable if the system had the ability to substitute *article* by *paper* "for free," and to substitute *article* by *book* with some "cost," say 3. The system could then warn the user about the distortions, by producing a query answer, weighted by the distortion cost. The database system administrator could capture such allowed distortions by building a weighted regular expression

$$(\Delta, 0, \Delta)^*.((article, 0, paper) + (article, 3, book)).(\Delta, 0, \Delta)^*$$

This regular expression is defined over symbol-weight-symbol triplets (R, k, S), where k is the semantic "cost" of distorting R to S [3], and $(\Delta, 0, \Delta)$ is a shorthand for $\sum_{R \in \Delta}(R, 0, R)$, with Δ being the underlying (finite) alphabet. It is easy to see that such extended regular expressions exactly correspond to *weighted transducers*, if we think of the transducers as finite automata over symbol-weight-symbol triplets.

For simplicity, we can also allow the system administrator to use word-weight-word triplets (v, k, w) for building such weighted regular expressions. It can easily be shown that the (corresponding) weighted transducers over such word-weight-word can be transformed into transducers over symbol-weight-symbol triplets. Although the weighted regular expressions over word-weight-word triplets are equivalent to the ones over symbol-weight-symbol triplets, the former are easier to use when we want to capture path structural distortion, as for example

$$(automata.book.author.hopcroft, 0, automata.book.author.ullman).$$

We can even allow full general regular expressions in the above triplets as for example

$$(automata._^*.book._^*.hopcroft, 0, automata._^*.book._^*.ullman).$$

The semantics of such triplets is that we can distort any word in the language of the first regular expression to any database path spelling a word in the language

[3] R or S could be ϵ as well, but not both.

of the second. It can be shown, that we are still able to find an equivalent weighted regular expression over the symbol-weight-symbol triplets.

In this paper, we formally define the notion of weighted approximate answers to regular path queries. Given such a query and having available a weighted transducer (through a weighted regular expression) we show that we can effectively compute all the approximate answers on a database. Furthermore, we can produce the approximate answers in increasing order of their weight, i.e. from the less to the more distorted.

The similar problem of finding approximate patterns in sequence databases is treated in depth in [JMM95]. There, Jagadish, Mendelzon and Milo formalized a very powerful rule-based system through which one can specify the possible allowed distortions of a word to some other word. Unfortunately, their distortion model has an undecidable word problem. Hence, should we use the model of [JMM95], we would not be able to decide in general the membership of a tuple in the approximate answer to a query.

We can say that, the motivation for using weighted regular expressions (i.e. weighted transducers) as a distortion model is similar to the motivation for using regular expressions for querying recursive graph patterns and not the more powerful formalisms such as context-free rule-based grammars.

Having built our query approximation framework, we turn to defining a query containment notion, which takes into account the quantitative distortion information available in the tuples of the answer sets. For this, we define the *approximate containment*, and its variants k-containment, and *reliable* containment. For the first notion, we say that a query is approximately contained in a another query, if for *any* database the tuples for the first query are also tuples for the second one, and furthermore, in the second query, those tuples are more reliable, i.e. they are obtained through less query distortion. The reason behind this view is that, since for obtaining a tuple, the first query needs more distortion than the second one, semantically the first is "smaller" than the second. However, as we show, this unrestricted notion of approximate query containment does not help to much. Hence, in addition, we shall require that on *any* database the (distortion) weight of the tuples for the first query to not be greater than a given number, say k, compared to the weight of the corresponding tuples for the second query. We call this k-containment, and we give an optimal algorithm, based on algebraic properties of automata, for deciding such containment between two given queries.

Depending on the application, we might be interested only in the *existence* of the above number k. Namely, we would like to know, for two given queries and a distortion transducer, whether there exists a number k, such that on any database, the (distortion) weight of the tuples for the first query is not greater than k compared to the weight of the corresponding tuples for the second query. We call this variant reliable containment, and show that it is polynomial time equivalent with the intricate limitedness problem for distance automata, intensely investigated by Hashiguchi and others [Has82,Has90,Has00,Leu91,Sim94].

A practical application for the k- and reliable containment appears in the data integration framework of answering queries using views, which are approximate descriptions of datasources. Namely, suppose that those views have been approximately evaluated and as a consequence their tuples are weighted. Normally, we can use a view V for answering a given user query Q if V is contained in Q, and in such a case we produce the tuples of V as answers for Q. However, by just doing that, we don't have a good quantitative estimation of each tuple reliability in the answer for the query. If a tuple in V has a distortion weight of n, then the weight of this tuple for Q could be any number between n and 0.

Now, if we have available the fact the view V is k-contained in the query Q then we get a narrowed interval for the distortion weight of each tuple produced in the answer for Q. For this, suppose that some tuple t is in V with a weight n. Then, this tuple t will have (with respect to Q) a reliability m, which lies between n and $n-k$, i.e. $n-k \leq m \leq n$.

The above says that the tuple t will never be "better" than $n-k$ in any possible database on which the view V could have been (hypothetically) evaluated. As it is the case of the pure information integration framework, the database on which the view has been evaluated doesn't exist at all, and hence, we should be content with the range $(n-k, n)$ for some tuple t.

However, in the case of warehouses, where the database exists but is expensive to access, we can go in the database and try to approximately evaluate the query only if there exists at least one tuple produced (from the view), whose lower bound of the distortion is less than some cut-off level that the user desires. Otherwise, it doesn't make sense to try to approximately answer the query on the database, since all the tuples that we will get will have greater distortion weight than the cut-off level.

Finishing this discussion, having also an algorithm for finding whether or not V is reliably contained in Q helps in deciding if we should try to find the exact k of this contaiment or not. If yes, then we can use our optimal algorithm to test for each k from 1 to the greatest distortion weight the tuples in V, and find in this way the exact k of the approximate containment of V in Q.

Finally, the more general problem of finding an approximate rewriting of a query having available a set of views is also very interesting and surely will benefit from our notion of approximate query containment. This is one of our directions for future work.

2 Basic Definitions

We consider a database to be an edge labeled graph. This graph model is typical in semistructured data, where the nodes of the database graph represent the objects and the edges represent the attributes of the objects, or relationships between the objects.

Formally, let Δ be a finite 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. Figure 1 shows an example of a database. If there is a path labeled $R_1, R_2, \ldots, R_k$ from a node a to a node b we write $a \xrightarrow{R_1 R_2 \ldots R_k} b$.

Fig. 1. An example of a graph database

A *query* Q is a regular language over Δ. Let Q be a query and $DB = (V, E)$ a database. Then, the *exact answer* to Q on DB is defined as

$$ans(Q, DB) = \{(a, b) : \{a, b\} \subseteq V \times V \text{ and } a \xrightarrow{w} b \text{ in } DB \text{ for some } w \in Q\}.$$

For instance, if DB is the graph in Figure 1, and $Q = \{SR, T\}$, then $ans(Q, DB)$ $= \{(b, d), (d, b), (c, a)\}$.

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$. A *weighted transducer* $\mathcal{T} = (P, \Delta, \tau, P_0, F)$ consists of a finite set of states P, an input/output alphabet Δ, a set of starting states P_0, a set of final states F, and a transition relation $\tau \subseteq P \times \Delta^* \times \Delta^* \times \mathbb{N} \times P$.

An example of a weighted transducer $(\{p_0, p_1, p_2\}, \{R, S, T\}, \tau, \{p_0\}, \{p_2\})$ is shown in Figure 2. Intuitively, for instance $(p_0, RT, RS, 2, p_1) \in \tau$ means that if the transducer is in state p_0 and reads word RT, it emits the word RS at cost 2 and goes to state p_1.

Fig. 2. A weighted transducer.

Given a weighted transducer $\mathcal{T} = (P, \Delta, \tau, P_0, F)$, and a word $u \in \Delta^*$ we say that a word $w \in \Delta^*$ is an *output of $\mathcal{T}$ for u through a k-weighted distortion* if there exists a sequence $(p_0, u_1, w_1, k_1, p_1)$, $(p_1, u_2, w_2, k_2, p_2)$, $\ldots$, $(p_{n-1}, u_n, w_n, k_n, p_n)$ of state transitions of τ, such that $p_1 \in P_0$, $p_n \in F$, $u = u_1 \ldots u_n$, $w = w_1 \ldots w_n$, and $k = k_1 + \cdots + k_n$. We denote the set of all outputs of $\mathcal{T}$ for u (regardless of distortion) by $\mathcal{T}(u)$. For a language $L \subseteq I^*$, we define $\mathcal{T}(L) = \bigcup_{u \in L} \mathcal{T}(u)$. Later we will also use the notation *rel*($\mathcal{T}$) to denote the set of all pairs $(u, w) \in \Delta^* \times \Delta^*$, where w is an output of $\mathcal{T}$ when providing u as input. Similarly, *dom*($\mathcal{T}$) and *ran*($\mathcal{T}$), will be used to denote the domain and range of *rel*($\mathcal{T}$).

Given a weighted transducer $\mathcal{T}$, and words u and w, the *$\mathcal{T}$-distance* between u and w is defined as

$$d_{\mathcal{T}}(u, w) = \begin{cases} inf\{k : w \text{ is an output of } \mathcal{T} \text{ for } u \text{ through a } k\text{-weighted distortion}\} \\ \infty, \text{ if } w \notin \mathcal{T}(u). \end{cases}$$

Now, the *approximate answer* of Q on DB, through a distortion transducer $\mathcal{T}$, is defined as

$$ans_{\mathcal{T}}(Q, DB) = \{(a, b, k) \in V \times V \times \mathbb{N} : \\ k = inf\{d_{\mathcal{T}}(u, w) : u \in Q \text{ and } a \xrightarrow{w} b \text{ in } DB\}\}$$

For example, in the database DB of Figure 1, if $Q = \{RTT\}$, and the distortion transducer is as in Figure 2, then $\mathcal{T}(Q) = \{RSR, RSS\}$. We thus have $ans(Q, DB) = \emptyset$, while $ans_{\mathcal{T}}(Q, DB) = \{(a, d, 2), (c, b, 2)\}$.

For a query Q we want to get also the query itself from the transduction. For this reason we will usually consider that the distortion transducers also have the ability to "leave everything unchanged." This can be easily achieved automatically by the system, which can add to a distortion transducer a new additional initial state, say p_{id}, and the neutral transitions $\{(p_{id}, R, R, 0, p_{id}) : R \in \Delta\}$. Notably, all our lower complexity bounds will be derived by considering this class of distortion transducers, i.e. those having in addition the ability to "leave everything unchanged."

A transducer $(P, \Delta, \tau, P_0, F)$ is said to be in the *standard form* if τ is a relation over $P \times (\Delta \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{N} \times P$. Intuitively, the standard form restricts the input and output of each transition to be only a single letter or ϵ. We call such transitions *elementary* transitions. It is easy to see that any weighted regular transducer is equivalent to a weighted regular transducer in standard form. The transducer transformation is done by applying the following two steps.

In the first step, we eliminate the transitions of the form $(p, w, R_1 \ldots R_n, h, q)$. For this, we introduce new states $p_1, \ldots, p_{n-1}$ and replace such a transition by the elementary transitions (p, w, R_1, h, p_1), $(p_1, \epsilon, R_2, 0, p_2)$, $\ldots$, $(p_{n-1}, \epsilon, R_n, 0, q)$.

In the second step, we eliminate the transitions of the form $(p, S_1 \ldots S_m, R, k, q)$. For this, we introduce new states $p_1, \ldots, p_{m-1}$ and replace such a transition by the elementary transitions (p, S_1, R, k, p_1), $(p_1, S_2, \epsilon, 0, p_2)$, $\ldots$, $(p_{m-1}, S_m, \epsilon, 0, q)$.

3 Computing Approximate Answers

A graph database can be seen as an NFA where the graph nodes are the automaton states and all states are both initial and final. Seen from another perspective, in the "classical" case of exact semantics (see [MW95,ABS99]), computing $ans(Q, DB)$ given the automata $\mathcal{A}_Q$ for Q and $\mathcal{A}_{DB}$ for the database, essentially amounts to constructing the Cartesian product automaton $\mathcal{A}_Q \times \mathcal{A}_{DB}$ (in a lazy way) and outputting the pair (a, b), if and only if there is, in the Cartesian product automaton, an initial state $(\_, a)$ leading to a final state $(\_, b)$.

We show next that for computing $ans_{\mathcal{T}}(Q, DB)$ we can construct an automaton from the Cartesian product of $\mathcal{A}_Q$, $\mathcal{T}$, and $\mathcal{A}_{DB}$. The approximate answer can then be read from this automaton, similarly to the "classical" case.

Let $\mathcal{A}_Q = (P_Q, \Delta, \tau_Q, P_{0_Q}, F_Q)$ be an ϵ-free NFA that accepts Q, and let $\mathcal{T} = (P_{\mathcal{T}}, \Delta, \tau_{\mathcal{T}}, P_{0_{\mathcal{T}}}, F_{\mathcal{T}})$ be the distortion transducer in standard form. Considering the database DB as another ϵ-free NFA, $\mathcal{A}_{DB} = (P_{DB}, \Delta, \tau_{DB}, P_{DB}, P_{DB})$, we construct the transducer $\mathcal{C} = (P, \Delta, \tau, P_0, F)$, where $P = P_Q \times P_{\mathcal{T}} \times P_{DB}$, $P_0 = P_{0_Q} \times P_{0_{\mathcal{T}}} \times P_{DB}$, $F = F_Q \times F_{\mathcal{T}} \times P_{DB}$, and the transition relation τ is defined by, for $(p, q, r) \in P$ and $R, S \in \Delta$,

$$
\begin{aligned}
\tau = \{&((p,q,r), R, S, k, (p',q',r')) : \\
&\quad (p, R, p') \in \tau_Q, (q, R, S, k, q') \in \tau_{\mathcal{T}}, (r, S, r') \in \tau_{DB}\} \cup \\
&\{((p,q,r), \epsilon, S, k, (p,q',r')) : (q, \epsilon, S, k, q') \in \tau_{\mathcal{T}}, (r, S, r') \in \tau_{DB}, p \in P_Q\} \cup \\
&\{((p,q,r), R, \epsilon, k, (p',q',r)) : (p, R, p') \in \tau_Q, (q, R, q', \epsilon, k) \in \tau_{\mathcal{T}}, r \in P_{DB}\}.
\end{aligned}
$$

It is easy to see that $(a, b, k) \in ans_{\mathcal{T}}(Q, DB)$, if and only if there exists, in the graph representation of $\mathcal{C}$, a final state $(\_, \_, b)$ reachable from an initial state $(\_, \_, a)$, and the shortest path between them has cost k.

For shortest paths, both Dijkstra's algorithm and the Floyd-Warshall algorithm (see e.g. [AHU74]) could be used. Although the running times for both Dijkstra's and Floyd-Warshall algorithms are asymptotically the same, perhaps Dijkstra's algorithm is better suited in our scenario. The first reason is that in practice the user might be interested in computing objects reachable only from a limited number of objects, for example when we have a rooted database graph. In such a case, the running time of Dijkstra's algorithm is better, since we don't need to compute the shortest paths between all pairs of objects, as in the Floyd-Warshall algorithm.

The second reason is that most of the time the user is interested only in receiving, say, the 20 best answers. Then, the Dijkstra's algorithm is the ideal choice: It processes the nodes in the order of their distance from the source. Hence, if we could construct the transducer $\mathcal{C}$ on the fly, while at the same time applying the Dijkstra algorithm, then we could stop the execution of the algorithm after 20 iterations.

We can construct the transducer $\mathcal{C}$ on the fly by utilizing a lazy algorithm similar to the lazy query evaluation algorithm of [ABS99], which in essence constructs the Cartesian product of a query with a database. Notably, the Dijkstra

algorithm can be elegantly combined with such a lazy construction of $\mathcal{C}$. By assuming in addition a temporary cache of the "so far reached" objects (the set *reach* in the afore mentioned book), we can avoid accessing the same object in the database more than once. Because of space limitation, we omit the presentation of such a query evaluation algorithm.

4 Containment

In this section, we define and study three notions of containment for regular path queries under approximate semantics.

Recall, that a query Q_1 is (in the usual sense) *contained* in a query Q_2, denoted $Q_1 \sqsubseteq Q_2$ iff $ans(Q_1, DB) \subseteq ans(Q_2, DB)$, for all DB's [GT01]. It is easy to see that this notion of query containment coincides with the (algebraic) language containment of Q_1 and Q_2, i.e. $Q_1 \sqsubseteq Q_2$ iff $Q_1 \subseteq Q_2$. However, under approximate semantics the tuples in the query answers are weighted, and so the containment should take into consideration the tuple weights.

As we know, the smaller the weight of a tuple the better or more reliable it is. For out first notion of containment, we say that a query Q_1 is *approximately contained* in a query Q_2, if for any database the answer-tuples for Q_1 are also answer-tuples for Q_2, and furthermore, under Q_2, those tuples are more reliable. The reason behind this view is that, since for obtaining a tuple, Q_1 needs more distortions than Q_2, semantically Q_1 is "smaller" than Q_2.

However, the approximate query containment is perhaps not very useful. This is because the distance between the corresponding tuples can be arbitrarily large. In other words, it could happen that, for any $n \in \mathbb{N}$ we can find a database such that the tuples obtained for Q_1 on this database, have a distortion weight greater than n compared to the weight of the corresponding tuples for Q_2 obtained on the same database.

Hence, we are also interested in the *quality* of the approximate query containment. For this, we define the *k-containment*, which in addition to approximate containment requires that the weights of the corresponding tuples do not *differ* more than a given number k. Also, depending on the application, the mere existence of such a number k could be useful to know. For this, we define the *reliable containment*, which asks whether or not there exists a number k, for which the k-containment holds. Formally, we have

1. A query Q_1 is *approximately contained* in a query Q_2, denoted $Q_1 \sqsubseteq_{\mathcal{T}} Q_2$, when for *any* database DB, if $(a, b, n) \in ans_{\mathcal{T}}(Q_1, DB)$, then $(a, b, m) \in ans_{\mathcal{T}}(Q_2, DB)$ and $m \leq n$.
2. A query Q_1 is *k-contained* in a query Q_2, denoted $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$, if in the above we also have that $n - m \leq k$.
3. A query Q_1 is *reliably contained* in a query Q_2, denoted $Q_1 \sqsubseteq_{\mathcal{T},\omega} Q_2$, if there exists a $k \in \mathbb{N}$, such that $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$.

Surprisingly enough, the (unbounded) approximate containment does not offer more information in reasoning about queries, than the containment under exact semantics. Namely, we show that for any distortion transducer $\mathcal{T}$

Theorem 1. *$Q_1 \sqsubseteq_{\mathcal{T}} Q_2$ iff $Q_1 \subseteq Q_2$.*

PROOF "If." From the definition of the approximate answers through a distortion transducer, we have that for any database DB, if $(a, b, n) \in ans_{\mathcal{T}}(Q_1, DB)$, there exists a word $w \in Q_1$, and a word u such that $a \xrightarrow{u} b$ in DB, and $d_{\mathcal{T}}(w, u) = n$. Since $Q_1 \subseteq Q_2$, we have that $w \in Q_2$ as well, and so, for sure there exists an m not bigger than n (i.e. $m \leq n$) such that $(a, b, m) \in ans_{\mathcal{T}}(Q_2, DB)$.

"Only if." We show that $Q_1 \sqsubseteq_{\mathcal{T}} Q_2$ implies $Q_1 \subseteq Q_2$. Let $w = R_1 \dots R_h$ be a word in Q_1. We construct a canonical database DB with vertices $\{a, c_1, \dots, c_{h-1}, b\}$ and edges $\{(a, R_1, c_1), \dots, (c_{h-1}, R_h, b)\}$. Clearly, $(a, b, 0) \in ans_{\mathcal{T}}(Q_1, DB)$, and from $Q_1 \sqsubseteq_{\mathcal{T}} Q_2$ we have that $(a, b, m) \in ans_{\mathcal{T}}(Q_2, DB)$, where $m \leq 0$, i.e. $m = 0$. So, there is a word in Q_2 that without being distorted at all can label a path from a to b in DB. Since there is only one path between a and b in DB and this path spells w, we have that $w \in Q_2$. □

In the rest of the paper, we will be interested in the k- and reliable containments because of their practical usability.

Although related, the problems of k-containment and the reliable containment are different. For the k-containment problem, the input is two queries, a distortion transducer, and a fixed number k that the user provides. Then, the question is whether the queries are at most "k steps apart," or not. On the other hand, for the reliable containment problem, k is not part of the input, and the question is existential.

Now, we will define the $\mathcal{T}$-distance between two queries, and then give a necessary and sufficient condition for the k- and reliable containment, based on their $\mathcal{T}$-distance.

Consider a word w on Δ. The $\mathcal{T}$-distance between Q_1 and w is

$$d_{\mathcal{T}}(Q_1, w) = inf\{d_{\mathcal{T}}(u, w) : u \in Q_1\}.$$

Based on that, the $\mathcal{T}$-distance between Q_1 and Q_2 can be naturally defined as

$$d_{\mathcal{T}}(Q_1, Q_2) = sup\{d_{\mathcal{T}}(Q_1, w) : w \in Q_2\}.$$

Returning to our problem, we give the following characterization.

Theorem 2.

1. *$Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ if and only if $Q_1 \subseteq Q_2$ and $d_{\mathcal{T}}(Q_1, Q_2) \leq k$.*
2. *$Q_1 \sqsubseteq_{\mathcal{T},\omega} Q_2$ if and only if $Q_1 \subseteq Q_2$ and there is a $k \in \mathbb{N}$, such that $d_{\mathcal{T}}(Q_1, Q_2) \leq k$.*

PROOF. We will prove only the first claim, since the second one follows directly from the first.

"If." Let DB be a database and $(a, b, n) \in ans_{\mathcal{T}}(Q_1, DB)$. Since $Q_1 \subseteq Q_2$ we have that there exists a $m \leq n$, such that $(a, b, m) \in ans_{\mathcal{T}}(Q_2, DB)$. Now, we want to prove that $n - m \leq k$. Since $(a, b, m) \in ans_{\mathcal{T}}(Q_2, DB)$, there exists a word $w_2 \in Q_2$ such that $d_{\mathcal{T}}(w_2, u) = m$, for some $a \xrightarrow{u} b$ in DB. Now, from

the condition $d_{\mathcal{T}}(Q_1, Q_2) \leq k$, we have that for the word w_2, there exists a word $w_1 \in Q_1$, such that $d_{\mathcal{T}}(w_1, w_2) \leq k$. In plain language, w_1 needs less than k transducer distortions to become w_2. Hence, we finally have

$$\begin{aligned} n &\leq inf\{d_{\mathcal{T}}(w_1, u) : a \xrightarrow{u} b \text{ in } DB\} \\ &\leq k + inf\{d_{\mathcal{T}}(w_2, u) : a \xrightarrow{u} b \text{ in } DB\} \\ &= k + m, \end{aligned}$$

i.e. $n - m \leq k$.

"Only if." The fact that $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ implies $Q_1 \subseteq Q_2$ follows directly from Theorem 1. Now, we continue showing that $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ implies $d_{\mathcal{T}}(Q_1, Q_2) \leq k$. Let $w = R_1 \dots R_l$ be a word in Q_2. We construct a canonical database DB with vertices $\{a, c_1, \dots, c_{l-1}, b\}$ and edges $\{(a, R_1, c_1), \dots, (c_{l-1}, R_l, b)\}$. Clearly, $(a, b, 0) \in ans_{\mathcal{T}}(Q_2, DB)$ and from $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ we have that $(a, b, n) \in ans_{\mathcal{T}}(Q_1, DB)$, where $n \leq k$. Observe that, there is only one path between a and b in DB and this path spells w. So, we have that $d_{\mathcal{T}}(Q_1, w) = n$, i.e. $d_{\mathcal{T}}(Q_1, w) \leq k$. Since w was an arbitrary word in Q_2, we finally get that $d_{\mathcal{T}}(Q_1, Q_2) \leq k$. □

Now we will focus on reasoning about $d_{\mathcal{T}}(Q_1, Q_2)$. Let $\mathcal{A}_1 = (P_1, \Delta, \tau_1, P_{0_1}, F_1)$ and $\mathcal{A}_2 = (P_2, \Delta, \tau_2, P_{0_2}, F_2)$ be two ϵ-free automata for Q_1 and Q_2 respectively, and let $\mathcal{C} = \mathcal{A}_2 \times \mathcal{T} \times \mathcal{A}_1 = (P, \Delta, \tau, P_0, F)$ be a Cartesian product transducer constructed as in Section 3.

For simplicity of exposition, we we will call a sequence of transitions a *path* (not necessarily simple) in the transducer. Suppose that a path π is the sequence of transitions $(p_1, u_1, w_1, k_1, p_2), (p_2, u_2, w_2, k_2, p_3), \dots, (p_n, u_n, w_n, k_n, p_{n+1})$. We say that π *spells* $u_1 \dots u_n$ *as input*, and denote this as $in(\pi) = u_1 \dots u_n$. Additionally, we say that π *spells* $w_1 \dots w_n$ *as output*, and denote this as $out(\pi) = w_1 \dots w_n$. Finally, we say that π has *weight* k, and denote this as $weight(\pi) = k$, if $k = k_1 + \dots + k_n$.

The following lemma says that measuring the distortion through the transducer $\mathcal{T}$ or $\mathcal{C}$ is in essence the same.

Lemma 1. *For $u, w \in \Delta^*$ we have that $d_{\mathcal{T}}(u, w) = d_{\mathcal{C}}(u, w)$*

Proof. From the construction of the Cartesian product transducer $\mathcal{C}$, if there is a path π in $\mathcal{T}$, such that $in(\pi) = u$ and $out(\pi) = w$, then π is also in $\mathcal{C}$ and vice versa. Hence, we get $d_{\mathcal{T}}(u, w) = d_{\mathcal{C}}(u, w)$. Otherwise, if there is not such a path, we can easily see that $d_{\mathcal{T}}(u, w) = d_{\mathcal{C}}(u, w) = \infty$. □

Now, consider the weighted automaton $\mathcal{A}$, that we get if we project out the input column of the transition relation of $\mathcal{C}$. Formally, $\mathcal{A} = (P, \Delta, \tau_{\mathcal{A}}, P_0, F)$, where $\tau_{\mathcal{A}} = \{(p, R, k, q) : (p, S, R, k, q) \in \tau\}$. Let p and q be two states of $\mathcal{A}$, and let π be a path between them. Suppose $out(\pi) = w$. Note that there can be more than one path [4] between p and q spelling w. In reasoning about the

[4] Such paths could have some ϵ-transitions as well.

k-containment we will be interested in the "best" path(s) spelling w, i.e. the one(s) with the smallest weight. Let therefore

$d_{\mathcal{A}}(p, w, q) = inf\{weight(\pi) : \pi$ is a path spelling w and going from p to q in $\mathcal{A}\}$.

Now, we define the *distance* of $\mathcal{A}$, as

$$d(\mathcal{A}) = sup\{d_{\mathcal{A}}(p, w, p) : w \text{ is accepted by } \mathcal{A}, p \in P_0, q \in F\}.$$

Based on the these definitions, Lemma 1, and the construction of the weighted automaton $\mathcal{A}$, the following theorem can be easily verified.

Theorem 3. $d_{\mathcal{T}}(Q_1, Q_2) = d(\mathcal{A})$.

We say that, a weighted automaton $\mathcal{A}$ is k-*limited* (for a given k) if $d(\mathcal{A}) \leq k$, and $\mathcal{A}$ is *limited* if there *exists* a $k \in \mathbb{N}$, such that $\mathcal{A}$ is k-limited.

Now, Theorem 3 along with Theorem 2, say that the k-containment (reliable containment) is reducible to the k-limitedness (limitedness) of weighted automata. Since such an automaton is constructible in polynomial time on the size of Q_1, Q_2, and $\mathcal{T}$, we have that the afore mentioned reduction is polynomial as well.

If we restrict ourselves in weighted automata without ϵ-transitions, we get a class of automata, which are widely known as *distance automata*, and whose limitedness problem is well known for its intricacy. The first solution was obtained by Hashiguchi in 1982, and it gave him the key for solving the star height problem, that had been open for over two decades. Hashiguchi's solution runs in doubly exponential time. By now, it is known that the problem is PSPACE hard [Leu91]. The best known algorithm for deciding whether a distance automaton is limited is by Leung [Leu91] and it runs in single exponential time.

Leung's algorithm is based on the notion of "distance matrices" which can elegantly capture the behavior of distance automata. However, the fact that the distance automata are ϵ-free is of essential importance in using Leung's distance matrices.

In order to make use of Leung's algorithm for deciding the reliable containment (which corresponds to the limitedness problem), we show in Section 6, that we can efficiently transform the transducer $\mathcal{C}$ into one with ϵ-free output transitions, while preserving the semantics of $\mathcal{C}$. Consequently, the automaton $\mathcal{A}$, that we obtain from this ϵ-free output transducer, will be ϵ-free as well.

Unfortunately, Leung's algorithm is unable to decide the k-limitedness of a distance automaton, which in turn is needed to decide the k-containment of queries. Also, as far as the authors know, there is no previous work on k-limitedness of distance automata.

In the next section, we provide an optimal solution to the k-limitedness problem, by automata constructs. Our solution is applicable to general weighted automata, as opposed to only distance automata.

5 Deciding k-Containment

We consider the automaton $\mathcal{A}$, constructed in the previous section, having as weights on its transitions only 0 and 1. If not, we can easily "normalize" it by replacing each transition (p, R, m, q), where $m > 1$, by the sequence of transitions $(p, R, 1, r_1)$, $(r_1, \epsilon, 1, r_2)$, ..., $(r_{m-1}, \epsilon, 1, q)$. For technical reasons, we also add to the transition relation of $\mathcal{A}$ the neutral transitions $(p, \epsilon, p, 0)$ for each state, i.e. self-loops of weight 0, and labeled with ϵ. Evidently, these neutral transitions do not alter any salient features of $\mathcal{A}$. However, we can now assume that for any two transitions in the automaton $\mathcal{A}$ there is always a 0-weighted transition between them.

We will need a few simple operations on automata. Let $\mathcal{A}$ and $\mathcal{B}$ be automata. Then we denote with $\mathcal{A} \cup \mathcal{B}$ the automaton, obtained by the usual construction, recognizing $L(\mathcal{A}) \cup L(\mathcal{B})$. Similarly, $\mathcal{A} \bullet \mathcal{B}$, denotes the automaton recognizing $L(\mathcal{A}).L(\mathcal{B})$.

Now, let's assume that all automata have their states labeled by consecutive integers starting from 1. We denote with $\mathcal{A}_{i,j}$ the automaton obtained from $\mathcal{A}$, by shifting the set of initial states to be $\{i\}$ and the final states to be $\{j\}$. Also, let $\mathbf{0}(\mathcal{A})$ be the automaton obtained from $\mathcal{A}$ by deleting all transitions with cost 1. Finally, for $\{i, j\} \subset \{1, \ldots, n\}$, we consider the set of elementary automata $\mathbf{1}_{i,j}(\mathcal{A})$, each obtained from $\mathcal{A}$ by retaining only transitions between i and j, and only those that have cost 1. Observe that, an automaton, say $(\mathbf{0}(\mathcal{A}))_{i,j}$, can be a full-fledged automaton i.e. with loops, while an elementary automaton, say $\mathbf{1}_{i,j}(\mathcal{A})$, is simple in the sense that it does not contain any loops.

Given a normalized weighted automaton $\mathcal{A} = (\{1, \ldots, n\}, \Delta, \tau, S, F)$, we wish to compute an automaton $\mathbf{k}(\mathcal{A})$, such that $L(\mathbf{k}(\mathcal{A})) = \{w \in L(\mathcal{A}) : d_{\mathcal{A}}(w) \leq k\}$.

Clearly, if we are able to construct $\mathbf{k}(\mathcal{A})$, then we can decide whether or not $d(\mathcal{A}) \leq k$, by testing the language equality $L(\mathbf{k}(\mathcal{A})) = L(\mathcal{A})$. Hence, by this, we cast the decision of the (weighted) k-containment into a pure regular language equivalence test, which can be done in polynomial space.

We will construct $\mathbf{k}(\mathcal{A})$ by a recursive algorithm obtained from the following equations:

$$\mathbf{k}(\mathcal{A}) = \mathcal{A}^0 \cup \mathcal{A}^1 \cup \ldots \cup \mathcal{A}^k$$

where $\mathcal{A}^0 = \mathbf{0}(\mathcal{A})$, and for $1 \leq h \leq k$

$$\mathcal{A}^h = \bigcup_{i \in S, j \in F} \mathcal{A}^h_{i,j}$$

where

$$\mathcal{A}^h_{i,j} = \begin{cases} \bigcup_{m \in \{1,\ldots,n\}} \mathcal{A}^{h/2}_{i,m} \bullet \mathcal{A}^{h/2}_{m,j} & \text{for } h \text{ even} \\ \bigcup_{m \in \{1,\ldots,n\}} \mathcal{A}^{(h-1)/2}_{i,m} \bullet \mathcal{A}^{(h+1)/2}_{m,j} & \text{for } h \text{ odd} \end{cases}$$

for $h > 1$, and

$$\mathcal{A}^1_{i,j} = \bigcup_{\{m,l\} \subset \{1,\ldots,n\}} (\mathbf{0}(\mathcal{A}))_{i,m} \bullet \mathbf{1}_{m,l}(\mathcal{A}) \bullet (\mathbf{0}(\mathcal{A}))_{l,j} \,.$$

We can now show that indeed:

Theorem 4. $L(\mathbf{k}(\mathcal{A})) = \{w \in L(\mathcal{A}) : d_{\mathcal{A}}(w) \leq k\}$.

PROOF. We will prove that for all $0 \leq h \leq k$, the automaton $\mathcal{A}^h$, considered as graph, consists of *all* the paths[5] in $\mathcal{A}$ with weight exactly h, and going from an initial to a final state. So, a word spelled by such a path cannot have distance more than h. As a consequence the automaton

$$\mathbf{k}(\mathcal{A}) = \mathcal{A}^0 \cup \mathcal{A}^1 \cup \ldots \cup \mathcal{A}^k$$

will accept *all* the words in $L(\mathcal{A})$ that cannot have $\mathcal{A}$-distance more than $0, 1, \ldots, k$. From this, our claim follows.

We proceed by induction on h. For $h = 0$, the automaton $\mathcal{A}^0$ is in fact $\mathbf{0}(\mathcal{A})$, so it consists of all the 0-weighted paths in $\mathcal{A}$, going from some initial to some final state. For $h = 1$, the automaton, say

$$\mathcal{A}^1_{i,j} = \bigcup_{\{m,l\} \subset \{1,\ldots,n\}} (\mathbf{0}(\mathcal{A}))_{i,m} \bullet \mathbf{1}_{m,l}(\mathcal{A}) \bullet (\mathbf{0}(\mathcal{A}))_{l,j}\,,$$

consists of the $\mathcal{A}$-paths starting from state i and traversing any number of 0-weighted arcs (transitions) up to some state m, then a 1-weighted arc going to some state l, and after that, any number of 0-weighted arcs ending up in state j. Since m and l range over *all* the possible states, we have that the above described paths are in fact *all* the 1-weighted paths of $\mathcal{A}$ going from state i to state j. Hence, $\mathcal{A}^1$ will consists of all the 1-weighted paths of $\mathcal{A}$ going from an initial to a final state.

For $h = 2$, the automaton, say

$$\mathcal{A}^2_{i,j} = \bigcup_{m \in \{1,\ldots,n\}} \mathcal{A}^1_{i,m} \bullet \mathcal{A}^1_{m,j},$$

consists of concatenations of the 1-weighted paths of $\mathcal{A}$ starting from state i and going to some state m, with the 1-weighted paths of $\mathcal{A}$ starting from that state m and ending up in state j. Since m ranges over all the possible states, these concatenations are in fact *all* the possible 2-weighted paths of $\mathcal{A}$ going from state i to state j. Hence, $\mathcal{A}^2$ will consists of all the 2-weighted paths of $\mathcal{A}$ going from an initial to a final state.

Now we want to show that for some $h > 2$, the automaton, say $\mathcal{A}^h_{i,j}$ consists of all the h-weighted paths of $\mathcal{A}$ going from state i to state j. We assume that this is true for all $\mathcal{A}^m_{i,j}$, when $m < h$. Suppose that h is even. The case when h is odd can be similarly dealt with. We have that

$$\mathcal{A}^h_{i,j} = \bigcup_{m \in \{1,\ldots,n\}} \mathcal{A}^{h/2}_{i,m} \bullet \mathcal{A}^{h/2}_{m,j}.$$

[5] Talking about paths we do not necessarily mean simple paths. Some authors call such paths also walks.

From the induction hypothesis, $\mathcal{A}^{h/2}_{i,m}$ and $\mathcal{A}^{h/2}_{m,j}$ consists of *all* the $h/2$-weighted paths of $\mathcal{A}$ going from state i to some state m, and *all* the $h/2$-weighted paths of $\mathcal{A}$ going from that state m to state j respectively. Since m ranges over all the possible states, from the above equation we get that $\mathcal{A}^{h}_{i,j}$ consists of *all* the possible h-weighted paths of $\mathcal{A}$ going from state i to state j. Hence, $\mathcal{A}^{h}$ will consists of all the h-weighted paths of $\mathcal{A}$ going from an initial to a final state. □

Notably, writing $\mathcal{A}^{h}_{i,j} = \bigcup_{m\in\{1,\ldots,n\}} \mathcal{A}^{h/2}_{i,m} \bullet \mathcal{A}^{h/2}_{m,j}$ (supposing h is even) instead of naively writing equivalently $\mathcal{A}^{h}_{i,j} = \bigcup_{m\in\{1,\ldots,n\}} \mathcal{A}^{h-1}_{i,m} \bullet \mathcal{A}^{1}_{m,j}$, makes us very efficient with respect to h (and in turn with respect to k) for computing $\mathcal{A}^{h}_{i,j}$ (and in turn $\mathcal{A}^{k}_{i,j}$). In order to see that, suppose for simplicity that h is a power of 2. Now, from our equation $\mathcal{A}^{h}_{i,j} = \bigcup_{m\in\{1,\ldots,n\}} \mathcal{A}^{h/2}_{i,m} \bullet \mathcal{A}^{h/2}_{m,j}$ we have that $\mathcal{A}^{2}_{i,j}$ will be a union of n automata of size $2p$ (where p is a polynomial on n), $\mathcal{A}^{4}_{i,j}$ will be a union of n automata of size $4np$, $\mathcal{A}^{8}_{i,j}$ will be a union of n automata of size $8n^2p$, and so on. Hence, by using our recurrence equation we will get a resulting automaton $\mathcal{A}^{h}_{i,j}$, which is a union of n automata of length $hn^{log_2 h-1}p$, i.e. the size of $\mathcal{A}^{h}_{i,j}$ will be $hn^{log_2 h}p$. In other words, had we used the equivalent equation $\mathcal{A}^{h}_{i,j} = \bigcup_{m\in\{1,\ldots,n\}} \mathcal{A}^{h-1}_{i,m} \bullet \mathcal{A}^{1}_{m,j}$, the automata $\mathcal{A}^{h}_{i,j}$ would be a union of n automata of size pn^{h-1}, i.e. the total size would be pn^{h}.

We are now ready to show the following theorem.

Theorem 5. *The k-limitedness problem is in PSPACE with respect to the size of the automaton. Furthermore, the decision can be made in space sub-exponential with respect to k.*

Proof. Recall that to decide whether $d(\mathcal{A}) \leq k$, amounts to testing the language equality $L(\mathbf{k}(\mathcal{A})) = L(\mathcal{A})$. Now, from the above discussion it is clear that the size of $\mathbf{k}(\mathcal{A})$ is $\mathcal{O}(k^2 n^{log_2 k})$. So, we can test the language equivalence $L(\mathbf{k}(\mathcal{A})) = L(\mathcal{A})$ in polynomial space on the size of $\mathcal{A}$ (see [HRS76]), and in sub-exponential space on k. □

Based on the above Theorem and on Theorem 3, we can state the following corollary.

Corollary 1. *The problem of deciding whether $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ is in PSPACE with respect to the combined size of Q_1, Q_2, and $\mathcal{T}$. Furthermore, the decision can be done in space sub-exponential with respect to k.*

We turn now on the lower bound for deciding the k-containment.

Theorem 6. *The problem of deciding whether $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ is PSPACE-hard, even if we know that $Q_1 \subseteq Q_2$.*

Proof. First recall from Theorem 2 that $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ is equivalent to $d_{\mathcal{T}}(Q_1, Q_2) \leq k$. We will now reduce the NFA universality problem to the $d_{\mathcal{T}}(Q_2, Q_1) \leq k$

problem. The universality problem says: given an NFA $\mathcal{A}$, is $\Delta^* \subseteq L(\mathcal{A})$? The universality problem is PSPACE complete [HRS76].

For an an arbitrary NFA $\mathcal{A}$ on Δ we take $Q_1 = L(\mathcal{A})$. We choose $Q_2 = \Delta^*$ and we take as a distortion transducer $\mathcal{T}$ the one that corresponds to the free applications of the three edit operations insertion, deletion, and substitution. The transducer will consist of a single state which will be both initial and final and loop transitions to this single state. Formally, this transducer is $\mathcal{T} = (\{p\}, \Delta, \tau, \{p\}, \{p\})$, where the transition relation is

$$\begin{aligned}\tau = &\{(p, R, R, 0, p) : R \in \Delta\} \cup \\ &\{(p, \epsilon, R, 1, p) : R \in \Delta\} \cup \\ &\{(p, R, \epsilon, 1, p) : R \in \Delta\} \cup \\ &\{(p, R, S, 1, p) : \{R, S\} \subset \Delta \text{ and } S \neq R\}.\end{aligned}$$

Intuitively, the above says: For each symbol R in Δ we will have a transition R/R leaving the symbol unchanged at no cost, or, at cost 1, through the transitions ϵ/R and R/ϵ we can insert and delete respectively a symbol or finally we can substitute at cost 1 a symbol by another through R/S transitions. Clearly, through the edit distance transducer any word can be transformed (or distorted) to any other word. From this fact, we have that $\Delta^* \subseteq \mathcal{T}(L(\mathcal{A}))$. However, a word can be transformed to another different word only through the application of non-zero cost edit operations. Hence, $d_{\mathcal{T}}(Q_1, Q_2) \leq 0$ if and only if $\Delta^* \subseteq L(\mathcal{A})$. □

Finally, Corollary 1 and Theorem 6 imply

Corollary 2. *To problem of deciding whether $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$ is PSPACE complete with respect to the combined size of Q_1, Q_2, and $\mathcal{T}$.*

6 Deciding Reliable Containment

As we already mentioned in Section 4, the problem of limitedness for distance automata (ϵ-free weighted automata) has been thoroughly investigated by [Has82,Has90,Has00,Leu91,Sim94]. However, in all previous works, the requirement of ϵ-freeness is very essential in devising algorithms for deciding the limitedness of automaton distance.

In this section, we show how to efficiently transform the transducer $\mathcal{C}$ into one with ϵ-free output transitions, is such a way that the essential features of $\mathcal{C}$ are preserved. As a consequence, the automaton $\mathcal{A}$, that we obtain from this output ϵ-free transducer, will be ϵ-free as well.

From the transducer $\mathcal{C}$ we will construct another "distance equivalent" transducer $\mathcal{D}$. We shall use $\epsilon\text{-}closure_{\mathcal{C}}(p)$, similarly to [HU79], to denote the set of all vertices q such that there is path π, from p to q in $\mathcal{C}$, with $out(\pi) = \epsilon$.

Obviously, we will keep all the transitions with non-ϵ output of $\mathcal{C}$ in the transducer $\mathcal{D}$, that we are constructing.

Now, we will insert a transition with R-output ($R \neq \epsilon$) in $\mathcal{D}$ from a state p to a state q whenever there is in $\mathcal{C}$ a path π, with $out(\pi) = \epsilon$, from p to an

intermediate state r and there is a transition with R-output, from that state r to the state q. Formally, if $\mathcal{C} = (P, \Delta, \tau, P_0, F)$, then $\mathcal{D} = (P, \Delta, \rho, P_0, G)$, where

$$G = F \cup \{p : p \in P_0 \text{ and } \epsilon\text{-}closure_{\mathcal{C}}(p) \cap F \neq \emptyset\}$$

and

$$\begin{aligned}\rho = &\{(p, R, S, k, q) : (p, R, S, k, q) \in \tau \text{ and } R \neq \epsilon\} \cup \\ &\{(p, w, S, \ell, q) : S \neq \epsilon, \exists r \in \epsilon\text{-}closure_{\mathcal{C}}(p), \text{ such that } (r, R, S, m, q) \in \tau\},\end{aligned}$$

where w will be a word, such that $w = in(\pi)$, where π is the[6] cheapest path from p to r in $\mathcal{C}$, such that $out(\pi) = \epsilon$. Also, the weight ℓ will be the weight of π (going from p to r) plus the weight of the[7] cheapest transition with R-output, from state r to state q in $\mathcal{C}$.

It is easy to verify about the above constructed transducer $\mathcal{D}$ that

Lemma 2. *rel*$(\mathcal{D}) \subseteq$ *rel*$(\mathcal{C})$, *dom*$(\mathcal{D}) \subseteq$ *dom*$(\mathcal{C})$, *and* *ran*$(\mathcal{D}) =$ *ran*$(\mathcal{C})$.

Then, we show that the distance features of $\mathcal{C}$ are preserved in $\mathcal{D}$.

Lemma 3. *Let* $w \in$ *ran*$(\mathcal{C}) =$ *ran*$(\mathcal{D})$. *Then* $d_{\mathcal{C}}(dom(\mathcal{C}), w) = d_{\mathcal{D}}(dom(\mathcal{D}), w)$

Proof. We have that

$$d_{\mathcal{C}}(dom(\mathcal{C}), w) = inf\{d_{\mathcal{C}}(u, w) : u \in dom(\mathcal{C})\}.$$

Let $u_0 \in dom(\mathcal{C})$ such that $d_{\mathcal{C}}(u_0, w) = d_{\mathcal{C}}(dom(\mathcal{C}), w)$, and consider the corresponding cheapest path π, labeled u_0/w, in $\mathcal{C}$. For simplicity suppose that u_0 is unique. The case when u_0 is not unique can be handled with some additional straightforward omitted technicalities. If none of the edges of π is labeled with ϵ as output, then by the construction we know that the path π is also in $\mathcal{D}$ and the proof is finished.

Now, let's suppose that some part of the path π is

$$p_1 \overset{R_1/\epsilon,\ h_1}{\longrightarrow} p_2, \ldots, p_{k-1} \overset{R_{k-1}/\epsilon,\ h_{k-1}}{\longrightarrow} p_k, p_k \overset{R_k/S,\ h_k}{\longrightarrow} p_{k+1},$$

where $S \neq \epsilon$.

We claim that in $\mathcal{D}$ we will have the transition $(p_1, R_1 \ldots R_k, S, h, p_{k+1})$ with weight $h = h_1 + \cdots + h_k$.

The only way that this could not be true is, if in $\mathcal{C}$ there is a path, comprised of ϵ-output transitions, which is cheaper than using π, of getting from p_1 to p_k. Suppose this cheap path is labeled $T_1/\epsilon, \ldots, T_{m-1}/\epsilon$. There could also be a transition from p_k to p_{k+1}, labeled with T_m/S, that is cheaper than the last π-transition. If such a cheaper path were to exist, and if we think of u_0 as being of the form $xR_1 \ldots R_{k-1}R_k y$, then for the word $u_1 = xT_1 \ldots T_{m-1}T_m y$, we would have

[6] The cheapest path can be non-unique.

[7] The cheapest transition can also be non-unique.

$d_{\mathcal{C}}(u_1, w) < d_{\mathcal{C}}(u_0, w)$, which is a contradiction. Hence $(p_1, R_1 \dots R_k, S, p_{k+1}, h)$ belongs to the transition relation of $\mathcal{D}$. Now, we can decompose the path π into disjoint (with respect to edges) subpaths such as the above, and subpaths that do not have any edge labeled by ϵ on the output. Recall that the subpaths that do not have any edge labeled by ϵ on the output are kept in $\mathcal{D}$, and so finally, we have that there exists a path σ in $\mathcal{D}$ labeled u_0/w and with the same weight as π. From this, we conclude that $d_{\mathcal{C}}(dom(\mathcal{C}), w) \geq d_{\mathcal{D}}(dom(\mathcal{D}), w)$.

To prove $d_{\mathcal{C}}(dom(\mathcal{C}), w) \leq d_{\mathcal{D}}(dom(\mathcal{D}), w)$ we reason in the following way. From the Lemma 2, we have that $rel(\mathcal{D}) \subseteq rel(\mathcal{C})$. So, if $(u, w) \in rel(\mathcal{D})$ then $(u, w) \in rel(\mathcal{C})$ and by the construction of $\mathcal{D}$ we know that for the cheapest path in $\mathcal{D}$ labeled with u/w there is a corresponding path in $\mathcal{D}$ labeled with u/w and having the same weight. Based on this observation the above inequality follows. □

If we now eliminate the input from the transitions in $\mathcal{D} = (P, \Delta, \rho, P_0, G)$, we obtain an ϵ-free distance automaton $\mathcal{A} = (P, \Delta, \tau_{\mathcal{A}}, P_0, G)$, where

$$\tau_{\mathcal{A}} = \{(p, R, \ell, q) : (p, w, R, k, q) \in \rho \text{ for some } w\},$$

and the weight ℓ is given by the weight of the (possibly non-unique) corresponding cheapest transition in the transducer $\mathcal{D}$, i.e.

$$\ell = inf\{k : (p, w, R, k, q) \in \rho \text{ for some } w\}.$$

Now, we can state the following theorem.

Theorem 7. *Let Q_1 and Q_2 be queries and $\mathcal{T}$ a distortion transducer. Compute the output ϵ-free Cartesian product transducer $\mathcal{D}$ and consider the distance automaton $\mathcal{A}$ constructed from $\mathcal{A}$. Then, $d(\mathcal{A}) = d_{\mathcal{T}}(Q_1, Q_2)$.*

Proof. For $p \in P_0$ and $q \in G$, let w be a word in Q_2 such that there exists a path π between the states p and q in $\mathcal{D}$ spelling w as output, i.e. $w = out(\pi)$. By the construction of $\mathcal{D}$ and $\mathcal{A}$, we have that $d_{\mathcal{A}}(p, w, q) = d_{\mathcal{D}}(dom(\mathcal{D}), w)$. Now, by using Lemma 3 and Lemma 1 we derive

$$\begin{aligned} d_{\mathcal{A}}(p, w, q) &= d_{\mathcal{D}}(dom(\mathcal{D}), w) \\ &= d_{\mathcal{C}}(dom(\mathcal{C}), w) \\ &= d_{\mathcal{T}}(Q_1, w). \end{aligned}$$

Since w was an arbitrary word in Q_2, we finally get that $d(\mathcal{A}) = d_{\mathcal{T}}(Q_1, Q_2)$. □

Hence, we are able now to use Leung's algorithm [Leu91], which is computationally the best known algorithm for solving the limitedness problem (in single exponential time), but for which the ϵ-freeness of the automata is essential.

Additionally, we show the following complexity bound for the reliable containment, which says that the exponential time algorithm of Leung is almost the best one could do for deciding the problem of reliable containment.

Theorem 8. *The reliable query containment problem is PSPACE-hard.*

Proof. We will give a reverse reduction from the limitedness problem, which is known to be PSPACE-hard [Leu91].

Let $\mathcal{A} = (P, \Delta, \tau, P_0, F)$ be a distance automaton. We take another alphabet Γ of the same size as, but disjoint from Δ. Let φ be a one-to-one mapping from Δ onto Γ. Now, from the automaton $\mathcal{A}$ we construct a distortion transducer $\mathcal{T} = (P \cup \{p_{id}\}, \Delta \cup \Gamma, \rho, P_0 \cup \{p_{id}\}, F)$, where

$$\rho = \{(p, \varphi(R), R, k, q) : (p, R, k, q) \in \tau\} \cup \{(p_{id}, R, R, 0, p_{id}) : R \in \Delta \cup \Gamma\}.$$

Note that by adding the new state p_{id} in the above transducer, we do have the property to "leave everything" unchanged that is required from a distortion transducer, in order include the original query words, undistorted, in the result of transduction.

Let $\mathcal{U}$ be the transducer that get if we drop in $\mathcal{T}$ the state p_{id}, and let $\mathcal{V}$ be the identity transducer that get if we drop all the other states and keep p_{id}. Clearly, the transducer $\mathcal{T}$ can be seen as the union of $\mathcal{U}$ and $\mathcal{V}$.

Now, let $Q_1 = \mathit{dom}(\mathcal{U})$ and $Q_2 = \mathit{ran}(\mathcal{U}) = L(\mathcal{A})$. Since, $\Gamma \cap \Delta = \emptyset$ and $Q_1 \subseteq \Gamma^*$, while $Q_2 \subseteq \Delta^*$, we have that $Q_1 \cap Q_2 = \emptyset$. So, we cannot get any word of Q_2 from Q_1 through the 0-weighted identity transducer $\mathcal{V}$, i.e. Q_1 can be distorted only through $\mathcal{U}$ (which closely corresponds to $\mathcal{A}$) to Q_2.

Finally, from all the above we have that $d_{\mathcal{T}}(Q_1, Q_2) = d_{\mathcal{U}}(Q_1, Q_2) = d(\mathcal{A})$. Hence, $d(\mathcal{A}) \leq k$ if $d_{\mathcal{T}}(Q_1, Q_2) \leq k$, which by Theorem 2 is equivalent with $Q_1 \sqsubseteq_{\mathcal{T},k} Q_2$. □

References

[ABS99] Abiteboul S., P. Buneman and D. Suciu. *Data on the Web : From Relations to Semistructured Data and Xml.* Morgan Kaufmann Pulishers. San Francisco, Ca., 1999.

[AHU74] Aho A., J. E. Hopcroft and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley. Reading Ma., 1974.

[C+99] Calvanese D., G. Giacomo, M. Lenzerini and M. Y. Vardi. Rewriting of Regular Expressions and Regular Path Queries. *Proc. PODS '99*, pp. 194–204.

[C+00] Calvanese D., G. Giacomo, M. Lenzerini and M. Y. Vardi. View-Based Query Processing and Constraint Satisfaction. *Proc. LICS '00*, pp. 361–371

[GT00] Grahne G., and A. Thomo. An Optimization Technique for Answering Regular Path Queries *Proc. WebDB '00*, pp. 99–104.

[GT01] Grahne G., and A. Thomo. Algebraic rewritings for optimizing regular path queries. *Proc. ICDT '01*, pp. 303–315

[Has82] Hashiguchi K. Limitedness Theorem on Finite Automata with Distance Functions. *J. Comp. Syst. Sci.* 24 (2) : 233–244, 1982

[Has90] Hashiguchi K. Improved Limitedness Theorems on Finite Automata with Distance Functions. *Theoretical Computer Science* 72 (1) : 27–38, 1990

[Has00] Hashiguchi K. New upper bounds to the limitedness of distance automata. *Theoretical Computer Science* 233 (1-2) : 19–32, 2000

[HU79] Hopcroft J. E., and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley. Reading Ma., 1979.

[HRS76] Hunt H. B. III, D. J. Rosenkrantz, and T. G. Szymanski, On the Equivalence, Containment, and Covering Problems for the Regular and Context-Free Languages. *J. Comp. Syst. Sci.* 12 (2) : 222–268, 1976

[Kru83] Kruskal J. An Overview of Sequence Comparison. In: *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* D. Sankoff and J. Kruskal (Eds.). Addison-Wesley. Reading Ma., 1983. pp. 1–44.

[JMM95] Jagadish H. V., A. O. Mendelzon, and T. Milo. Similarity-Based Queries. *Proc. PODS '95*, pp. 36–45.

[Leu91] Leung H. Limitedness Theorem on Finite Automata with Distance Functions: An Algebraic Proof. *Theoretical Computer Science* 81 (1) : 137–145, 1991

[MW95] Mendelzon A. O., and P. T. Wood, Finding Regular Simple Paths in Graph Databases. *SIAM J. Comp.* 24 (6) : 1235–1258, 1995.

[MMM97] Mendelzon A. O. G. A. Mihaila and T. Milo. Querying the World Wide Web. *Int. J. Dig. Lib.* 1 (1) : 57–67, 1997

[Pin98] Pin. J. E. Tropical Semirings, *in Idempotency*, J. Gunawardena (ed.) Cambridge University Press, pp. 50–69, 1998

[Sim94] Simon I. On Semigroups of Matrices over the Tropical Semiring. *Informatique Theorique et Applications* 28 (3-4) : 277–294, 1994

[WF74] Wagner R. A., and M. J. Fischer. The String-to-String Correction Problem. *J. ACM* 21 (1) : 168–173, 1974