Path Queries under Distortions: Answering and Containment

Gosta Grahne – Concordia University
Alex Thomo – Suffolk University

Foundations of Information and Knowledge Systems (FoIKS '04)
**Postulate 1**

The world is a database, and a database is a graph

**Fact 1**

Regular path queries are at the core of querying graph databases

**Fact 2**

Query containment is instrumental in query optimization and information integration

**Postulate 2**

Query optimization and information integration is the future
Viewing Data as Graphs

- Relational data
  - Tuples are the **nodes**
  - Foreign keys are the **edges**

- Object-oriented data

- Linked **Web** pages

- **XML**

- **AI**: **Semantic Networks**
Regular Path Queries and Distortions

Q: _* . Foto Afrati . Classes . Automata

- Will fetch the node of the automata course of Foto Afrati.
- However, suppose the user gives:

Q: _* . Foto Afrati . Automata

- For this the answer is empty!
- Well, we could distort the query by applying an edit operation – an insertion of ‘Classes’ in this case.

- However, Foto Afrati could also have some automata papers.
- But, “we” (DBA) know that Foto Afrati is a database person, so she probably doesn’t have many automata papers.
- On the other hand, there at NTU, it’s Foto who always teaches Automata.

To reflect these facts about the world, the DBA could write:

Graph Database $DB$

- Set of objects/nodes $D$, edges labeled with symbols from a database alphabet $\Delta$

![Graph DB Diagram]

- Query $Q$: regular language over $\Delta$
  For example $Q = ST + T + RR$

- $\text{ans}(Q, DB) = \{(x, y) \in D \times D : \text{there is a path from } x \text{ to } y \text{ in } DB \text{ labeled by a word in } Q \}$
Construct an automaton $A_Q$ with $p_0$ initial state.

Compute the set $\text{Reach}_a$ as follows.

1. Initialize $\text{Reach}_a$ to $\{(a, p_0)\}$.
2. Repeat step 3 until $\text{Reach}_a$ no longer changes.
3. Choose a pair $(b, p) \in \text{Reach}_a$.
   
   If there is a transition $(p, R, p')$ in $A_Q$, and there is an edge $(b, R, b')$ in $DB$, then add the pair $(b', s')$ to $\text{Reach}_a$.

Finally, $\text{ans}(Q, a, DB) = \{(a, b) : (b, s) \in \text{Reach}_a, \text{ and } s \text{ is a final state in } A_Q \}$.
**Query** $Q = \text{RTT}$

- $\text{ans}_T(Q, DB) = \{(a,d,2), (c,b,2)\}$

- $d_T(u,w) = \inf\{k : u \text{ goes to } w \text{ through } T \text{ by } k \text{ distortions}\}$

- $\text{ans}_T(Q, DB) = \{(a,b,k) : k = \inf\{d_T(u,w) : u \in Q, a \rightarrow_{w} b \in DB\}\}$
Although the full cartesian product has $4 \times 3 \times 4 = 48$ states, we needed only 3 states starting from ‘a’.
A Sketch…
Construct an automaton $A_Q$ with $p_0$ initial state
Compute the set $\text{Reach}_a$ as follows.

1. Initialize $\text{Reach}_a$ to $\{(p_0, s_0, a, 0, \text{false})\}$.
   /* The boolean flag is for the membership in the set of nodes for which we know the
   exact cost from source */

2. Repeat 3 until $\text{Reach}_a$ no longer changes.

3. Choose a $(p, s, b, k, \text{false}) \in \text{Reach}_a$, where $k$ is min
   If [there is a transition $(p, R, p')$ in $A_Q$] and
   [a transition $(s, R/S, s', n)$ in $T$] and
   [there is an edge $(b, S, b')$ in $DB$]
   Then
   
   add $(p', s', b', k+n, \text{false})$ to $\text{Reach}_a$ if there is no $(p', s', b', _, _) \in \text{Reach}_a$
   relax the weight of any successor of $(p, s, b, k, \text{false})$ in $\text{Reach}_a$.
   update $(p, s, b, k, \text{false})$ to $(p, s, b, k, \text{true})$.

Finally, $\text{ans}_T(Q, a, DB) = \{(a, b, k) : (p, s, b, k, \text{true}) \in \text{Reach}_a, \text{ and } p \text{ is a final state in } A_Q,$
and $s$ is a final state in $T\}$
• In other words, the priority queue of Dijkstra’s algorithm is brought on demand (lazily) in memory.

• **Complexity:** If we keep the set $\text{Reach}_a$ in main memory we avoid accessing objects in secondary memory more than once.

• Data complexity (i.e. number of I/O’s) is all we care in databases! ...And it is *linear!*
Redefining Query Containment

• Classical case: $Q_1 \subseteq Q_2$ iff $\text{ans}(Q_1, \text{DB}) \subseteq \text{ans}(Q_2, \text{DB})$ on any DB.
  • We can provide the answers of $Q_1$ as answers for $Q_2$ and be certain that they will be valid for $Q_2$ on any DB.

• Suppose now that $Q_1 \not\subsetneq Q_2$. However, by using the distortion transducer some kind of containment might still hold.
An Example

- $Q_1 = \{R, S\}$, $Q_2 = \{U, V\}$, $T = \{(U/R, 1), (V/S, 3)\}$

- Suppose $(a, b, 0) \in \text{ans}_T(Q, DB)$ --- what could be the DB?

  $$
  \begin{align*}
  DB_1 & \quad a \xrightarrow{R, T} b \\
  DB_2 & \quad a \xrightarrow{R} b \\
  DB_3 & \quad a \xrightarrow{S} b
  \end{align*}
  $$

  $(a, b, 0) \in \text{ans}_T(Q_2, DB_3)$

  $(a, b, 1) \in \text{ans}_T(Q_2, DB_1)$

  $(a, b, 3) \in \text{ans}_T(Q_2, DB_2)$

- $Q_1 \not\subseteq Q_2$.

  However, for any DB, if $(a, b, 0) \in \text{ans}_T(Q_2, DB)$
  then $(a, b, m) \in \text{ans}_T(Q_2, DB)$, where $m \leq 0 + 3$. 
Another Example

- $Q_1 = \{\text{RRR}\}$, $Q_2 = \{\text{RST}\}$ T is the edit transducer

- Suppose $(a,b,1) \in \text{ans}_T(Q, \text{DB})$ --- what could be the DB?

- $Q_1 \not\subseteq Q_2$.
  However, for any DB, if $(a,b,1) \in \text{ans}_T(Q_2, \text{DB})$ then $(a,b,m) \in \text{ans}_T(Q_2, \text{DB})$, where $m \leq 1 + 2$. 

\[ DB_1 \quad a \xrightarrow{R} \xrightarrow{R,S} \xrightarrow{T} \quad b \]
\[ (a,b,0) \in \text{ans}_T(Q_2, DB_1) \]

\[ DB_2 \quad a \xrightarrow{U} \xrightarrow{R,S} \xrightarrow{R,T} \quad b \]
\[ (a,b,1) \in \text{ans}_T(Q_2, DB_2) \]

\[ DB_3 \quad a \xrightarrow{U} \xrightarrow{R} \xrightarrow{R,T} \quad b \]
\[ (a,b,2) \in \text{ans}_T(Q_2, DB_3) \]

\[ DB_4 \quad a \xrightarrow{U} \xrightarrow{R} \xrightarrow{R} \quad b \]
\[ (a,b,3) \in \text{ans}_T(Q_2, DB_4) \]
Query Containment (Continued)

- $Q_1 \subseteq_{(T,k)} Q_2$

  iff

  $$(a,b,n) \in \text{ans}_T(Q_1,\text{DB}) \Rightarrow (a,b,m) \in \text{ans}_T(Q_2,\text{DB}) \text{ and } m \leq n + k \text{ on any DB.}$$

- $Q_1 \not\subseteq Q_2$
- $Q_1 \not\subseteq_{(T,1)} Q_2$
  ...
- $Q_1 \subseteq_{(T,k)} Q_2$
- $Q_1 \subseteq_{(T,k+1)} Q_2$
  ...
- $Q_1 \subseteq T(Q_2)$

- What’s the $k$?
We devise a method for constructing: $Q^{(T,k)}$ : the language of all $Q$-words distorted by $T$ with cost at most $k$.

Clearly $Q^{(T,k-1)} \subseteq Q^{(T,k)}$.

In this way we control how bigger we need to make $Q_2$.

Suppose, that $k$ is the smallest number, such that $Q_1 \subseteq Q_2^{(T,k)}$.

If $d_T$ satisfies the triangle inequality property, we show that:

$$Q_1 \subseteq^{(T,k)} Q_2 \text{ iff } Q_1 \subseteq Q_2^{(T,k)}.$$
About the Triangle Property of $T$

- There are transducers, whose word distance doesn’t satisfy the triangle property. E.g. $\{(R,1,S), (S,2,T), (R,5,T)\}$.

\[ d_T(R,S)=1, \ d_T(S,T)=2, \text{ but } d_T(R,T)=5>3 \]

- Nevertheless, there are large classes which, posses the triangle propety.

- The pure edit distance transducers. E.g. $\{ (R,1,S), (S,1,T), (R,1,T), (S,1,R), (R,1,\varepsilon), (\varepsilon,1,R) \ldots \}$.

- Transducers whose input and output of distortions do not have intersection. Such transducers are idempotent wrt composition.

\[
(T \cup T_{id})^\circ (T \cup T_{id}) = (T \circ T) \cup (T \circ T_{id}) \cup (T_{id} \circ T) \cup (T_{id} \circ T_{id}) = T \cup T_{id}
\]

- In general, an idempotent transducer has the triangle property.
  - $uTv \land vTw \Rightarrow uT^\circ Tw \Rightarrow uTw$
  - Hence, $d_T(u,w) = d_{T^\circ T}(u,w) \leq d_T(u,v)+d_T(v,w)$. 
The class of “\(\text{range}(T) \cap \text{dom}(T) = \emptyset\)” transducers is indeed practical:

- Recall that it is the DBA who writes the reg. expr. for the distortion transducer.

- It is common sense that DBA has surely an idea about the DB.

- Hence, we can consider that all the words in \(\text{range}(T)\) match to DB paths.

- On the other hand, the words of the \(\text{dom}(T)\) can be considered not having a direct match on the database; otherwise why the system administrator would like them to be translated.
$Q_1 \subseteq^0_{(T,k)} Q_2$

- However, if we restrict ourselves in reasoning about those tuples in $Q_1$ with weight $0$, then we don’t need the triangle property for $T$.

- We obtain a relaxed definition for the k-containment:
  
  \[ Q_1 \subseteq^0_{(T,k)} Q_2 \iff \]
  
  \[ (a,b,0) \in \text{ans}_T(Q_1, DB) \Rightarrow (a,b,m) \in \text{ans}_T(Q_2, DB) \text{ and } m \leq k \text{ on any DB}. \]

- Clearly, $(a,b,0) \in \text{ans}_T(Q_1, DB)$ mainly correspond to the tuples of the pure answer of $Q_1$ on DB.

- We are able to prove that $Q_1 \subseteq^0_{(T,k)} Q_2$ iff $Q_1 \subseteq Q_2^{(T,k)}$.
  
  (Even when the triangle property doesn’t hold).
Computing $Q^{(T,k)}$ - I

- First we obtain a **weighted** transduction of $Q$ by $T$.

- Let $A_Q = (P_Q, \Delta, \tau_Q, p_{Q,0}, F_Q)$ be an $\varepsilon$-free NFA for $Q$.

- Let $T = (P_T, \Delta, \tau_T, p_{T,0}, F_T)$ in standard form.

- We construct the weighted transduction automaton of $Q$ by $T$ as $A = (P, \Delta, \tau, p_0, F)$, where $P = P_Q \times P_T$, $p_0 = p_{Q,0} \times p_{T,0}$, $F = F_Q \times F_T$.

- $\tau = \{ ((p,q), S, k, (p',q')) : (p,R,p') \in \tau_Q, (q,R,S,k,q') \in \tau_T \} \cup \{ ((p,q), S, k, (p,q')) : (q,\varepsilon,S,k,q') \in \tau_T \}$.

- Now, we should find all the paths in $A$, such that their weight is less than $k$. We denote it $k(A)$. 
Computing $Q^{(T,k)}$ - II

• Let $A^h$ be the sub-automaton consisting of all the paths with weight $h$.
  • $k(A) = A^0 \cup A^1 \cup ... \cup A^k$

• We suppose that all the weights in $A$ are 0 or 1.
  • If not, e.g. $(p,R,m,q)$ we replace by $(p,R,1,r_1), ..., (r_{m-1},R,1,q)$

• We number the states of $A$: 1,2,...,n

• $A_{ij}$ is $A$, but with initial state $i$ and final $j$.
• $0(A)$ keeping only the 0-weighted transitions in $A$.
• $1_{ij}(A)$ elementary two state (i and j) automata with the 1-weighted transitions from i to j.
Computing $Q(T, k) - III$

- $k(A) = A^0 \cup A^1 \cup ... \cup A^k$
- $A^0 = O(A)$, and for $1 \leq h \leq k$
- $A^h = \bigcup_{i \in S, j \in F} A^h_{ij}$

\[
A^h_{ij} = \begin{cases} 
\bigcup_{m \in \{1, ..., n\}} A^{h/2}_{im} \cdot A^{h/2}_{mj} & \text{for } h \text{ even} \\
\bigcup_{m \in \{1, ..., n\}} A^{(h-1)/2}_{im} \cdot A^{(h+1)/2}_{mj} & \text{for } h \text{ odd}
\end{cases}
\]

- $A^1_{ij} = \bigcup_{\{m,l\} \subseteq \{1, ..., n\}} O(A)_{im} \cdot 1_{ml}(A) \cdot O(A)_{lj}$

- $A^1_{ij}$ consists of $A$-paths starting from state $i$ and traversing any number of 0-weighted arcs up to some state $m$, then a 1-weighted arc going some state $i$, and after that, any number of 0-weighted arcs ending up in state $j$.
- $A^{h/2}_{im}$ all the $h/2$-weighted paths of $A$ going from state $i$ to some state $m$.
- $A^{h/2}_{mj}$ all the $h/2$-weighted paths of $A$ going from that “some” state $m$ to state $j$.
- Since $m$ ranges over all the possible states, $A^h_{ij}$ consists of all the possible $h$-weighted paths from state $i$ to state $j$. 
Computing $Q^{(T,k)}$ - IV

- E.g. Suppose that $A$ is:

- $0(A)$:

- $1_{12}(A)$:

- $A_{12}^1 = 0(A)_{12} \cdot 1_{22}(A) \cdot 0(A)_{22} \cup 0(A)_{11} \cdot 1_{12}(A) \cdot 0(A)_{22} = \{R\}$
  - $A_{11}^1 = \emptyset$, $A_{22}^1 = \{SR\}$, $A_{21}^1 = \emptyset$
  - $A^1 = A_{12}^1 = \{R\}$
  - $A_{12}^2 = A_{12}^1 \cdot A_{22}^1 \cup A_{11}^1 \cdot A_{12}^1 = \{R,S\} \cup \emptyset$
Computing $Q^{(T,k)} - V$

- From $A^h_{ij} = \bigcup_{m \in \{1, \ldots, n\}} A^{h/2}_{im} \cdot A^{h/2}_{mj}$ (for simplicity assume $h$ is power of 2)
  - $A^2_{ij}$ is a union of $n$ automata of size $2^p$ ($p$ is polynomial in $n$)
  - $A^4_{ij}$ is a union of $n$ automata of size $4np$
  - $A^8_{ij}$ is a union of $n$ automata of size $8n^2p$
  - ...
  - $A^h_{ij}$ is a union of $n$ automata of size $4n^{\log h \cdot p}$

- Hence, the size of $A^h_{ij}$ is $4n^{\log h \cdot p}$.

- Had we used the equivalent $A^h_{ij} = \bigcup_{m \in \{1, \ldots, n\}} A^{h-1}_{im} \cdot A^1_{mj}$ we would get $pn^h!$

- **Conclusion:** Computing $Q^{(T,k)}$ is polynomial in $n$ and sub-exponential in $k$. 
A broader perspective – semirings

- In the transducer, the weights were natural numbers and the specific operations were addition (+) along a path, and minimum (min) applied to path weights.

- This can be generalized to other weight sets, and to other operations.
- The weights, elements of a set K, can be multiplied along a path using an operation \( \otimes \), and then summarized using an operation \( \oplus \).

- Semirings: \((K, \oplus, \otimes, 0, 1)\)
  - \((K, \oplus, 0)\) commutative monoid with 0 as the identity element \( \oplus \).
  - \((K, \otimes, 1)\) monoid with 1 as the identity element for \( \otimes \).
  - \( \otimes \) distributes over \( \oplus \):
    - \((a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c), \ c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b)\)
  - 0 is anihilator for \( \otimes \): \(a \otimes 0 = 0\).
The on focus semiring

- Tropical Semiring: \((K, \oplus, \otimes, 0, 1)\), where \(K=\mathbb{N}\), \(\oplus=\min\), \(\otimes=+\), \(0=\infty\), \(1=0\)

- \((a \oplus b) \otimes c = \min(a, b) + c = \min(a+c, b+c) = (a \otimes c) \oplus (b \otimes c)\), hence \(\otimes\) distributes over \(\oplus\).

- Why does Dijkstra’s algorithm work?
  - It is based on the assumption that no shortest path needs to traverse a cycle!
  - This is true for the Tropical Semiring, because it is a **bounded** semiring. Boundedness is defined as:
    \[1 \oplus a = 1\] for each \(a\), (i.e. \(\min(0, a) = 0\)).
  - Hence, if we have a cycle with weight \(a\), we don’t gain anything traversing it: \(1 \oplus a \oplus a \otimes a + a \otimes a \otimes a + \ldots = 1\)

- In general, we can apply the Approximate Answering algorithm with any transducer whose weights are from a **bounded semiring**.
Other semirings

- Probabilistic: $([0,1], \max, \times, 0, 1)$
- Fuzzy: $([0,1], \max, \min, 0, 1)$

- Both of them are bounded.

- However, if we define the probabilistic semiring as: $(\mathbb{R}, +, \times, 0, 1)$, then we haven’t a bounded semiring.
  - Note: If $C^*$ is the weight of the shortest path, we produce as the answer from the Dijkstra algorithm the $\min(C^*, 1)$.

- In such cases, we can use the **Floyd-Warshall** algorithm, which doesn’t require boundedness.
Future work

• The Floyd-Warshall algorithm is impractical for sparse graphs, and modifying it for secondary memory is not known.

• Extending the algorithm for computing $Q^{(T,k)}$ in other semirings.
References

- Gösta Grahne, Alex Thomo. Approximate Reasoning in Semistructured Data. KRDB 2001