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

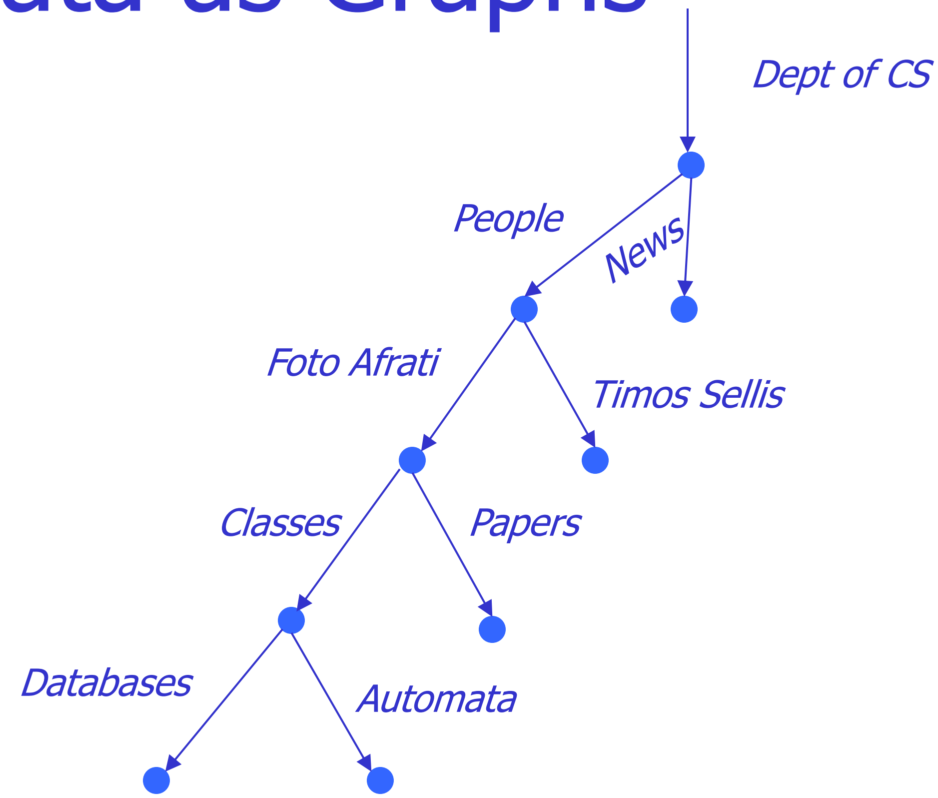
Query containment is instrumental in query optimization and information integration

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

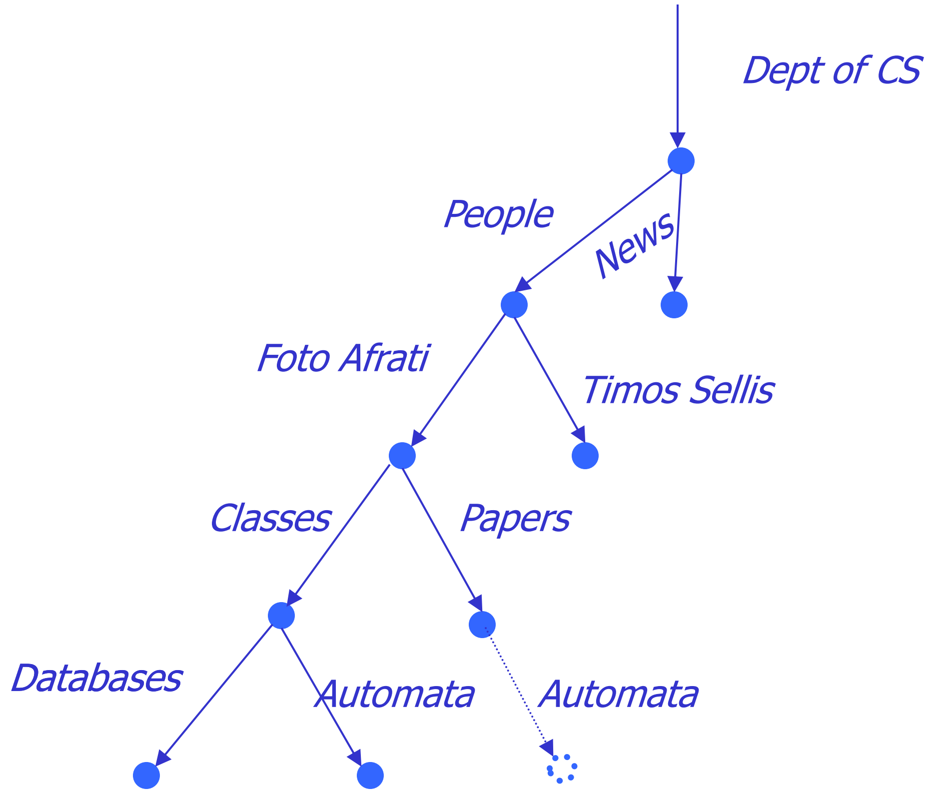
Q: _* . Foto Afrati . Classes . Automata

- 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.



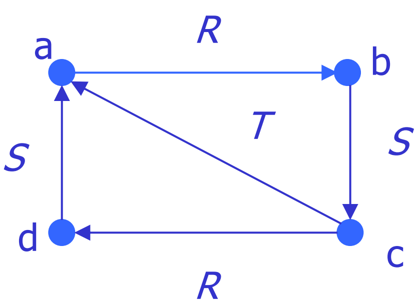
- To reflect these facts about the world, the DBA could write:

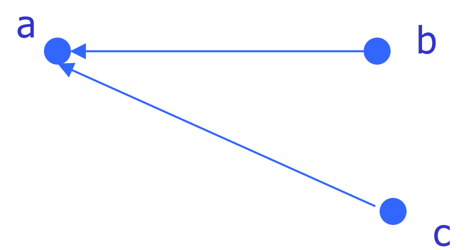
_* . (
    (Foto Afrati . Automata, 1,
    Foto Afrati . Classes. Automata)
    +
    (Foto Afrati . Automata, 5,
    Foto Afrati . Papers. Automata)
) . _*

Graph Database *DB*

- Set of objects/nodes *D*, edges labeled with symbols from a **database alphabet** Δ

DB





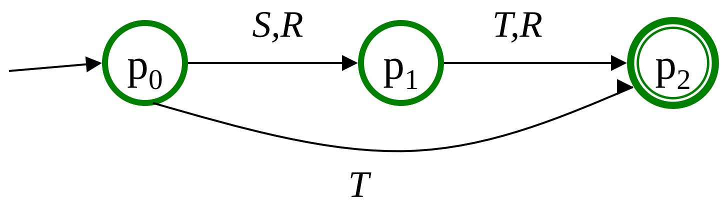
ans(Q,DB)

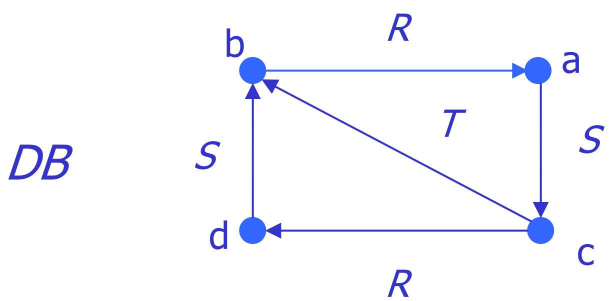
- Query *Q* : regular language over Δ

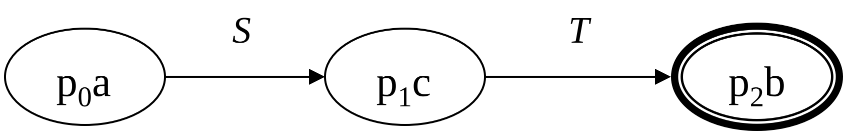
  For example Q = ST + T + RR

- ans(Q,DB) = {(x,y) in *D x D* : there is a path from x to y in DB labeled by a word in Q }

Computing the Answer







Construct an automaton $\mathbf{A_Q}$ with $\mathbf{p_0}$ initial state

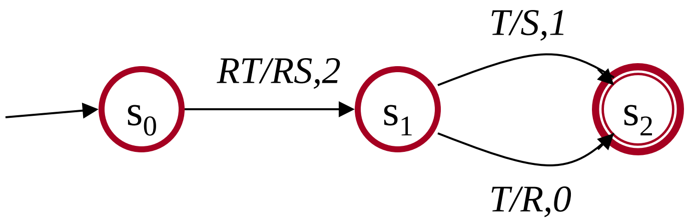
Compute the set $\mathbf{Reach_a}$ as follows.

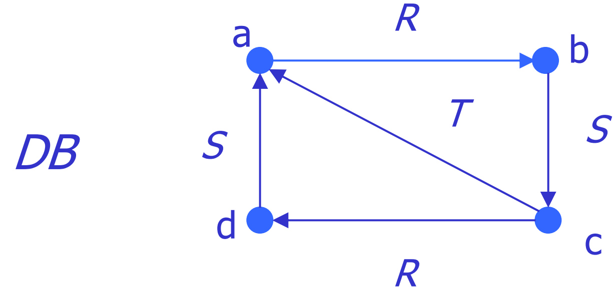
1. Initialize $\mathbf{Reach_a}$ to {(a, p_0)}.
2. Repeat **3** until $\mathbf{Reach_a}$ no longer changes.
3. Choose a pair (b,p)∈ $\mathbf{Reach_a}$.

   If there is a transition **(p,R,p')** in $\mathbf{A_Q}$, and there is an edge **(b,R,b')** in **DB**, then add the pair **(b',s')** to $\mathbf{Reach_a}$.

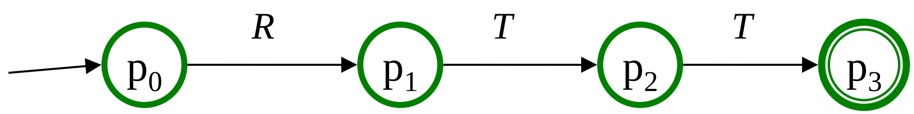
Finally, **ans(Q, a, DB)**={**(a, b)** : **(b,s)**∈ $\mathbf{Reach_a}$, and **s** is a final state in $\mathbf{A_Q}$}

Distortion Transducer *T*





- Query Q = RTT

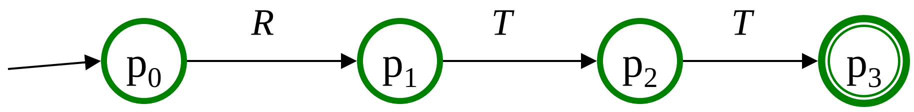


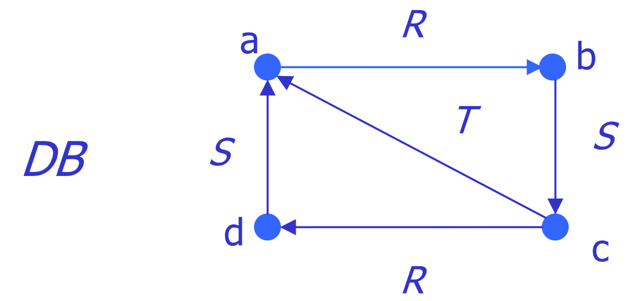
- $ans_T(Q,DB) = \{(a,d,2), (c,b,2)\}$

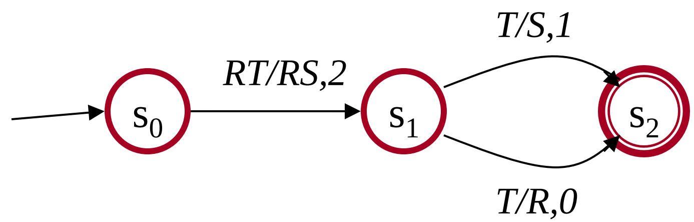
- $d_T(u,w) = \mathbf{inf}\{k : \text{u goes to w through } T \text{ by } k \text{ distortions}\}$

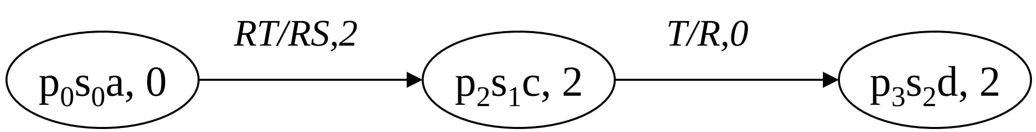
- $ans_T(Q,DB) = \{(a,b,k) : k = \mathbf{inf}\{d_T(u,w) : u \in Q,\ a \rightarrow_w b \in DB\}\}$

Lazy Dijkstra Algorithm on Cartesian Product









- Although the full cartesian product has **4*3*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 $Reach_a$ as follows.

1. Initialize $Reach_a$ to $\{(p_0, s_0, a, 0, false)\}$.

   /* 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 $Reach_a$ no longer changes.
3. Choose a **(p, s, b, k,false)**$\in$ $Reach_a$, 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, false)** to $Reach_a$ if there is no **(p', s', b', _, _)** in $Reach_a$

   relax the weight of any successor of **(p, s, b, k, false)** in $Reach_a$.

   update **(p, s, b, k, false)** to **(p, s, b, k, true)**.

Finally, $ans_T(Q, a, DB)=\{(a, b, k) : (p, s, b, k, true) \in Reach_a,$ and **p** 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 $\mathbf{Reach_a}$ 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** $ans(Q_1,DB) \subseteq ans(Q_2,DB)$ 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 $Q_1 \not\subset Q_2$. 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)** $\in$ **ans_T(Q, DB)** --- what could be the DB?

(a,b,0) $\in$ **$ans_T(Q_2, DB_3)$**

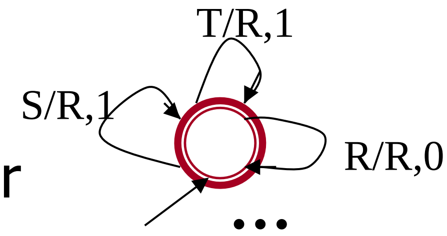
(a,b,1) $\in$ **$ans_T(Q_2, DB_1)$**

(a,b,3) $\in$ **$ans_T(Q_2, DB_2)$**

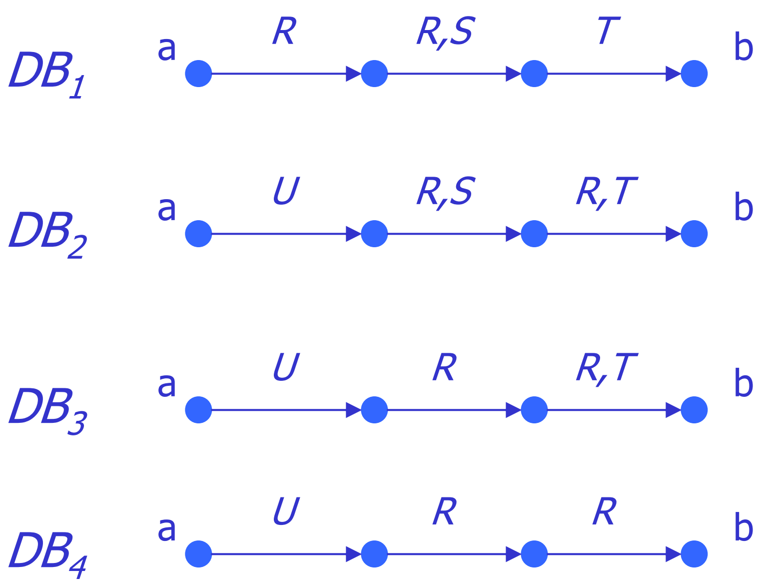
- $Q_1 \not\subset Q_2$.

  However, for any DB, if **(a,b,0)** $\in$**$ans_T(Q_2,DB)$**
  then **(a,b,m)** $\in$**$ans_T(Q_2,DB)$,** where m<=0+**3**.

Another Example



- Q_1 = {RRR}, Q_2 = {RST} T is the edit transducer
- Suppose **(a,b,1)** ∈ **$\mathbf{ans_T}$(Q, DB)** --- what could be the DB?



(a,b,0) ∈ $\mathbf{ans_T(Q_2, DB_1)}$

(a,b,1) ∈ $\mathbf{ans_T(Q_2, DB_2)}$

(a,b,2) ∈ $\mathbf{ans_T(Q_2, DB_3)}$

(a,b,3) ∈ $\mathbf{ans_T(Q_2, DB_4)}$

- $Q_1 \not\subset Q_2$.

  However, for any DB, if **(a,b,1) ∈$\mathbf{ans_T(Q_2,DB)}$**
  then **(a,b,m) ∈$\mathbf{ans_T(Q_2,DB)}$,** where m<=1+**2**.

Query Containment (Continued)

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

  iff

  $(a,b,n) \in ans_T(Q_1,DB) \Rightarrow (a,b,m) \in ans_T(Q_2,DB)$ and $m <= n + k$ on any DB.

$Q_1 \not\subset Q_2$
$Q_1 \not\subset_{(T,1)} Q_2$
…
$Q_1 \subseteq_{(T,k)} Q_2$
$Q_1 \subseteq_{(T,k+1)} Q_2$
…
$Q_1 \subseteq T(Q_2)$

- What's the **k**?

A tool for deciding k-containment

- We devise a method for constructing:
  $\mathbf{Q^{(T,k)}}$: the language of **all** Q-words distorted by T with cost **at most** k.
  Clearly $Q^{(T,k-1)} \subseteq Q^{(T,k)}$

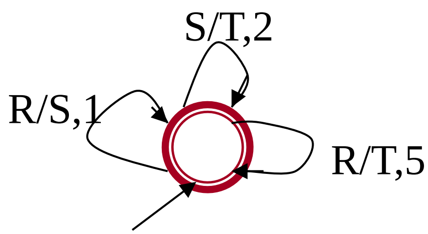
- In this way we **control** how bigger we need to make Q_2.

- 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:
  $\mathbf{Q_1 \subseteq_{(T,k)} Q_2 \text{ iff } Q_1 \subseteq Q_2^{(T,k)}}$.

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)} .



$d_T(R,S)=1$, $d_T(S,T)=2$, 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,ε), (ε,1,R)…}.

- Transducers whose input and output *of distortions* do not have intersection. Such tranducers 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.
  - **u**Tv Λ vT**w** ⇒ uT∘Tw ⇒ uTw
  - Hence, $d_T(u,w) = d_{T\circ T}(u,w) <= d_T(u,v)+d_T(v,w)$.

Triangle Property (Continued)

- The class of "***range(T)***$\cap$***dom(T)***=$\varnothing$" 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* ***range(T)*** *match to DB paths.*
  - *On the other hand, the words of the* ***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**

  $(a,b,0) \in ans_T(Q_1,DB) \Rightarrow (a,b,m) \in ans_T(Q_2,DB)$ *and*

  $m <= k$ on any DB.

- Clearly, $(a,b,0) \in ans_T(Q_1,DB)$ 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 ε-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.
    - $\mathbf{k}(A) = A^0 \cup A^1 \cup \ldots \cup A^k$

- 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: $1,2,\ldots,n$

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

Computing $Q^{(T,k)}$ - III

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

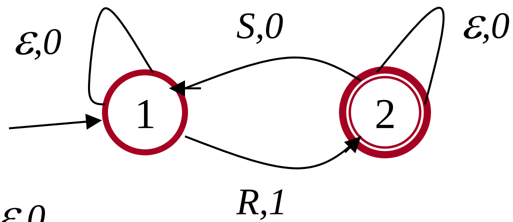
- $A^0 = \mathbf{0}(A)$, and for $1 <= h <= k$
- $A^h = \cup_{i \in S, j \in F} A^h_{ij}$
- $$A^h_{ij} = \begin{cases} \cup_{m \in \{1,\ldots,n\}} A^{h/2}_{im} \,.\, A^{h/2}_{mj} & \text{for } h \text{ even} \\ \cup_{m \in \{1,\ldots,n\}} A^{(h-1)/2}_{im} \,.\, A^{(h+1)/2}_{mj} & \text{for } h \text{ odd} \end{cases}$$

- $A^1_{ij} = \cup_{\{m,l\} \subset \{1,\ldots,n\}} \mathbf{0}(A)_{im} \,.\, \mathbf{1}_{ml}(A) \,.\, \mathbf{0}(A)_{lj}$

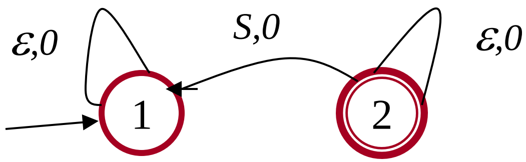
- A^1_{ij} consists of A-paths starting from state i and traversing any number of **0-weighted** arcs up to some state m, then a **1-weighted** arc going some state i, and after that, any number of **0-weighted** arcs ending up in state j.
- $A^{h/2}_{im}$ all the **h/2-weighted** paths of A going from 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

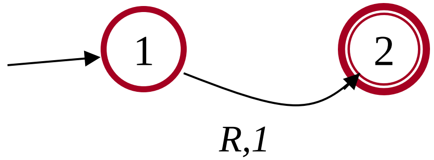
- E.g. Suppose that A is:



- 0(A) :



- 1_{12}(A):



- $A^1_{12} = 0(A)_{12} \,.\, 1_{22}(A) \,.\, 0(A)_{22} \cup 0(A)_{11} \,.\, 1_{12}(A) \,.\, 0(A)_{22} = \{R\}$
- $A^1_{11} = \varnothing$, $A^1_{22} = \{SR\}$, $A^1_{21} = \varnothing$
- $A^1 = A^1_{12} = \{R\}$

- $A^2_{12} = A^1_{12} \,.\, A^1_{22} \cup A^1_{11} \,.\, A^1_{12} = \{ R.SR\} \cup \varnothing$

Computing $Q^{(T,k)}$ - V

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

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

- Had we used the equivalent $A^h_{ij} = \cup_{m\in\{1,\ldots,n\}} A^{h-1}_{im} \,.\, A^1_{mj}$ 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: $(K, \oplus, \otimes, \underline{0}, \underline{1})$
  - $(K, \oplus, \underline{0})$ commutative monoid with $\underline{0}$ as the identity element $\oplus$.
  - $(K, \otimes, \underline{1})$ monoid with $\underline{1}$ as the identity element for $\otimes$.
  - $\otimes$ distributes over $\oplus$:
    - $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$, $c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b)$
  - $\underline{0}$ is anihilator for $\otimes$: $a \otimes \underline{0} = \underline{0}$.

The on focus semiring

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

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

- Why does Dijkstra's algorithms 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:

  $$\underline{\mathbf{1}} \oplus \mathbf{a} = \underline{\mathbf{1}} \text{ for each } a, \quad (\text{i.e. } \min(0, a) = 0).$$

- Hence, if we have a cycle with weight **a**, we don't gain anything traversing it: $\underline{\mathbf{1}} \oplus \mathbf{a} \oplus \mathbf{a}\otimes\mathbf{a} + \mathbf{a}\otimes\mathbf{a}\otimes\mathbf{a} + \ldots = \underline{\mathbf{1}}$

- *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, ×, 0, 1)
- Fuzzy: ([0,1], max, min, 0, 1)

- Both of them are bounded.

- However, if we define the probabilistic semiring as: (R, +, ×, 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. Query Answering and Containment for Regular Path Queries under Distortions. FoIKS 2004: 98-115
- Gösta Grahne, Alex Thomo. Approximate Reasoning in Semistructured Data. KRDB 2001