Graphs with Data

08/2021

DATA GRAPHS

- So far, we have focused on the "navigational" aspects of graph databases.
- But graph databases also handle data, which could then be compared while traversing a path.
- Example: Find authors who are linked by a coauthorship sequence in which everyone is from the same institution.

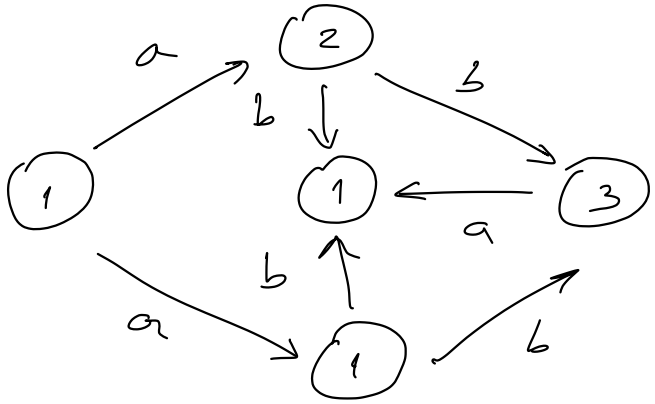
We use a single data model that allows to handle data at the level of nodes

---

A data graph over Σ is a tuple $G = (V, E, \alpha)$, where:

- (V, E) is a graph DB over Σ.
- $\alpha : V \longrightarrow IN$ assigns a data value in IN to each node in V.

Example :



Let $G = (V, E, \lambda)$ be a data graph.

If $\varphi = v_1 a_1 v_2 a_2 v_3 \dots v_{n-1} a_n v_n$

is a path in $G \Rightarrow$

$\lambda(\varphi) = \alpha(v_1) a_1 \alpha(v_2) \dots \alpha(v_{n-1}) a_n \alpha(v_n)$ is a data path.

- Query languages for data graphs aim at checking properties over its data paths.
- However, even some simple properties are intractable in this setting.

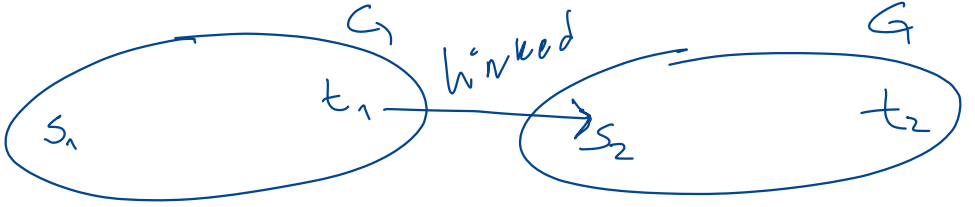
Proposition (Libkin, Vrgoc; 2012):

Given a data graph $G = (V, E, \lambda)$, checking if there is a data path from s to t in G all whose data values are different, is NP-complete.

Proof: We reduce from the NP-complete problem 2-disjoint-paths:

> Given graph G, nodes s_1, t_1, s_2, t_2, are there paths φ_1 from s_1 to t_1, and φ_2 from s_2 to t_2, s.t. φ_1 and φ_2 share no vertices?

The reduction is as follows:



$V = \{v_1, \ldots, v_n\}$

$\Rightarrow \alpha(v_i) = i.$

Observation: There are disjoint paths φ_1 from s_1 to t_1 and φ_2 from s_2 to t_2 $\iff$ there is a path from s_1 to t_2 that does not repeat data values.

Conclusion: If we want a query language for data graphs that is tractable in data complexity $\Rightarrow$ it cannot express this property.

REGULAR EXPRESSIONS WITH MEMORY

We use a formalism that exploits the properties of register automata:

- Data values can be stored in k registers $x_1, \ldots, x_k$
- At any point while traversing a path we can compare the current data value with any one stored in the registers.

A regular expression with memory (REM) over alphabet Σ is defined by the following grammar:

- Conditions on $\{x_1, \ldots, x_k\}$:

$$c := x_i^{=} \mid c \wedge c \mid \neg c$$

- REMs:

$$e := \emptyset \mid \varepsilon \mid a \mid e \cup e \mid e \cdot e \mid e^* \mid e[c] \mid \downarrow \bar{x} . e$$

($\bar{x}$ a tuple in $\{x_1, \ldots, x_k\}$)

Semantics of REMs: For defining the semantics of conditions, we need a data value $d \in \mathbb{N}$ and a tuple $\lambda \in (\mathbb{N} \cup \{*\})^k$.

Then: $(d, \lambda) \models x_i^{=} \iff \lambda(x_i) = d$.
(Boolean combinations are standard).

For the semantics of REMs we need to define an auxiliary relation $(e, \varphi_D, \lambda) \vdash \lambda'$

- e an REM
- φ_D a data path
- $\lambda, \lambda' \in (\mathbb{N} \cup \{*\})^k$

The definition is inductive:

1. $(\varepsilon, \varphi_D, \lambda) \vdash \lambda' \iff \varphi_D = d \in \mathbb{N}$ and $\lambda = \lambda'$
2. $(a, \varphi_D, \lambda) \vdash \lambda' \iff \varphi_D = d_1 a d_2$ and $\lambda = \lambda'$
3. $(e_1 \cup e_2, \varphi_D, \lambda) \vdash \lambda' \iff (e_1, \varphi_D, \lambda) \vdash \lambda'$ or $(e_2, \varphi_D, \lambda) \vdash \lambda'$
4. $(e_1 \cdot e_2, \varphi_D, \lambda) \vdash \lambda' \iff \varphi_D = \varphi_D^1 \cdot \varphi_D^2$ and $(e_1, \varphi_D^1, \lambda) \vdash \lambda''$; $(e_2, \varphi_D^2, \lambda'') \vdash \lambda'$ for some $\lambda'' \in (\mathbb{N} \cup \{*\})^k$.
5. $(e^*, \varphi_D, \lambda) \vdash \lambda' \iff \varphi_D = \varphi_D^1 \cdots \varphi_D^k$ and $(e, \varphi_D^i, \lambda_{i-1}) \vdash \lambda_i$, for some $\lambda_0, \ldots, \lambda_k \in (\mathbb{N} \cup \{*\})^k$ with $\lambda_0 = \lambda$, $\lambda_k = \lambda'$.

6. $(e[c], \varphi_D, \lambda) \vdash \lambda' \iff (e, \varphi_D, \lambda) \vdash \lambda'$ and $(d, \lambda') \in c$, where d last value in φ_D

7. $(\downarrow \bar{x}.e, \varphi_D, \lambda) \vdash \lambda' \iff (e, \varphi_D, \lambda_{\bar{x}=d}) \vdash \lambda'$, d is first value in φ_x, and $\lambda_{\bar{x}=d}$ modifies λ by setting $\lambda(x)=d$, $\forall x \in \bar{x}$.

Given a data graph $G=(V,E,\alpha)$ and REM e:

$$e(G) = \left\{ (u,v) \in V \times V \;\middle|\; \begin{array}{l} \text{there is data path } \varphi_D \\ \text{from } u \text{ to } v \text{ in } G \\ \text{s.t. } (e, \varphi_D, *^k) \vdash \lambda, \text{ for} \\ \text{some } \lambda \in (\mathbb{N} \cup \{*\})^k. \end{array} \right\}.$$

Example :

1. The REM $\downarrow x.(a[x^{\neq}])^*$ defines the set of pairs linked by paths in which the first data value is different from all others.

2. The REM $\downarrow x.(ab)^*[x^{\neq}]$ defines the set of pairs linked by a path labeled $abab\ldots ab$, s.t. the first and last data values are different.

3. The REM $\Sigma^* \cdot \downarrow x \cdot \Sigma^+ \cdot \Sigma^* [x^=] \cdot \Sigma^*$ defines pairs of nodes linked by a path in which some data value is repeated at least twice.

Observation: The complement of REM (3) is the NP-complete problem we saw before:

Pairs of nodes linked by paths without repeated values.

However, REMs are not closed under complement.

EVALUATION AND CONTAINMENT FOR REMS

For evaluating REM we make use of register automata (RA).

An RA over registers $\{x_1, \ldots, x_k\}$ is defined by the following:

- $Q = Q_d \cup Q_\varepsilon$, where:
  - Q_d data states, Q_ε label states.
- $q_0 \in Q_d$, initial state.
- $F \subseteq Q_\varepsilon$, final states.
- $\tau_0 = \ast^k$, initial configuration.
- $\delta = \delta_d \cup \delta_\varepsilon$, where $\begin{cases} \delta_\varepsilon \subseteq Q_\varepsilon \times \Sigma' \times Q_d \\ \delta_d \subseteq Q_d \times \mathbb{C}_k \times 2^{\{x_1, \ldots, x_k\}} \times Q_\varepsilon \end{cases}$

$\delta_d \subseteq Q_d \times C_k \times 2^{\{x_1,\dots,x_k\}} \times Q_\Sigma$, where:

- C_k set of conditions over $\{x_1,\dots,x_k\}$

Intuitively:

- Configurations are pairs $(q,\tau) \in Q \times \mathbb{N}^k$.
- When in configuration (q,τ) with $q \in Q_\Sigma$, we read $a \in \Sigma$ and move to configuration (s,τ) with $s \in Q_d$ and $(q,a,s) \in \delta_\Sigma$.
- When in configuration (s,τ) with $s \in Q_d$, we read data value d, and move to configuration (q,τ') for which there is $(s,c,I,q) \in \delta_d$ with:
  - $(d,\tau) \models c$
  - $\tau' = \tau_{I=d}$, where $\tau_{I=d}$ means $x \to d$ for each $x \in I$.

Visually:



$\Leftrightarrow (q, a, s) \in \delta_\Sigma$



$\Leftrightarrow (s, c, I, q) \in \delta_D$, $d, \tau \models c$, $\tau' = \tau_{I=d}$

- Given a RA $\mathcal{A}$ and a data path φ_d, we say that $\mathcal{A}$ accepts φ_d, if there is a run of $\mathcal{A}$ on φ_d from configuration (q_0, τ_0) to (q, τ), for $q \in F$.
- We define $\mathcal{L}(\mathcal{A})$ = data paths accepted by $\mathcal{A}$.

Theorem (see, e.g., [Demri, Lazić; 2009])

- Given a RA a, checking if $L(a) \neq \emptyset$ is PSPACE-complete.

How to evaluate REMs?

We make use of the following result:

Proposition [Libkin, Vrgoc; 2012]

Each REM e can be translated, in LOGSPACE, into a register automaton A_e satisfying:

$$\mathcal{L}(A_e) = \{ \varphi_D \mid (e, \varphi_D, *^k) \vdash \lambda, \text{ for some } \lambda \in (\mathbb{N} \cup \{*\})^k \}.$$

We can then use the following procedure to check if $(u, v) \in e(G)$, given an graph G, nodes u, v, and REM e:

- Compute RA a_e in LOGSPACE.
- Take the intersection $a_G^{u,v} \times a_e$, which is a RA that can be computed in LOGSPACE from G, e.
- Check in PSPACE if $\mathcal{L}(a_G^{s,t} \times a_e) \neq \emptyset$.

Corollary: REM-evaluation is in PSPACE (and, in fact, it is PSPACE complete).

What about data complexity?

Theorem [Libkin, Vrgoc; 2012]

REM-evaluation is in NLOGSPACE in data complexity.

Proof idea: Guess, on the fly, a word in G and the associated configurations of A_e.

WALK LOGIC

[Hellings et al; 2013]

- A natural and expressive logic for expressing properties of data graphs.
- WL was proposed as a "yard-stick" for measuring the expressive power of formalisms for data graphs.
- Problem: Awful data complexity.

Walk logic (WL) is defined using:

- path variables $\hat{\pi}_1, \hat{\pi}_2, \ldots$
- for each $\hat{\pi}$ a set of position variables $t_1^{\hat{\pi}}, t_2^{\hat{\pi}}, \ldots$ of sort $\hat{\pi}$.

The grammar of WL is as follows:

$$\phi, \phi' := Ea(t_1^{\hat{\pi}}, t_2^{\hat{\pi}}) \mid t_1^{\hat{\pi}} < t_2^{\hat{\pi}} \mid t_1 \sim t_2 \mid$$

$$\neg\phi \mid \phi \vee \phi' \mid \exists t\, \phi \mid \exists \hat{\pi}\, \phi.$$

not necessarily of same sort!

Semantics of WL: Given

- a data graph $G = (V, E, \alpha)$,
- a mapping θ that assigns a data path φ^π to each path var $\vec{\pi}$,
- a mapping λ that assigns a node in φ^π to each position variable t^π.

We only define semantics for atomic formulas:

- $(G, \theta, \lambda) \models Ea(t_1^{\vec{\pi}}, t_2^{\vec{\pi}}) \iff (\lambda(t_1^{\vec{\pi}}), a, \lambda(t_2^{\vec{\pi}})) \in E$
- $(G, \theta, \lambda) \models t_1^{\vec{\pi}} < t_2^{\vec{\pi}} \iff \lambda(t_1^{\vec{\pi}})$ appears before $\lambda(t_2^{\vec{\pi}})$ in $\theta(\vec{\pi})$.
- $(G, \theta, \lambda) \models t_1 \sim t_2 \iff \alpha(\lambda(t_1^{\vec{\pi}})) = \alpha(\lambda(t_2^{\vec{\pi}}))$ (same data value).

Examples:

- Positions linked by data paths in which all data values are different:

$$\varphi(t_1^{\pi}, t_2^{\pi}) := \exists \pi \Big[\neg \exists t_3^{\pi} \big(t_3^{\pi} < t_1^{\pi} \vee t_2^{\pi} < t_3^{\pi} \big) \wedge \forall t_3^{\pi}, t_4^{\pi}. \big(t_3^{\pi} < t_4^{\pi} \rightarrow (t_3 \neq t_4) \big) \Big]$$

Example: The following query in WL checks if $G=(V,E,\alpha)$ has a Hamiltonian cycle, assuming $\alpha(u) \neq \alpha(v)$ for each $u,v \in V$ with $u \neq v$:

$$\exists \pi \Big[\forall t_1^{\pi}, t_2^{\pi} \Big(t_1^{\pi} < t_2^{\pi} \longrightarrow t_1^{\pi} \neq t_2^{\pi} \Big) \wedge \forall \pi' \forall t_1^{\pi'} \exists t_2^{\pi} \Big(t_1^{\pi'} \sim t_2^{\pi} \Big) \Big]$$

Corollary: WL is NP-hard in data complexity.

In fact, the situation is much worst:

Theorem [Barceló, Fontaine, Lin; 2013]
WL is non-elementary in data complexity; i.e., for every $k \geq 1$ there is WL formula ψ_k s.t. evaluating ψ_k is k-EXPTIME-hard.

REGULAR LOGIC

- An attempt to bridge between WL and REMs:
  * More expressive than REMs.
  * Better complexity than WL.
- Idea: Closing REMs under:
  * Node quantification
  * Path quantification
  * Register quantification.

Register logic (RL) is defined via the following grammar, assuming x, y node variables, $\hat{\pi}, \hat{\pi}'$ path variables, $\bar{r}, \bar{r}'$ register value variables, and e an REM:

$$\text{atom} := x = y \mid \hat{\pi} = \hat{\pi}' \mid \bar{r} = \bar{r}' \mid \bar{r} = \bar{\ast} \mid (x, \hat{\pi}, y) \mid e(\hat{\pi}, \bar{r}, \bar{r}')$$

$$\phi := \text{atom} \mid \neg\phi \mid \phi \vee \phi \mid \exists x \phi \mid \exists \hat{\pi} \phi \mid \exists \bar{r} \phi$$

Semantics of RL: Given,

- data graph $G = (V, E, \alpha)$,
- mapping λ from node variables to nodes in V
- mapping Θ from path variables to paths ψ in G,
- mapping χ from register variables to tuples in $(\mathbb{N} \cup \{\#\})^k$.

Then we have:

1. $(G, \lambda, \Theta, \gamma) \models \bar{v} = \bar{*} \iff \gamma(\bar{v}) = *^k$.

2. $(G, \lambda, \Theta, \gamma) \models (x, \bar{\pi}, y) \iff \Theta(\bar{\pi})$ is a path from $\lambda(x)$ to $\lambda(y)$

3. $(G, \lambda, \Theta, \gamma) \models e(\tilde{\pi}, \bar{v}, \bar{v}') \iff \Theta(\tilde{\pi}) = \varphi$, $(e, \varphi_D, \gamma(\bar{v})) \models \gamma(\bar{v}')$

Example: The RL formula

$$\exists \hat{\pi}\,[(x,\hat{\pi},y) \wedge \exists z\, \forall v\,(e(\hat{\pi},v,v) \rightarrow \exists z'\, \exists \bar{\pi}'\,(z',\hat{\pi}',z) \wedge e'(\bar{\pi},v,v))]$$

where $e := a^*[x^=].a^*$, $e' := \varepsilon[x^=].a^*$

checks if there is path φ from x to y and node z s.t.:

- for every node v in φ, there exists a path from such node v to z.

Theorem [Barceló, Fontaine, Lin ; 2013]

- RL-evaluation is EXPSPACE-complete.
- RL-evaluation is PSPACE-complete in data complexity.

- Unlike WL, complexity is elementary.
- Unfortunately, data complexity is still quite high.

Corollary (to the proof):

The positive fragment of RL admits efficient evaluation in data complexity.

Different ways of obtaining expressive languages that
- extend REMs,
- allow for tractable data complexity,

can be obtained by using XPath-like features [Libkin, Martens, Vrgoc; 2013].