M2 – DAC - LODAS - 2019

MU5IN860 Linked Open Data, Apprentisssage Symbolique

Bernd Amann

SU

23 octobre 2019

SPARQL : Evaluation et Optimisation

Outline

1 - Algèbre SPARQL

- Algèbre de mappings
- Plans d'exécution ARQ-SPARQL

Sémantique de SPARQL

Approche

La sémantique d'une requête SPARQL est définie par une algèbre sur des mappings de variables présentes dans les motifs de graphes :

- similaire à l'algèbre relationnel
- opérations : jointure, union, différence, jointure externe

Objectif

- détecter des incophérences entre l'évaluation et sémantique formelle
- définir des techniques d'optimisation
- évaluer la complexité du langage SPARQL

Exemple

Turtle bibliosem

```
@prefix : <http://example.org/ns#> .

:book1 :title "SPARQL Tutorial" ; :price 42 ; :editor :jena .
:book2 :title "The Semantic Web" ; :price 23 .
:book3 :title "RDF Framework"; :prix 53 .
:book4 :title "SPARQL pour les Nuls" ; :editor :pourlesnuls .
:pourlesnuls :address "Paris" .
```

Requête optional2

```
SELECT ?x ?editor ?price
  FROM <bibliosem.ttl>
 WHERE { ?x :editor ?editor .
  OPTIONAL { ?x :price ?price }
}
```

Résultat de optional2

x	editor	price
:book4	:pourlesnuls	
:book1	:jena	42

Le résultat est un ensemble de mapping de variables.

Mappings de variables

Variables, Termes, Patterns

- V : ensemble de variables ;
- $T = B \cup U \cup L$: ensemble de termes ;
- $p \in (V \cup T) \times (V \cup T) \times (V \cup T)$: triple pattern avec variables $var(p) \subseteq V$.

Mapping de variables ν : fonction μ et ensemble ν

- $\mu : V \rightarrow T$: fonction partielle de variables vers les termes ;
- $\nu = \{(x, y) | x \in V \wedge y \in T \wedge y = \mu(x)\}$: ensemble de couples $(x, y) \in V \times T$.

Domaine $dom(\nu)$ d'un binding ν

- $dom(\nu) = \{x | \exists y \in T : (x, y) \in \nu\}$
- $\mu(p)$ = triplet obtenu après avoir remplacé chaque variable $v \in var(p)$ par son mapping $\nu(v)$.

Compatibilité de mapping de variables

Deux mapping ν_1 et ν_2 sont compatibles si pour toutes les variables $x \in dom(\nu_1) \cap dom(\nu_2)$, $\mu_1(x) = \mu_2(x)$ (l'union $\nu_1 \cup \nu_2$ est aussi un mapping).

Motifs et mapping de variables

Requête optional2

```
SELECT ?x ?editor ?price
  FROM <bibliosem.ttl>
 WHERE { ?x :editor ?editor .
   OPTIONAL { ?x :price ?price }
}
```

Résultat de optional2

```
-------------------------------
| x     | editor      | price |
===============================
| :book4 | :pourlesnuls |      |
| :book1 | :jena       | 42    |
-------------------------------
```

Mappings (de variables) pour les deux motifs de triplets :

- ?x :editor ?editor
  - $\nu_{11} = \{(?x, :book1), (?editor, :jena)\}$
  - $\nu_{12} = \{(?x, :book4), (?editor, :pourlesnuls)\}$
- ?x :price ?price
  - $\nu_{21} = \{(?x, :book1), (?price, 42)\}$
  - $\nu_{22} = \{(?x, :book2), (?price, 23)\}$

- ν_{11} est compatible avec $\nu_{21} \Rightarrow$ l'union $\nu_{11} \cup \nu_{21}$ est un mapping
- ν_{11} n'est pas compatible avec $\nu_{22} \Rightarrow$ on ne peut pas faire l'union
- ν_{12} n'est pas compatible ni avec ν_{21}, ni avec $\nu_{22} \Rightarrow$ on ne peut pas faire l'union

La réponse est l'ensemble de mappings : $\{\nu_{11} \cup \nu_{21}, \nu_{12}\}$

Motifs et mapping de variables

Requête optfilter

```
SELECT ?x ?editor ?price
FROM <bibliosem.ttl>
WHERE { ?x :editor ?editor .
        OPTIONAL { ?x :price ?price }
        FILTER (!bound(?price)) }
```

Résultat de optfilter

```
-------------------------------
| x      | editor       | price |
===============================
| :book4 | :pourlesnuls |       |
-------------------------------
```

Mappings (de variables) pour les deux motifs de triplets :

- ?x :editor ?editor
  - $\nu_{11} = \{(?x, :book1), (?editor, :jena)\}$
  - $\nu_{12} = \{(?x, :book4), (?editor, :pourlesnuls)\}$
- ?x :price ?price
  - $\nu_{21} = \{(?x, :book1), (?price, 42)\}$
  - $\nu_{22} = \{(?x, :book2), (?price, 23)\}$

La réponse est un ensemble de mappings :
$\{\nu_{12}\} = \{\{(?x, :book4), (?editor, :pourlesnuls)\}\}$

Opérations sur des ensembles de mappings

Soient deux *ensembles de mapping* Ω_1 et Ω_2. On définit les opérations suivantes :

- Jointure :

$$\Omega_1 \bowtie \Omega_2 = \{\nu_1 \cup \nu_2 | \nu_1 \in \Omega_1 \text{ et } \nu_2 \in \Omega_2 \text{ sont compatibles}\}$$

- Union :

$$\Omega_1 \cup \Omega_2 = \{\nu | \nu \in \Omega_1 \text{ ou } \nu \in \Omega_2\}$$

- Différence :

$$\Omega_1 \setminus \Omega_2 = \{\nu_1 | \nu_1 \in \Omega_1, \forall \nu_2 \in \Omega_2 : \nu_1 \text{ et } \nu_2 \text{ sont incompatibles}\}$$

- Jointure externe gauche (left outer join) :

$$\Omega_1 \mathbin{⟕} \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$$

Interprétation algébrique d'un motif de graphe P

L'interprétation d'un motif de graphe E sur un graphe RDF D est un ensemble de mappings noté $[\![E]\!]_D$
$[\![\circ]\!]_D$ est défini récursivement :

- $[\![p]\!]_D = \{\nu | \nu(p) \in D\}$: l'ensemble de mappings générés par un triple pattern p dans D.
- $[\![(E_1 \ . \ E_2)]\!]_D = [\![E_1]\!]_D \bowtie [\![E_2]\!]_D$
- $[\![(E_1 \ \mathit{OPTIONAL} \ E_2)]\!]_D = [\![E_1]\!]_D \mathbin{⟕} [\![E_2]\!]_D$
- $[\![(E_1 \ \mathit{UNION} \ E_2)]\!]_D = [\![E_1]\!]_D \cup [\![E_2]\!]_D$

Exemple : Interprétation algébrique de motifs

- $[\![$?x :editor ?editor $]\!]_D = \{\nu_{11}, \nu_{12}\}$:
  - ν_{11} = {(?x, ' :book1'), (?editor, :jena) }
  - ν_{12} = {(?x, ' :book4'), (?editor, :pourlesnuls)}
- $[\![$?x :price ?price $]\!]_D = \{\nu_{21}, \nu_{22}\}$:
  - ν_{21} = {(?x, ' :book1'), (?price, 42)}
  - ν_{22} = {(?x, ' :book2'), (?price, 23)}
- $[\![$ { ?x :editor ?editor } OPTIONAL { ?x :price ?price } $]\!]_D$:
  - $\{\nu_{11}, \nu_{12}\} \ltimes\!\!\!\bowtie \{\nu_{21}, \nu_{22}\}$
  - $(\{\nu_{11}, \nu_{12}\} \bowtie \{\nu_{21}, \nu_{22}\}) \cup (\{\nu_{11}, \nu_{12}\} \setminus \{\nu_{21}, \nu_{22}\})$
  - $(\{\nu_{11} \cup \nu_{21}\}) \cup (\{\nu_{12}\})$
  - $\{\nu_{11} \cup \nu_{21}, \nu_{12}\}$
  - { {(?x, ' :book1'), (?editor, :jena), (?price, 42)},
    {(?x, ' :book4'), (?editor, :pourlesnuls)} }

Rappel : $\Omega_1 \ltimes\!\!\!\bowtie \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$

Interprétation d'un filtre

Sémantique d'un filtre *P FILTER R*

$[\![E \ FILTER \ (R)]\!]_D = \{\nu | \nu \in [\![E]\!]_D \text{ et } \nu \models R\}$

Satisfaction $\nu \models R$

Un mapping ν satisfait R, dénoté $\nu \models R$, si :

1. R est $bound(?X)$ et $?X \in dom(\nu)$;
2. R est $?X = c$, $?X \in dom(\nu)$ et $\nu(?X) = c$;
3. R est $?X = ?Y$, $?X \in dom(\nu)$, $?Y \in dom(\nu)$ et $\nu(?X) = \nu(?Y)$;
4. R est $(\neg R1)$, $R1$ est une condition, et $\nu \not\models R1$;
5. R est $(R1 \vee R2)$, $R1$ et $R2$ sont des conditions et $\nu \models R1$ ou $\nu \models R2$;
6. R est $(R1 \wedge R2)$, $R1$ et $R2$ sont des conditions et $\nu \models R1$ et $\nu \models R2$.

Complexité de SPARQL

Problème de décision

Est-ce qu'un mapping Ω donné est valable pour une requête Q et un graphe D donné ?

Complexité de SPARQL [PAG06]

$\text{SPARQL}_{.,FILTER}$	$\bowtie$ et FILTER	$O(\|Q\| \times \|D\|)$
$\text{SPARQL}_{.,FILTER,UNION}$	$\bowtie$, FILTER et $\cup$	NP-complet
$\text{SPARQL}_{.,FILTER,OPTIONAL}$	$\bowtie$, FILTER et $⟕$	PSPACE-complet

Bibliographie

GHM04 C. Guttierez, C. Hurtado, A. Mendelzon, Foundations of Semantic Web Databases, PODS 2004

RDFSEM04 RDF Semantics, W3C Recommendation 10 February 2004, http ://www.w3.org/TR/rdf-mt/

PAG06 Jorge Perez, Marcelo Arenas, Claudio Gutierrez, Semantics and Complexity of SPARQL, http ://arxiv.org/abs/cs.DB/0605124, Mai 2006

1 - Algèbre SPARQL

- Algèbre de mappings
- Plans d'exécution ARQ-SPARQL

Exemple : Plans ARQ-SPARQL

Algèbre pour l'évaluation de requêtes SPARQL :

- traduction de l'expression SPARQL
- optimisation logique
- optimisation physique
- évaluation

Plan simple

Requête ex2.1

```
SELECT ?x ?name
FROM <ex2.ttl>
WHERE { ?x foaf:name ?name }
```

Result of ex2.1

```
-------------
| name      |
=============
| "Alice"   |
| "Bob"     |
| "Alice"   |
-------------
```

Expression algébrique de ex2.1

```
(prefix ((foaf: <http://xmlns.com/foaf/0.1/>))
  (project (?name)
    (bgp (triple ?x foaf:name ?name))))
```

Opérateurs ARQ-SPARQL

`(triple p)`	$[\![p]\!]_D$	retourne tous les **mappings** ν d'un motif de triplet p
`(bgp` p_1 p_2 `...` p_n`)`	$[\![p_1]\!]_D \bowtie [\![p_2]\!]_D \bowtie ... \bowtie [\![p_n]\!]_D$	retourne la **jointure** $\bowtie$ des mappings d'un ensemble de motifs du triplets $\{p_1, p_2, ..., p_n\}$ (basic graph pattern)
`(leftjoin` E_1 E_2`)`	$[\![E_1]\!]_D \rtimes [\![E_2]\!]_D$	retourne la **jointure externe gauche** $\rtimes$ entre les mappings de ses deux sous-expressions
`(union` E_1 E_2`)`	$[\![E_1]\!]_D \cup [\![E_2]\!]_D$	retourne la **union** $\cup$ entre les mappings de ses deux sous-expressions
`(filter (F) E)`	$\{\nu \mid \nu \in [\![E]\!]_D \text{ et } \nu \models F\}$	applique le filtre P aux mappings de E
`(project (`v_1 v_2 `...` v_n`))`		retourne les mappings pour les variables v_1 v_2 `...` v_n

Motifs simples (AND)

Requête couleur1a

```
PREFIX : <http://colors.org/ns#>
SELECT DISTINCT ?a ?b ?c ?d ?f ?g
FROM <couleur.ttl>
WHERE {
  ?a :e ?b ; :e ?c .
  ?b :e ?a ; :e ?c .
  ?c :e ?a ; :e ?b ; :e ?f ; :e ?g .
}
```

Expression algébrique de couleur1a

```
(prefix ((: <http://colors.org/ns#>))
  (distinct
    (project (?a ?b ?c ?d ?f ?g)
      (bgp
        (triple ?a :e ?b)
        (triple ?a :e ?c)
        (triple ?b :e ?a)
        (triple ?b :e ?c)
        (triple ?c :e ?a)
        (triple ?c :e ?b)
        (triple ?c :e ?f)
        (triple ?c :e ?g)
      ))))
```

Union = ∪

Requête union3

```
SELECT ?x ?price ?title
FROM <ex5.ttl>
WHERE { { ?x ns:price ?price . }
        UNION
        { ?x dc:title ?title . } }
```

Result of union3

x	price	title
ns:book1	42	
ns:book2	23	
ns:book3		"RDF Framework"
ns:book1		"SPARQL Tutorial"
ns:book2		"The Semantic Web"

Expression algébrique de construct

```
(prefix ((dc: <http://purl.org/dc/elements/1.1/>)
         (ns: <http://asws.org/ns#>))
  (union
    (bgp (triple ?x ns:price ?price))
    (bgp (triple ?x dc:title ?title))))
```

OPTIONAL = ⟕

Requête optional3

```
SELECT ?title ?price ?editor
FROM <bibliosem.ttl>
WHERE { ?x :title ?title .
        OPTIONAL { ?x :price ?price }
        OPTIONAL { ?x :editor ?editor } }
```

Result of optional3

title	price	editor
"SPARQL pour les Nuls"		:pourlesnuls
"RDF Framework"		
"The Semantic Web"	23	
"SPARQL Tutorial"	42	:jena

Expression algébrique de optional3

```
(prefix ((: <http://example.org/ns#>))
  (project (?title ?price ?editor)
    (leftjoin
      (leftjoin
        (bgp (triple ?x :title ?title))
        (bgp (triple ?x :price ?price)))
      (bgp (triple ?x :editor ?editor)))))
```

OPTIONAL et FILTER

Requête optfilter

```
SELECT ?x ?editor ?price
FROM <bibliosem.ttl>
WHERE { ?x :editor ?editor .
      OPTIONAL { ?x :price ?price }
      FILTER (!bound(?price)) }
```

Result of optfilter

```
-------------------------------
| x      | editor       | price |
===============================
| :book4 | :pourlesnuls |       |
-------------------------------
```

Expression algébrique de optfilter

```
(prefix ((: <http://example.org/ns#>))
  (project (?x ?editor ?price)
    (filter (! (bound ?price))
      (leftjoin
        (bgp (triple ?x :editor ?editor))
        (bgp (triple ?x :price ?price))))))
```

Graphes nommés = graph

Requête ex10

```
SELECT  ?a ?b ?title ?price
FROM NAMED <ex5.ttl>
FROM NAMED <ex6.ttl>
{ { GRAPH ?b  { ?x dc:title ?title . } }
  OPTIONAL
  { GRAPH ?a  { ?x ns:price ?price . } } }
```

Expression algébrique de ex10

```
(prefix ((dc: <http://purl.org/dc/elements/1.1/>)
         (ns: <http://asws.org/ns#>))
  (project (?a ?b ?title ?price)
    (leftjoin
      (graph ?b
        (bgp (triple ?x dc:title ?title)))
      (graph ?a
        (bgp (triple ?x ns:price ?price))))))
```

Expressions de chemins = path

Requête ex12

```
SELECT ?xn ?yn
FROM <ex11.ttl>
WHERE  { ?x foaf:name ?xn;
         (foaf:knows+ | ^foaf:knows+)
                  [ foaf:name ?yn ] . }
```

Expression algébrique de ex12

```
(prefix ((foaf: <http://xmlns.com/foaf/0.1/>))
  (project (?xn ?yn)
    (sequence
      (bgp (triple ?x foaf:name ?xn))
      (path ?x (alt (path+ foaf:knows) (reverse (path+ foaf:knows))) ??0)
      (bgp (triple ??0 foaf:name ?yn)))))
```

Outline

Optimisation de requêtes (SPARQL)

Qu'est-ce qu'on peut optimiser ?

- requête : simplification / réduction logique
- plan d'exécution :
  - réécriture algébrique
  - reordonner les opérations
- opérations :
  - indexation
  - algorithmes d'évaluation
- technologie :
  - relationnel, noSQL, MapReduce, native

L'espace de solutions est très grand.

Optimisation de requêtes (SPARQL)

Simplification des requêtes

- Utiliser les règles RDF/S pour supprimer des motifs de triplets redondants (« inverse » de la saturation)
- Exemple : :a :p :b . :p rdf:type rdf:Property . est équivalent à :a :p :b .

Réécriture algébrique du plan d'exécution

- évaluer les motifs de triplets sélectifs d'abord ;
- réordonner les jointures (motifs de triplets).

Règles de réécritures et normalisation

Règles de réécriture

1. $\bowtie$ et $\cup$ sont associatifs et commutatifs.
2. distribution jointures et unions :
   1. $(E_1 \bowtie (E_2 \cup E_3)) \equiv((E_1 \bowtie E_2) \cup (E_1 \bowtie E_3))$.
   2. $(E_1 \ltimes (E_2 \cup E_3)) \equiv ((E_1 \ltimes E_2) \cup (E_1 \ltimes E_3))$.
   3. $((E_1 \cup E_2) \rtimes E_3) \equiv ((E_1 \rtimes E_3) \cup (E_2 \rtimes E_3))$.
3. pousser les sélections dans l'union :
   - $\sigma_R(E_1 \cup E_2) \equiv \sigma_R(E_1) \cup \sigma_R(E_2)$.

Normalisation

Tous les motifs de graphe p peuvent être réécrits en une union de motifs sans union :

$$p \equiv \bigcup_i E_i$$

où E_i sont sans union $\cup$.

UNION : Normalisation

Expression algébrique de couleur4

```
(prefix ((: <http://colors.org/ns#>))
  (distinct
    (project (?a ?b ?c)
      (leftjoin
        (bgp
          (triple ?a :e ?b)
          (triple ?b :e ?c)
        )
        (union
          (union
            (bgp (triple ?a :e ?c))
            (bgp (triple ?a :e ?d)))
          (bgp (triple ?b :e ?d)))))))
```

- p_1=(?a :e ?b), p_2=(?b :e ?c), p_3=(?a :e ?c), p_4=(?a :e ?d), p_5=(?b :e ?d)
- algèbre : $(p_1 \bowtie p_2) \bowtie\!\!\!\!- ((p_3 \cup p_4) \cup p_5)$
- règle 1 : $(p_1 \bowtie p_2) \bowtie\!\!\!\!- (p_3 \cup p_4 \cup p_5)$
- règle 2 : $((p_1 \bowtie p_2) \bowtie\!\!\!\!- p_3) \cup ((p_1 \bowtie p_2) \bowtie\!\!\!\!- p_4) \cup ((p_1 \bowtie p_2) \bowtie\!\!\!\!- p_5)$

Expression algébrique de couleur4n2

```
(prefix ((: <http://colors.org/ns#>))
  (distinct
    (project (?a ?b ?c)
      (union
        (union
          (leftjoin
            (bgp
              (triple ?a :e ?b)
              (triple ?b :e ?c)
            )
            (bgp (triple ?a :e ?c)))
          (leftjoin
            (bgp
              (triple ?a :e ?b)
              (triple ?b :e ?c)
            )
            (bgp (triple ?a :e ?d))))
        (leftjoin
          (bgp
            (triple ?a :e ?b)
            (triple ?b :e ?c)
          )
          (bgp (triple ?b :e ?d)))))))
```

Évaluation de motifs sans UNION

Algorithme $Eval_D(E, \Omega)$ (top-down, récursif)

- Ω : ensemble de bindings déjà calculé
- Ω est initialisé avec l'ensemble de mapping vide : $\Omega = \{\emptyset\}$
- D : graphe RDF

expression E	$Eval_D(E, \Omega)$
p (motif de triplet)	$(\Omega \bowtie [\![p]\!]_D)$
$E_1.E_2$	$Eval_D(E_2, Eval_D(E_1, \Omega))$
E_1 OPTIONAL E_2	$Eval_D(E_1, \Omega) \ltimes Eval_D(E_2, \mathbf{Eval_D(E_1, \Omega)})$
E_1 FILTER R	$\{\nu \in Eval_D(E_1, \Omega) \mid \nu \models R\}$

Hypothèse OPTIONAL

Pour évaluer E_1 OPTIONAL E_2, seulement les mappings dans Ω compatibles avec les mappings dans $\mathbf{Eval_D(E_1, \Omega)}$ sont utiles.

Sémantique d'évaluation et sémantique formelle

Exemple : SPARQL

SELECT $*$
WHERE { E1 **OPTIONAL** { E2 **OPTIONAL** E3 } }

Exemple : Expression algébrique

$E = ((E1 \ltimes (E2 \ltimes E3))$

Sémantique formelle : $[\![E]\!]_D$

1. $[\![((E1 \ltimes (E2 \ltimes E3))]\!]_D$
2. $[\![E1]\!]_D \ltimes ([\![E2]\!]_D \ltimes [\![E3]\!]_D)$

Sémantique d'évaluation : $Eval_D(E, \emptyset)$

1. $Eval_D(E1 \ltimes (E2 \ltimes E3), \{\emptyset\})$
2. $Eval_D(E1, \{\emptyset\}) \ltimes Eval_D(E2 \ltimes E3, Eval_D(E1, \{\emptyset\}))$
3. $[\![E1]\!]_D \ltimes (Eval_D(E2, [\![E1]\!]_D) \ltimes Eval_D(E3, Eval_D(E2, [\![E1]\!]_D)$
4. $[\![E1]\!]_D \ltimes (([\![E1]\!]_D \bowtie [\![E2]\!]_D) \ltimes ([\![E1]\!]_D \bowtie [\![E2]\!]_D \bowtie [\![E3]\!]_D))$

$\Rightarrow [\![E]\!]_D \not\equiv Eval_D(E, \{\emptyset\})$ (la sémantique formelle est différente de la sémantique d'évaluation)

Motif bien-formé : sémantique formelle et évaluation

Motif bien formé

Un motif E est bien formé si pour chaque occurrence d'un sous-motif $E' = E_1 \bowtie E_2$ de E et pour chaque variable $?x$ dans E : si $?x$ apparaît dans E_2 et en dehors de E', alors elle apparaît également dans E_1.

Exemples

- { ?x a ?y ⋈ { ?y b ?z ⋈ { ?x c ?u } } : *pas bien formé*
- { ?x a ?y ⋈ { ?y b ?z ⋈ { ?u c ?v } } : *bien formé*

Théorème

Si E est un motif bien formé, alors $[\![E]\!]_D = Eval_D(E, \{\emptyset\})$

2 - Optimisation de requêtes SPARQL

- Ordonnancement de jointures
- Indexation de données
- SPARQL et Inférence

Reordonner les Jointures

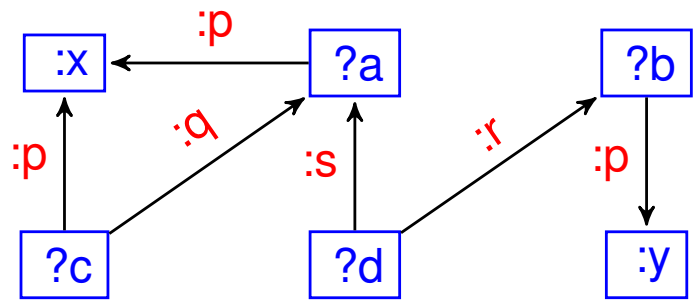
- Au niveau logique la jointure est **associative** et **commutative**.
- Au niveau physique :
  - Le coût d'une jointure dépend de la taille de ses arguments (ensembles de mappings).
  - Un plan de jointure est plus ou moins "parallélisable".

$\Rightarrow$ **L'ordre des jointures est important.**

Ordonnancement : graphes de jointures / variables

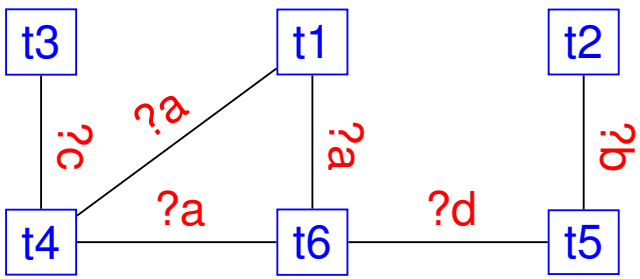
t1 : (?a :p :x) t2 : (?b :p :y)
t3 : (?c :p :x) t4 : (?c :q ?a)
t5 : (?d :r ?b) t6 : (?d :s ?a)

Motif de Graphe



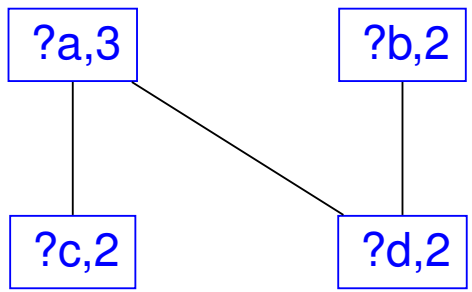
Graphe de jointures

- noeud t = motif de triplet t
- arc étiqueté (t, t', V) = t et t' partagent les variables V



Graphe de variables

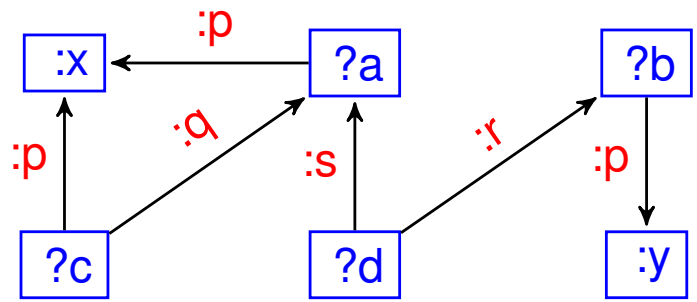
- noeud étiqueté (v, n) = variable v est partagé par n triplets.
- arc (v, v') = v et v' apparaissent dans le même motif de triplet



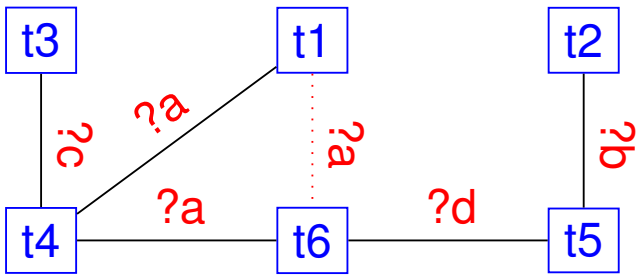
Ordonnancement des jointures

t1 : (?a :p :x) t2 : (?b :p :y)
t3 : (?c :p :x) t4 : (?c :q ?a)
t5 : (?d :r ?b) t6 : (?d :s ?a)

Motif de Graphe



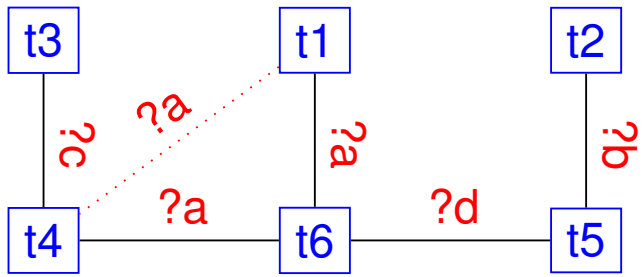
Graphe de Jointure Acyclique



Plans linéaires :

- ((((t1$\bowtie_{?a}$ t4)$\bowtie_{?a}$ t6)$\bowtie_{?c}$ t3)$\bowtie_{?d}$ t5)$\bowtie_{?b}$ t2
- ((((t1$\bowtie_{?a}$ t4)$\bowtie_{?c}$ t3)$\bowtie_{?a}$ t6)$\bowtie_{?d}$ t5)$\bowtie_{?b}$ t2
- ((((t3$\bowtie_{?c}$ t4)$\bowtie_{?c}$ t6)$\bowtie_{?a}$ t1)$\bowtie_{?d}$ t5)$\bowtie_{?b}$ t2
- ...

Graphe de Jointure Acyclique

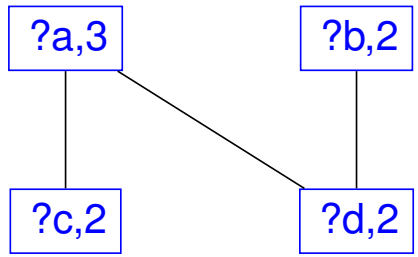


Plans touffus (favorise la parallélisation) :

- ((t1$\bowtie_{?a}$ t6)$\bowtie_{?a}$ (t3$\bowtie_{?c}$ t4))$\bowtie_{?d}$ (t2$\bowtie_{?b}$ t5)
- ((t1$\bowtie_{?a}$ t6)$\bowtie_{?d}$ (t2$\bowtie_{?b}$ t5))$\bowtie_{?a}$ (t3$\bowtie_{?c}$ t4)
- ...

Ordonnancement de jointures

```
t1 : (?a :p x)    t4 : (?c :q ?a)
t2 : (?b :p y)    t5 : (?d :r ?b)
t3 : (?c :p x)    t6 : (?d :s ?a)
```



Objectif

Produire un plan parallélisables avec des **jointures n-aires** (une jointure par variable).

Ensemble indépendant d'un *graphe de variables*

- Ensemble indépendant : sous-ensemble de noeuds (variables) sans arcs (triplets) en commun
- Ensemble indépendant de poids maximal : ensemble de jointures n-aires qui permet de joindre le plus de triplets.

Algorithme [Tsialiamanis et.al. 2012]

- Calculer tous les ensembles indépendants maximaux *I* sur le graphe de variables : $\{?a, ?b\}$
- Appliquer heuristiques sur *I* pour choisir l'ensemble le plus sélectif.
- Utiliser heuristiques (sélectivité, chemins d'accès possibles) pour ordonner les jointures et choisir les index.

Plan résultat : $\bowtie_{?c} (\bowtie_{?d} (\bowtie_{?a} (t1, t6, t4), \bowtie_{?b} (t2, t5)), t3)$

Ordonnancement par sélectivité des motifs

- H1 : Nombre et position des variables dans les motifs :
  - $(s, p, o) < (s, ?, o) < (?, p, o) < (s, p, ?) < (?, ?, o) < (s, ?, ?) < (?, p, ?) < (?, ?, ?)$
  - s, p, o désignent des constantes (URI, littéraux), ? désigne une variable
- H2 : Position des variables de jointure dans les motifs joints :
  - $p \bowtie o < s \bowtie p < s \bowtie o < o \bowtie o < s \bowtie s < p \bowtie p$
  - $p \bowtie o$ est équivalent à une jointure $(a, ?x, b), (c, d, ?x)$
- H3 : Les littéraux sont plus sélectives que les URI
- ...

2 - Optimisation de requêtes SPARQL

- Ordonnancement de jointures
- Indexation de données
- SPARQL et Inférence

Indexation de données RDF

Pourquoi indexer les triplets ?

- Chaque motif de triplet correspond à une sélection d'un sous-ensemble de triplets T.
- T peut contenir des milliards de triplets.
- Un parcours séquentiel (sans index) n'est pas efficace.
- Le problème est encore plus important pour les jointures : une évaluation naïve (boucle imbriquée sans index) de n jointures lit T^N triplets (coût exponentiel dans le nombre de jointures)

Index

Structure de données qui permet de chercher « rapidement » les triplets qui satisfont un motif de triplet.

Indexation exhaustive

Observation

L'utilité d'un index dépend du motif :

- **(?x, 'a', ?z)** $\rightarrow$ index sur les propriétés ;
- **('s', ?y, ?z)** $\rightarrow$ index sur les sujets ;
- **('s', 'p', ?z)** $\rightarrow$ index sur les sujets et les propriétés etc... ;

Solution : Indexation exhaustive

- On créé un index pour chacune des 6 permutations de **S**ubject/**P**roperty/**O**bject : SPO, SOP, OSP, OPS, PSO, POS
- Solution adoptée par RDF-3X, Hexastore, **Jena TDB**

Indexation exhaustive : Exemple RDF-3X [Neuman'08]

Caractéristiques

- compression des valeurs par dictionnaire
- indexation exhaustive : 6 index *SPO*, *SOP*, *OSP*, *OPS*, *PSO*, *POS* sous forme d'arbres B+ optimisés $\rightarrow$ recherche par intervalle, merge join
- génération de plan d'exécution algébrique
- **architecture RISC** : index scan, jointure et filtrage

RDF-3X : Stockage et indexation de données

Compression par dictionnaire

S	P	O
0	1	2
0	3	4
0	5	6
2	3	7
...	...	...

ID	Value
0	URI_1
1	URI_1
2	URI_1
3	URI_1
...	...

Indexation : 6 arbres B+

- un arbre B+ par ordre lexicographique SPO, SOP, PSO, POS, OSP, OPS
- compression par encodage delta
- scan efficace
- merge-join

Principe du merge-join

Principe du merge-join (jointure par fusion)

Jointure $R \bowtie_{R.A=S.B} S$:

- On tri respectivement les tables R et S sur les attributs $R.A$ et $S.B$.
- Pour faire la jointure, on parcourt ensuite d'un manière synchronisée les deux tables dans l'ordre de tri.
- A chaque itération, on génère $r \bowtie s$ ssi $r.A = s.B$ (r et s sont des n-uplets dans les deux tables).

Coût : $O(log(R) + log(S))$ (tri) + $O(R) + O(S)$ (fusion).

Observation

Si R est indexé sur A et S est indexé sur B avec un arbre B+, il n'est plus nécessaire d'ordonner les données (l'arbre B+ permet un parcours séquentiel dans l'ordre de la clé).

RDF-3X : merge-join avec arbre B+

- Motif : ?a p1 r1 . ?a p2 ?b . ?c p3 ?b .
- Traduction : ?a 123 45 . ?a 345 ?b . ?c 678 ?b .
- Filtre et Jointure : $(\sigma_{p=123,o=45}(POS) \bowtie_{?a} \sigma_{p=345}(PSO)) \bowtie_{?b} (\sigma_{p=678}(POS)$

	P=123	O=45	S
	3	7	2
	...	...	...
→	*123*	*45*	**23**
→	*123*	*45*	**56**
→	*123*	*45*	**78**
	...	...	...
	345	67	23
	...	...	...
	678	67	99
	678	103	456
	...	...	...

$\bowtie_{?a}$

	P=345	S= ?a	O
	3	2	7
	...	...	...
	123	23	45
	...	...	...
→	*345*	23	**67**
→	*345*	66	188
→	*345*	78	**103**
	...	...	...
	678	99	67
	678	456	103
	...	...	...

$\bowtie_{?b}$

	P=678	O= ?b	S
	3	7	2
	...	...	...
	123	45	23
	...	...	...
	345	67	23
	...	...	...
→	*678*	67	**99**
→	*678*	103	**456**
	...	...	...

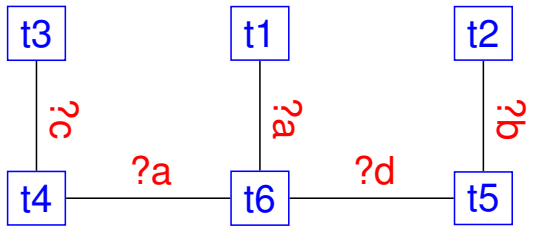
- Les arbre B+ POS et PSO permettent un accès rapide à des intervalles de triplets.
- Chaque motif de triplet génère un parcours séquentiel de l'index correspondant (jointure par fusion).
- Coût total : 3*taille de l'index (linéaire dans la taille des données)

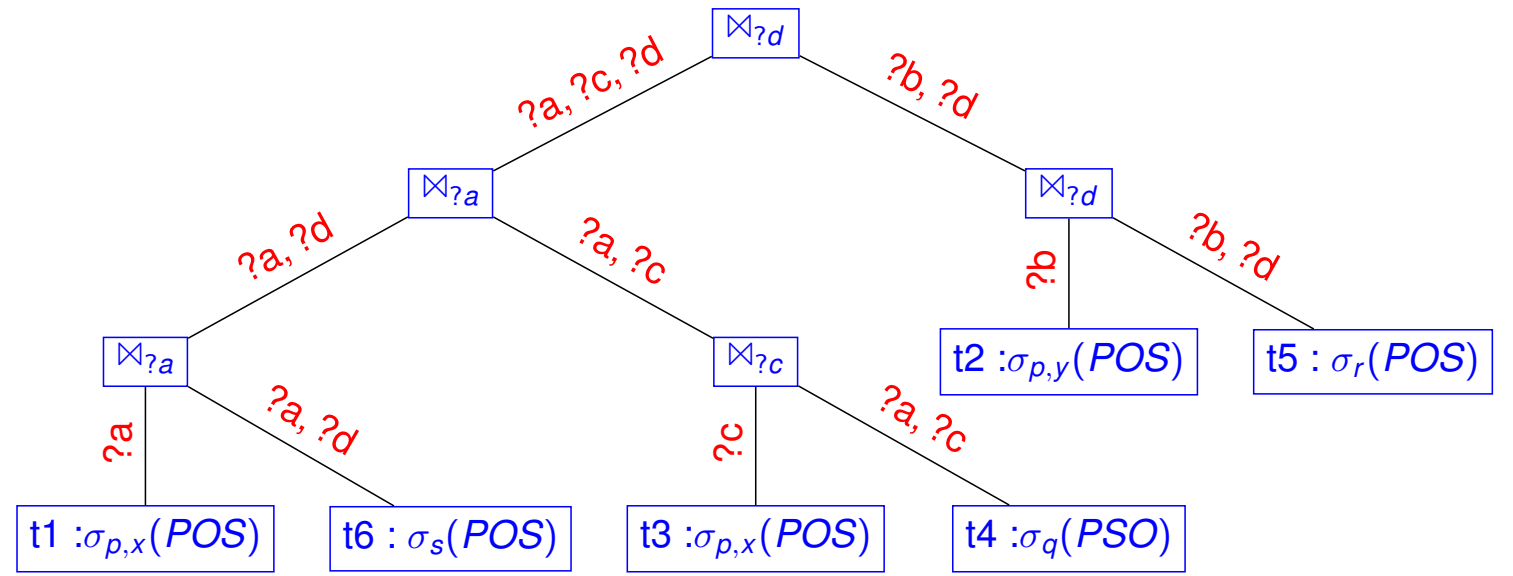
RDF-3X : Traduction de requêtes

Motif de graphe

t1 : (?a :p :x)
t2 : (?b :p :y)
t3 : (?c :p :x)
t4 : (?c :q ?a)
t5 : (?d :r ?b)
t6 : (?d :s ?a)

Graphe de jointure (dirigé et acyclique)





$$((\overbrace{\sigma_{p,x}(POS)}^{t1} \bowtie_{?a} \overbrace{\sigma_s(POS)}^{t6}) \bowtie_{?a} (\overbrace{\sigma_{p,x}(POS)}^{t3} \bowtie_{?c} \overbrace{\sigma_q(PSO)}^{t4})) \bowtie_{?d} (\overbrace{\sigma_{p,y}(POS)}^{t2} \bowtie_{?b} \overbrace{\sigma_r(POS)}^{t5})$$

2 - Optimisation de requêtes SPARQL

- Ordonnancement de jointures
- Indexation de données
- SPARQL et Inférence

Évaluation SPARQL avec sémantique RDF/S

Problème

- Comment prendre en compte la sémantique RDF/S (voir modèle/entailment RDF/S) dans l'évaluation des requêtes ?

Solutions

- Réécriture de requête : injecter la sémantique dans la requête en faisant un réécriture ;
  - remplacer rdfs :subClassof par rdfs :subClassof+
  - remplacer ?a :p ?b par ?a ?q ?b . ?q rdfs :subProperty+ :p
- Matérialisation de la sémantique RDFS dans les données (entailment rules) :
  - statique : avant l'évaluation des requêtes (forward chaining)
  - dynamique : pendant l'évaluation des requête (backward chaining)

Évaluation avec inférence : +/-

Réécriture sémantique

+ utilisation espace mémoire/disque « optimale »
+ pas besoin de moteur d'inférence sur les données
- les algorithmes de réécritures sont complexes (inférence sémantique)
- les requêtes générées sont complexes

Matérialisation dynamique

+ utilisation espace mémoire/disque « optimale »
- évaluation des règles « déconnectée » de l'évaluation de la requête
- coût difficile à estimer et à optimiser

Matérialisation statique

+ simplicité : utilisation « directe » de l'algèbre SPARQL
- utilisation importante de espace mémoire/disque
- génération de très grands graphes

$\Rightarrow$ solution adoptée par la plupart des moteurs SPARQL

TME : Jena TDB Optimizer

Trois stratégies :

- none.opt : ne modifie pas le plan d'origine et exécute les motifs de triplets dans l'ordre de la requête
- fixed.opt : réordonne les motifs de triples par le nombre de variables dans le motif
- stats.opt : utilise des statistiques sur les données de l'entrepôt interrogé

Bibliographie

1. Marcelo Arenas , Jorge Pérez, Querying semantic web data with SPARQL, Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, June 13-15, 2011, Athens, Greece
2. Reinhard Pichler, Sebastian Skritek, Containment and equivalence of well-designed SPARQL, Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, June 22-27, 2014, Snowbird, Utah, USA.
3. Andrés Letelier, Jorge Pérez , Reinhard Pichler , Sebastian Skritek, Static analysis and optimization of semantic web queries, ACM Transactions on Database Systems (TODS), v.38 n.4, p.1-45, 2013.
4. Petros Tsialiamanis , Lefteris Sidirourgos , Irini Fundulaki , Vassilis Christophides , Peter Boncz, Heuristics-based query optimisation for SPARQL, Proceedings of the 15th International Conference on Extending Database Technology, March 27-30, 2012, Berlin, Germany
5. Katja Losemann, Wim Martens, The complexity of regular expressions and property paths in SPARQL, ACM Transactions on Database Systems (TODS), v.38 n.4, p.1-39, November 2013
6. Nikolay Yakovets, Parke Godfrey, and Jarek Gryz. 2016. Query Planning for Evaluating SPARQL Property Paths. In Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16). ACM, New York, NY, USA, 1875-1889.