A Node Indexing Scheme for Web Entity Retrieval

Renaud Delbru, Nickolai Toupikov, Michele Catasta, and Giovanni Tummarello

Digital Enterprise Research Institute, Galway

June 2, 2010

© Copyright 2009 Digital Enterprise Research Institute. All rights reserved.

Introduction

Web of Data

- Pages with semantic markups: RDF, RDFa, Microformats.
- Currently in the area of X00.000.000 pages with semantic markups.

How to consume these data ?

- Traditional search engines ineffective;
- Shift from text document to data entity.

Semi-structured IR: node indexing scheme

- Technique from XML IR world;
- Good compromise between query expressiveness, query processing time and update complexity.

SIREn (Semantic Information Retrieval Engine)

- Open Source implementation;
- At the core of the Sindice search engine.

From "Web" to "Web of Data"

Web

- Document
- Bag of words
- Unstructured
- Document centric

Web of Data

- Dataset - Entity
- Bag of RDF assertions
- Semi-structured
- Dataset - entity centric

Enabling **networked** knowledge.

From "Web" to "Web of Data"

Web

- Document
- Bag of words
- Unstructured
- Document centric

Web of Data

- Dataset - Entity
- Bag of RDF assertions
- Semi-structured
- Dataset - entity centric

Enabling **networked** knowledge.

From "Web" to "Web of Data"

Web

- Document
- Bag of words
- Unstructured
- Document centric

Web of Data

- Dataset - Entity
- Bag of RDF assertions
- Semi-structured
- Dataset - entity centric

Enabling **networked** knowledge.

From "Web" to "Web of Data"

Web

- Document
- Bag of words
- Unstructured
- Document centric

Web of Data

- Dataset - Entity
- Bag of RDF assertions
- Semi-structured
- Dataset - entity centric

Enabling **networked** knowledge.

From "Web" to "Web of Data"

Web

- Document
- Bag of words
- Unstructured
- Document centric

Web of Data

- Dataset - Entity
- Bag of RDF assertions
- Semi-structured
- Dataset - entity centric

Enabling **networked** knowledge.

From "Web" to "Web of Data"

Web

- Document
- Bag of words
- Unstructured
- Document centric

Web of Data

- Dataset - Entity
- Bag of RDF assertions
- Semi-structured
- Dataset - entity centric

Enabling **networked** knowledge.

From "Web" to "Web of Data"

Web

- Document
- Bag of words
- Unstructured
- Document centric

Web of Data

- Dataset - Entity
- Bag of RDF assertions
- Semi-structured
- Dataset - entity centric

Enabling **networked** knowledge.

From "Web" to "Web of Data"

Web

- Document
- Bag of words
- Unstructured
- Document centric

Web of Data

- Dataset - Entity
- Bag of RDF assertions
- Semi-structured
- Dataset - entity centric

Enabling **networked** knowledge.

Entity Retrieval

Entity Retrieval

Given an entity search query, find the most relevant entities (list of entities ordered by relevance).

Entity Search Query

We aim to support three types of queries:

full-text search keyword-based queries when the data structure is unknown;

structural query complex queries specified in a star-shaped structure when the data schema is known;

semi-structural query combination of the two (where full-text search can be used on any part of the star-shaped query) when the data structure is partially known.

- Relevant subset of SPARQL
- Match well with IR

Enabling **networked** knowledge.

Entity Retrieval

Entity Retrieval

Given an entity search query, find the most relevant entities (list of entities ordered by relevance).

Entity Search Query

We aim to support three types of queries:

full-text search keyword-based queries when the data structure is unknown;

structural query complex queries specified in a star-shaped structure when the data schema is known;

semi-structural query combination of the two (where full-text search can be used on any part of the star-shaped query) when the data structure is partially known.

- Relevant subset of SPARQL
- Match well with IR

Enabling **networked** knowledge.

Entity Retrieval

Entity Retrieval

Given an entity search query, find the most relevant entities (list of entities ordered by relevance).

Entity Search Query

We aim to support three types of queries:

full-text search keyword-based queries when the data structure is unknown;

structural query complex queries specified in a star-shaped structure when the data schema is known;

semi-structural query combination of the two (where full-text search can be used on any part of the star-shaped query) when the data structure is partially known.

- Relevant subset of SPARQL
- Match well with IR

Enabling **networked** knowledge.

Entity Retrieval

Entity Retrieval

Given an entity search query, find the most relevant entities (list of entities ordered by relevance).

Entity Search Query

We aim to support three types of queries:

full-text search keyword-based queries when the data structure is unknown;

structural query complex queries specified in a star-shaped structure when the data schema is known;

semi-structural query combination of the two (where full-text search can be used on any part of the star-shaped query) when the data structure is partially known.

- Relevant subset of SPARQL
- Match well with IR

Enabling **networked** knowledge.

Entity Retrieval

Entity Retrieval

Given an entity search query, find the most relevant entities (list of entities ordered by relevance).

Entity Search Query

We aim to support three types of queries:

full-text search keyword-based queries when the data structure is unknown;

structural query complex queries specified in a star-shaped structure when the data schema is known;

semi-structural query combination of the two (where full-text search can be used on any part of the star-shaped query) when the data structure is partially known.

- Relevant subset of SPARQL
- Match well with IR

Enabling **networked** knowledge.

Entity Retrieval

Entity Retrieval

Given an entity search query, find the most relevant entities (list of entities ordered by relevance).

Entity Search Query

We aim to support three types of queries:

full-text search keyword-based queries when the data structure is unknown;

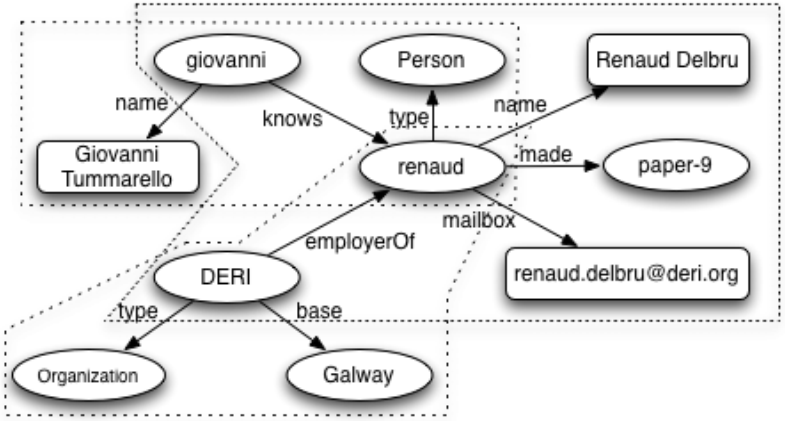
structural query complex queries specified in a star-shaped structure when the data schema is known;

semi-structural query combination of the two (where full-text search can be used on any part of the star-shaped query) when the data structure is partially known.

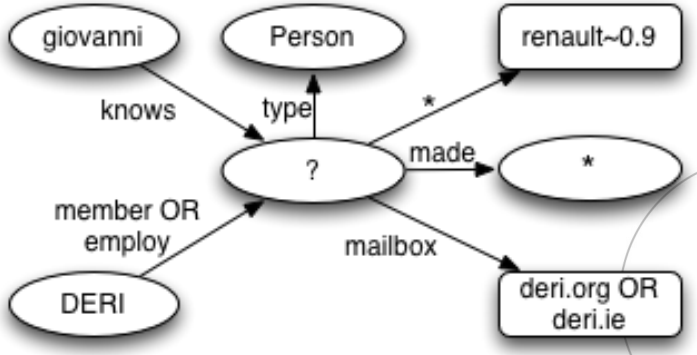
- Relevant subset of SPARQL
- Match well with IR

Enabling **networked** knowledge.

Entity Retrieval: Star Query



(a) Visual representation of an RDF graph.



(b) Star-shaped query

Figure: Oval nodes represent resources and rectangular ones represent literals.

Enabling **networked** knowledge.

Outline: Tree Model

Tree Model
- Conceptual Model
- Node-Labelled Tree: Model
- Node-Labelled Tree: Example

Enabling **networked** knowledge.

Conceptual Model

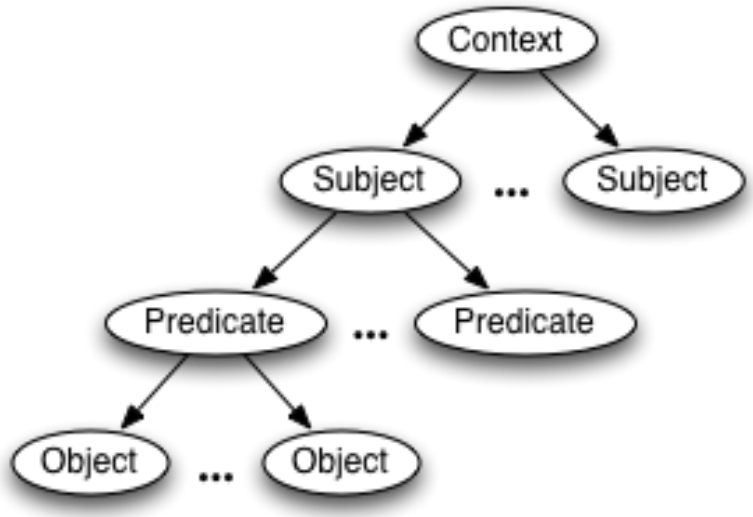


Figure: Conceptual representation of the node-labelled tree model

Enabling **networked** knowledge.

Node-Labelled Tree: Model

Origin: Semi-structured information retrieval, more recently XML retrieval.

Goal: Encode relationship between nodes

Operators: *Parent-Child* and *Ancestor-Descendant* (as in XPath)

Requirement: Assign unique identifiers (node labels) that encode relationships between the nodes

Solution: Node labelling scheme (e.g., Dewey Encoding)

Enabling **networked** knowledge.

Node-Labelled Tree: Example

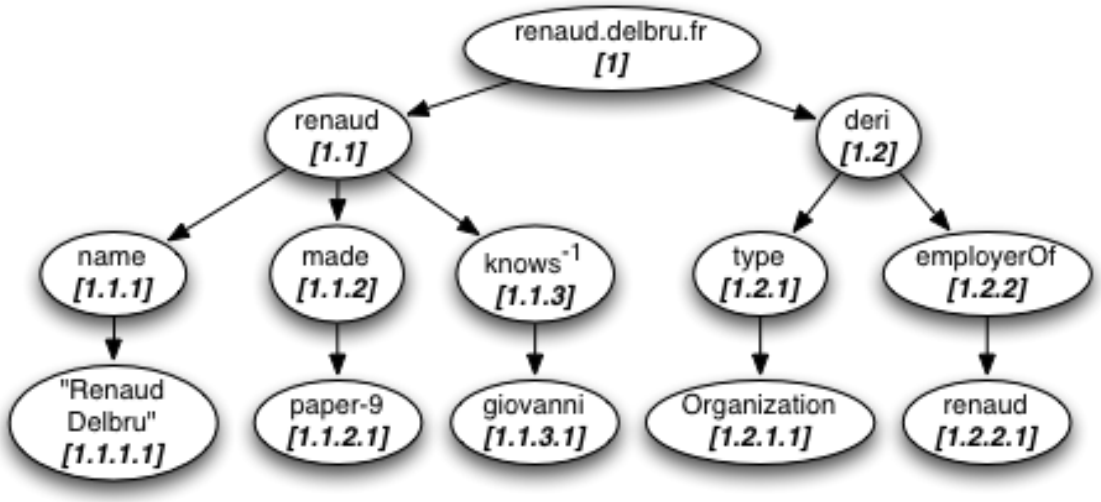


Figure: Node-labelled tree using Dewey's encoding

Enabling **networked** knowledge.

Outline: Query Model

Query Model
- Operator Overview
- Structure Operators

Enabling **networked** knowledge.

Operator Overview

Content Operators

- Orthogonal to the structure operators;
- Atomic search element: keyword;
- Boolean operators (intersection, union, difference), proximity operators (phrase, etc.), ...
- Allow to compose complex keyword queries to retrieve nodes.

Structure Operators

- Atomic search element: node;
- Allow to compose path queries to retrieve quads;
- Allow combination of quads.

Enabling **networked** knowledge.

Operator Overview

Content Operators

- Orthogonal to the structure operators;
- Atomic search element: keyword;
- Boolean operators (intersection, union, difference), proximity operators (phrase, etc.), ...
- Allow to compose complex keyword queries to retrieve nodes.

Structure Operators

- Atomic search element: node;
- Allow to compose path queries to retrieve quads;
- Allow combination of quads.

Enabling **networked** knowledge.

Structure Operator: Ancestor-Descendant

Ancestor-Descendant: A//D

A node A is the ancestor of a node D if it exists a path between A and D.

```
SELECT DISTINCT ?s WHERE {
 GRAPH <renaud.delbru.fr> {
  ?s ?p <person>
}}
```

→ `renaud.delbru.fr // person`

```
SELECT DISTINCT ?g WHERE {
 GRAPH ?g {
  <renaud> ?p <giovanni>
}}
```

→ `renaud // giovanni`

Enabling **networked** knowledge.

Structure Operator: Ancestor-Descendant

Ancestor-Descendant: A//D

A node A is the ancestor of a node D if it exists a path between A and D.

```
SELECT DISTINCT ?s WHERE {
 GRAPH <renaud.delbru.fr> {
  ?s ?p <person>
}}
```

→

```
renaud.delbru.fr // person
```

```
SELECT DISTINCT ?g WHERE {
 GRAPH ?g {
  <renaud> ?p <giovanni>
}}
```

→

```
renaud // giovanni
```


Enabling **networked** knowledge.

Structure Operator: Ancestor-Descendant

Ancestor-Descendant: A//D

A node A is the ancestor of a node D if it exists a path between A and D.

```
SELECT DISTINCT ?s WHERE {
 GRAPH <renaud.delbru.fr> {
  ?s ?p <person>
}}
```

→

```
renaud.delbru.fr // person
```

```
SELECT DISTINCT ?g WHERE {
 GRAPH ?g {
  <renaud> ?p <giovanni>
}}
```

→

```
renaud // giovanni
```


Enabling **networked** knowledge.

Structure Operator: Parent-Child

Parent-Child: P/C

A node P is the parent of a node C if P is an ancestor of C and C is exactly one level above P.

```
SELECT DISTINCT ?g ?s WHERE {
 GRAPH ?g {
  ?s <knows> <giovanni> .
}}
```

→

```
knows / giovanni
```


Enabling **networked** knowledge.

Structure Operator: Parent-Child

Parent-Child: P/C

A node P is the parent of a node C if P is an ancestor of C and C is exactly one level above P.

```
SELECT DISTINCT ?g ?s WHERE {
 GRAPH ?g {
  ?s <knows> <giovanni> .
}}
```

→

```
knows / giovanni
```


Enabling **networked** knowledge.

Structure Operator: Set Operators

Set Operators

Union ($\cup$), set-difference ($\setminus$) and intersection ($\cap$) over set of nodes.

```
SELECT DISTINCT ?g ?s WHERE {
 GRAPH ?g {
  ?s <name> "renaud" .
  { ?s <made> <paper-2> }
  UNION
  { ?s <made> <paper-12> }
  OPTIONAL
  { ?s <made> ?x .
    FILTER (?x = paper-1)
  } FILTER (!bound(?x))
}}
```

$\rightarrow$

```
name / "renaud"
AND
made / paper-2 OR paper-12
NOT
made / paper-1
```


Enabling **networked** knowledge.

Outline: Implementation

Implementation

- Inverted Index
- Incremental Update of the Inverted Lists
- Query Processing

Enabling **networked** knowledge.

Inverted Index

Inverted Index: Collection of n inverted lists $I_{t_0}, I_{t_1}, \ldots, I_{t_n}$

I_t: List of postings for the term t

Posting: context, entity, predicate, object and position

Example (Inverted Index)

Term	Postings
name	[1.1.1], [1.2.1], [3.2.1], ...
renaud	[1.1.1.1], [5.3.2.1], ...

Enabling **networked** knowledge.

Inverted Index

Inverted Index: Collection of n inverted lists $I_{t_0}, I_{t_1}, \ldots, I_{t_n}$

I_t: List of postings for the term t

Posting: context, entity, predicate, object and position

Example (Inverted Index)

Term	Postings
name	[1.1.1], [1.2.1], [3.2.1], ...
renaud	[1.1.1.1], [5.3.2.1], ...

Enabling **networked** knowledge.

Incremental Update

Insertion of one quad

1. accessing the postings list of the term p and o;
2. appending a posting to each list.

Complexity of Insertion

$O(log(n) + 1)$

Enabling **networked** knowledge.

Incremental Update

Insertion of one quad

1. accessing the postings list of the term p and o;
2. appending a posting to each list.

Complexity of Insertion

$O(log(n) + 1)$

Enabling **networked** knowledge.

Incremental Update

Insertion of one quad

1. accessing the postings list of the term p and o;
2. appending a posting to each list.

Complexity of Insertion

$O(log(n) + 1)$

Enabling **networked** knowledge.

Incremental Update

Insertion of one quad

1. accessing the postings list of the term p and o;
2. appending a posting to each list.

Complexity of Insertion

$O(log(n) + 1)$

Enabling **networked** knowledge.

Query Processing

Intersection

- Quad pattern search;
- Combination of quad patterns.

Intersection of Posting Lists

1. Retrieve the posting list of of each term;
2. Walk through the lists simultaneously. At each step, we
   1. Compare the context and subject identifiers;
   2. Compare predicate and object identifiers. If they are the same, we put the pair `cid, sid` in the result list and advance the pointers to their next position in each postings list.

Complexity

- Worst-case: linear to the total number of posting entries
- Average-case: sub-linear time with an internal index

Enabling **networked** knowledge.

Query Processing

Intersection

- Quad pattern search;
- Combination of quad patterns.

Intersection of Posting Lists

1. Retrieve the posting list of of each term;
2. Walk through the lists simultaneously. At each step, we
   1. Compare the context and subject identifiers;
   2. Compare predicate and object identifiers. If they are the same, we put the pair `cid, sid` in the result list and advance the pointers to their next position in each postings list.

Complexity

- Worst-case: linear to the total number of posting entries
- Average-case: sub-linear time with an internal index

Enabling **networked** knowledge.

Query Processing

Intersection

- Quad pattern search;
- Combination of quad patterns.

Intersection of Posting Lists

1. Retrieve the posting list of of each term;
2. Walk through the lists simultaneously. At each step, we
   1. Compare the context and subject identifiers;
   2. Compare predicate and object identifiers. If they are the same, we put the pair `cid, sid` in the result list and advance the pointers to their next position in each postings list.

Complexity

- Worst-case: linear to the total number of posting entries
- Average-case: sub-linear time with an internal index

Enabling **networked** knowledge.

Query Processing

Intersection

- Quad pattern search;
- Combination of quad patterns.

Intersection of Posting Lists

1. Retrieve the posting list of of each term;
2. Walk through the lists simultaneously. At each step, we
   1. Compare the context and subject identifiers;
   2. Compare predicate and object identifiers. If they are the same, we put the pair `cid, sid` in the result list and advance the pointers to their next position in each postings list.

Complexity

- Worst-case: linear to the total number of posting entries
- Average-case: sub-linear time with an internal index

Enabling **networked** knowledge.

Query Processing

Intersection

- Quad pattern search;
- Combination of quad patterns.

Intersection of Posting Lists

1. Retrieve the posting list of of each term;
2. Walk through the lists simultaneously. At each step, we
   1. Compare the context and subject identifiers;
   2. Compare predicate and object identifiers. If they are the same, we put the pair `cid, sid` in the result list and advance the pointers to their next position in each postings list.

Complexity

- Worst-case: linear to the total number of posting entries
- Average-case: sub-linear time with an internal index

Enabling **networked** knowledge.

Query Processing

Intersection

- Quad pattern search;
- Combination of quad patterns.

Intersection of Posting Lists

1. Retrieve the posting list of of each term;
2. Walk through the lists simultaneously. At each step, we
   1. Compare the context and subject identifiers;
   2. Compare predicate and object identifiers. If they are the same, we put the pair `cid, sid` in the result list and advance the pointers to their next position in each postings list.

Complexity

- Worst-case: linear to the total number of posting entries
- Average-case: sub-linear time with an internal index

Enabling **networked** knowledge.

Outline: Comparison

Comparison

- Entity Retrieval: Candidate Systems
- Field-based indexing scheme
- Comparison Table

Enabling **networked** knowledge.

Entity Retrieval: Candidate Systems

- Node-labelled indexing scheme (SIREn)
- Field-based indexing scheme (Lucene)
- Quad table indexing scheme (RDF Database)
- Hybrid between field-based and quad table (Semplore)

Enabling **networked** knowledge.

Field-based indexing scheme

Example (Field-Based Inverted Index)

Term	Postings
name:renaud	[1], [2], [3], ...
name:delbru	[1], [5], ...

Comparison among Candidates

Criteria	Node Index	Field Index	Quad Table	Semplore
Dictionary Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n))$	$O(log(n))$
Quad Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n))$
Join in Quad Lookup	Yes	No	No	No
Star Query Evaluation	Sub-Linear	Sub-Linear	$O(n)$	$O(n*log(n))$
Update Cost	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n)+log(l))$
Multiple Indices	No	No	Yes	Yes
Query Expressiveness	Star	Star	Graph	Tree
Full-Text	Yes	Yes (on literals)	No	Yes (on literals)
Multi-Valued Support	Yes	No	Yes	No
False positive	No	Yes	No	Yes

Enabling **networked** knowledge.

Comparison among Candidates

Criteria	Node Index	Field Index	Quad Table	Semplore
Dictionary Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n))$	$O(log(n))$
Quad Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n))$
Join in Quad Lookup	Yes	No	No	No
Star Query Evaluation	Sub-Linear	Sub-Linear	$O(n)$	$O(n*log(n))$
Update Cost	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n)+log(l))$
Multiple Indices	No	No	Yes	Yes
Query Expressiveness	Star	Star	Graph	Tree
Full-Text	Yes	Yes (on literals)	No	Yes (on literals)
Multi-Valued Support	Yes	No	Yes	No
False positive	No	Yes	No	Yes

Enabling **networked** knowledge.

Comparison among Candidates

Criteria	Node Index	Field Index	Quad Table	Semplore
Dictionary Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n))$	$O(log(n))$
Quad Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n))$
Join in Quad Lookup	Yes	No	No	No
Star Query Evaluation	Sub-Linear	Sub-Linear	$O(n)$	$O(n*log(n))$
Update Cost	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n)+log(l))$
Multiple Indices	No	No	Yes	Yes
Query Expressiveness	Star	Star	Graph	Tree
Full-Text	Yes	Yes (on literals)	No	Yes (on literals)
Multi-Valued Support	Yes	No	Yes	No
False positive	No	Yes	No	Yes

Enabling **networked** knowledge.

Comparison among Candidates

Criteria	Node Index	Field Index	Quad Table	Semplore
Dictionary Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n))$	$O(log(n))$
Quad Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n))$
Join in Quad Lookup	Yes	No	No	No
Star Query Evaluation	Sub-Linear	Sub-Linear	$O(n)$	$O(n*log(n))$
Update Cost	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n)+log(l))$
Multiple Indices	No	No	Yes	Yes
Query Expressiveness	Star	Star	Graph	Tree
Full-Text	Yes	Yes (on literals)	No	Yes (on literals)
Multi-Valued Support	Yes	No	Yes	No
False positive	No	Yes	No	Yes

Enabling **networked** knowledge.

Comparison among Candidates

Criteria	Node Index	Field Index	Quad Table	Semplore
Dictionary Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n))$	$O(log(n))$
Quad Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n))$
Join in Quad Lookup	Yes	No	No	No
Star Query Evaluation	Sub-Linear	Sub-Linear	$O(n)$	$O(n*log(n))$
Update Cost	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n)+log(l))$
Multiple Indices	No	No	Yes	Yes
Query Expressiveness	Star	Star	Graph	Tree
Full-Text	Yes	Yes (on literals)	No	Yes (on literals)
Multi-Valued Support	Yes	No	Yes	No
False positive	No	Yes	No	Yes

Enabling **networked** knowledge.

Comparison among Candidates

Criteria	Node Index	Field Index	Quad Table	Semplore
Dictionary Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n))$	$O(log(n))$
Quad Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n))$
Join in Quad Lookup	Yes	No	No	No
Star Query Evaluation	Sub-Linear	Sub-Linear	$O(n)$	$O(n*log(n))$
Update Cost	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n)+log(l))$
Multiple Indices	No	No	Yes	Yes
Query Expressiveness	Star	Star	Graph	Tree
Full-Text	Yes	Yes (on literals)	No	Yes (on literals)
Multi-Valued Support	Yes	No	Yes	No
False positive	No	Yes	No	Yes

Enabling **networked** knowledge.

Comparison among Candidates

Criteria	Node Index	Field Index	Quad Table	Semplore
Dictionary Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n))$	$O(log(n))$
Quad Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n))$
Join in Quad Lookup	Yes	No	No	No
Star Query Evaluation	Sub-Linear	Sub-Linear	$O(n)$	$O(n*log(n))$
Update Cost	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n)+log(l))$
Multiple Indices	No	No	Yes	Yes
Query Expressiveness	Star	Star	Graph	Tree
Full-Text	Yes	Yes (on literals)	No	Yes (on literals)
Multi-Valued Support	Yes	No	Yes	No
False positive	No	Yes	No	Yes

Enabling **networked** knowledge.

Comparison among Candidates

Criteria	Node Index	Field Index	Quad Table	Semplore
Dictionary Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n))$	$O(log(n))$
Quad Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n))$
Join in Quad Lookup	Yes	No	No	No
Star Query Evaluation	Sub-Linear	Sub-Linear	$O(n)$	$O(n*log(n))$
Update Cost	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n)+log(l))$
Multiple Indices	No	No	Yes	Yes
Query Expressiveness	Star	Star	Graph	Tree
Full-Text	Yes	Yes (on literals)	No	Yes (on literals)
Multi-Valued Support	Yes	No	Yes	No
False positive	No	Yes	No	Yes

Enabling **networked** knowledge.

Comparison among Candidates

Criteria	Node Index	Field Index	Quad Table	Semplore
Dictionary Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n))$	$O(log(n))$
Quad Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n))$
Join in Quad Lookup	Yes	No	No	No
Star Query Evaluation	Sub-Linear	Sub-Linear	$O(n)$	$O(n*log(n))$
Update Cost	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n)+log(l))$
Multiple Indices	No	No	Yes	Yes
Query Expressiveness	Star	Star	Graph	Tree
Full-Text	Yes	Yes (on literals)	No	Yes (on literals)
Multi-Valued Support	Yes	No	Yes	No
False positive	No	Yes	No	Yes

Enabling **networked** knowledge.

Comparison among Candidates

Criteria	Node Index	Field Index	Quad Table	Semplore
Dictionary Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n))$	$O(log(n))$
Quad Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n))$
Join in Quad Lookup	Yes	No	No	No
Star Query Evaluation	Sub-Linear	Sub-Linear	$O(n)$	$O(n*log(n))$
Update Cost	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n)+log(l))$
Multiple Indices	No	No	Yes	Yes
Query Expressiveness	Star	Star	Graph	Tree
Full-Text	Yes	Yes (on literals)	No	Yes (on literals)
Multi-Valued Support	Yes	No	Yes	No
False positive	No	Yes	No	Yes

Enabling **networked** knowledge.

Outline: Experimental Results

Experimental Results
- Experimental Design
- Index Size
- Insertion Time
- Query Execution
- Scalability Test

Enabling **networked** knowledge.

Experimental Design

Datasets

Barton: 50M triples (approximately 6GB)

Real-World: Random sampling of Sindice, 10M triples (approximately 2GB)

Candidate Systems

- SIREn
- OpenRDF Sesame
- RDF-3X

Comparison Criteria

- Index size
- Incremental index creation time
- Query processing time

Enabling **networked** knowledge.

Index Size and Indexing Time

(a) Index size in MB per dataset and system

	SIREn	Sesame	RDF-3X
Barton	789	3400	3076
Real-World	296	799	1138

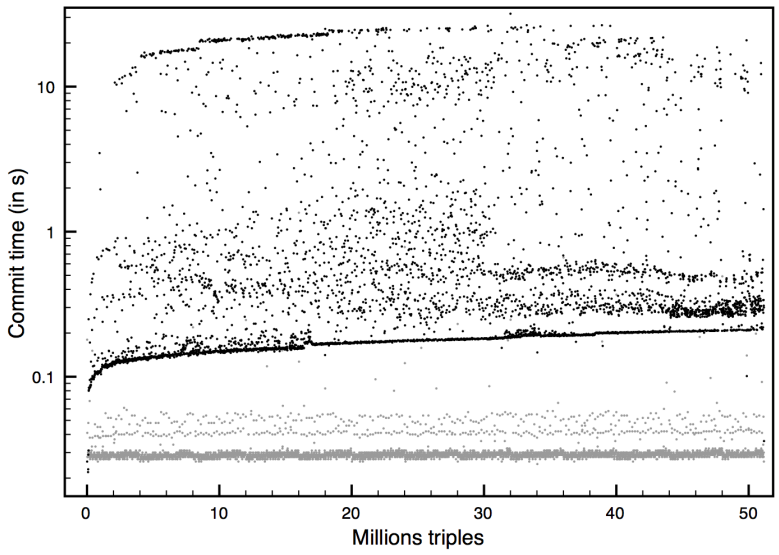
(b) Indexing time in minutes per dataset and system

	SIREn10	SIREn100	Sesame	RDF-3X
Barton	3	1.5	266	11
Real-World	1	0.5	47	3.6

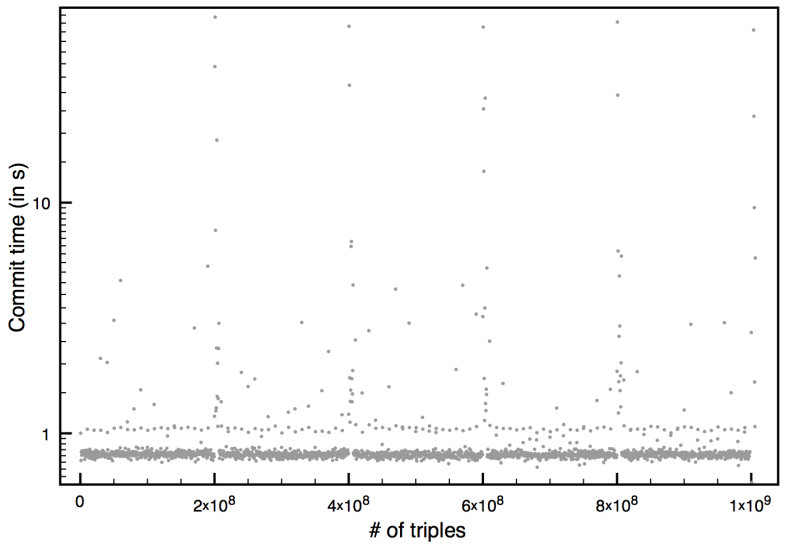
Table: Report on index size and indexing time

Enabling **networked** knowledge.

Insertion Time



(a) Plots showing the commit time every 10.000 triples during the index creation on Barton



(b) Plots showing the commit time every 500.000 triples during the index creation over one billion triples

Figure: Dark dots are Sesame commit time records while gray dots are SIREn commit time records

Enabling **networked** knowledge.

Query Execution: Benchmark Design

Setup

- 2 Quad-core, 8GB, SATA disk.
- Cold cache
- 50 measurements per query
- Arithmetic mean reported

Query Types

A*	Term lookup
B*	Triple pattern lookup
C*,D*,E*	Intersection, union, exclusion

Enabling **networked** knowledge.

Query Execution: Time

(a) Barton dataset

	A1	A2	B1	C1	C2	D1	D2	E
RDF-3X	16.12	0.12	1.38	1.16	0.38	0.23	0.14	X
SIREn	2.79	0.02	1.33	1.71	0.95	0.36	0.03	0.96

(b) Real-World dataset

	A1	A2	B1	B2	C1	C2	D1	E
RDF-3X	0.29	0.12	0.17	0.18	0.21	0.13	0.28	X
SIREn	0.23	0.03	0.04	0.05	0.09	0.08	0.16	0.53

Table: Querying time in seconds

Enabling **networked** knowledge.

Query Execution: Scalability Test

Dataset

- Billion Triple Challenge dataset replicated 10 times
- 1 Terabyte of data

(a) 10 Billion Triples dataset

	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Time (s)	0.75	1.3	1.4	0.5	1.5	1.6	4
Hits	7552	9344	3.5M	57K	448	8.2M	20.7M

Table: Querying time in seconds

Enabling **networked** knowledge.

Conclusion

SIREn (Semantic Information Retrieval Engine)

- Open source implementation: `http://siren.sindice.com`
- At the core of the Sindice search engine:
  - 100M semi-structured documents (6.5 Billion triples)
  - On a single index (Quad-core, 8GB)
  - Replicated: one master for updates, one slave for queries

About ranking

See "Hierarchical Link Analysis for Web Entity Retrieval"

Future Work

- Query expressiveness: investigate entity path queries
- Index optimisation: inverted list compression

Enabling **networked** knowledge.