Keyword-based and Top-K Querying

Prabhakar Dixit
IMSE

Efficient Execution of Top-K SPARQL queries

Outline

Introduction

- Top k-queries
- Need for optimization

Approach

- A rank-aware SPARQL algebra
- A rank aware execution model
- Planning strategies

Top k-queries

A query that returns

- A limited number of results "k"
- Ordered by a scoring function that combines several criteria

Rankings


THE WORLD UNIVERSITY RANKINGS
POWERED BY THOMSON REUTERS
HOME
WORLD UNIVERSITY RANKINGS
WORLD REPUTATION RANKINGS
100
YEAR
2012-13
RANKINGS
ANALYSIS
METHODOLOGY
ASIA UNIVERSITY RANKINGS
The Times Higher Education Asia University Rankings 2013 are based on the same criteria as the THE World University Rankings, powered by Thomson Reuters. We judge world class universities across all of their core missions - teaching, research, knowledge transfer and international outlook.
The top universities rankings employ 13 carefully calibrated performance indicators to provide the most comprehensive and balanced comparisons available, which are trusted by students, academics, university leaders, industry and governments.
Asia University Rankings 2013 Top 100
Top Asian 100

Rankings


CNNMoney
FORTUNE
Home
Video
Markets
Investing
Economy
Tech
100 BEST COMPANIES TO WORK FOR
2012
FORTUNE
Full List
Near You
Top Companies
Big Pay
Best Perks
Top 100
Rank
Company
Job growth
U.S. employees
1 Google 33% 18,500
2 Boston Consulting Group 10% 1,958
3 SAS Institute 8% 6,046
4 Wegmans Food Markets 5% 41,717
5 Edward Jones 1% 36,937
6 NetApp 30% 6,887
7 Camden Property Trust -2% 1,678
originality
all-inclusiveness
variety
integration
human touch
gestalt
O
A
V
I
H

Rankings

Mercer Quality of Living Survey

From Wikipedia, the free encyclopedia

The **Mercer Quality of Living Survey** ranks 221 cities from Vienna to Baghdad on quality of life.[1]

In 2010, Vienna won the title as the highest ranked city, followed by Zurich (2), Geneva (3), jointly Vancouver (4) and Auckland (4), and Düsseldorf (5). Vienna was again the highest ranked city in 2011, followed by Zurich (2), Auckland (3), Munich (4) and jointly Düsseldorf (5) and Vancouver (5). The quality of living survey is conducted to help governments and major companies place employees on international assignments.

The survey also identifies those cities with the highest personal safety ranking based upon internal stability, crime, effectiveness of law enforcement and relationships with other countries. In this case, Luxembourg is top, followed by Bern, Helsinki and Zurich, all equally placed at number 2.

Contents [hide]

Scoring [edit]

The cities — 221 in total — were evaluated on 39 factors including political, economic, environmental, personal safety, health, education, transportation and other public service factors. Cities were compared to New York City which was given a base score of 100.

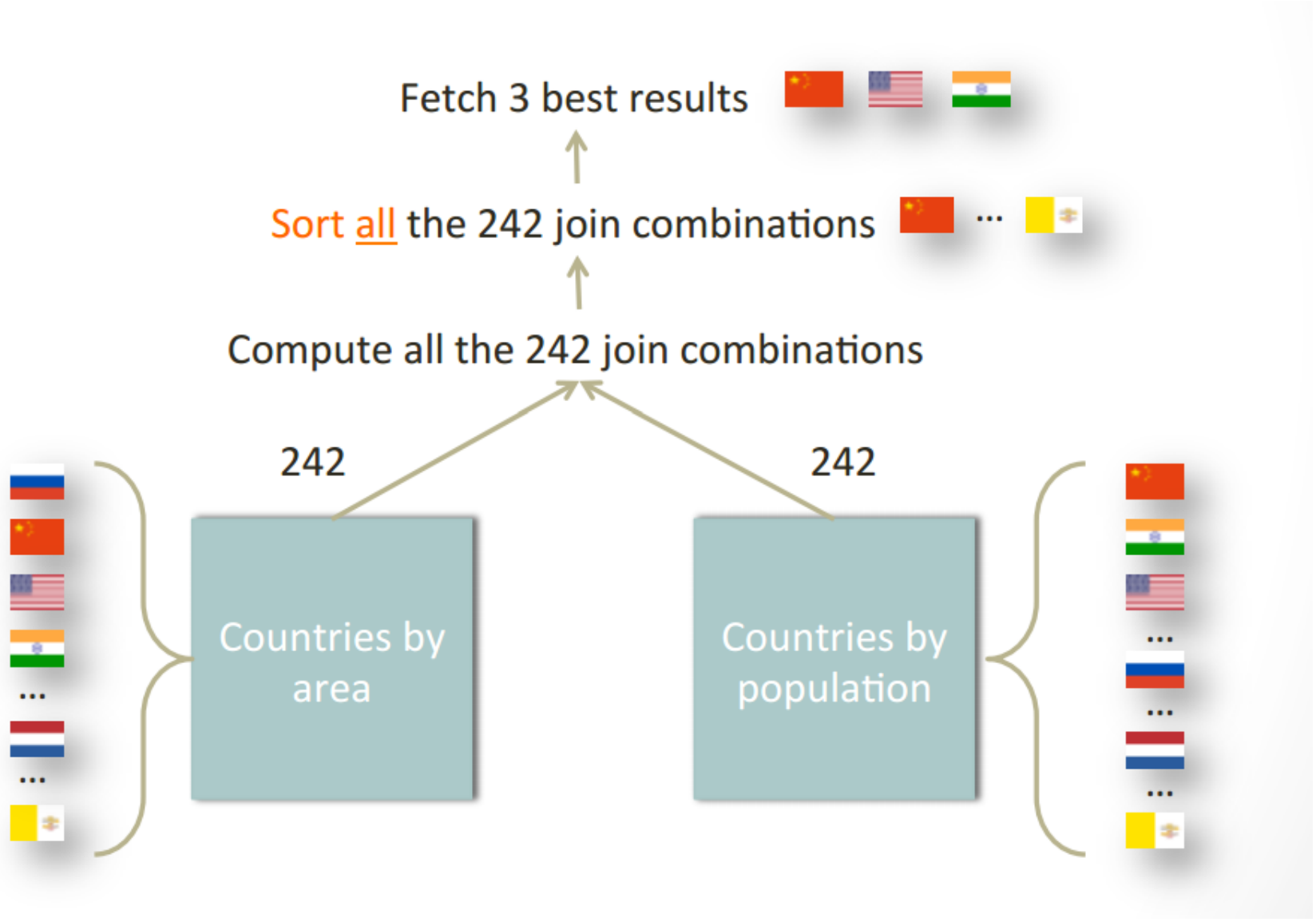
Need for Optimization

An intuitive example

Top 3 largest countries (by both area and population)

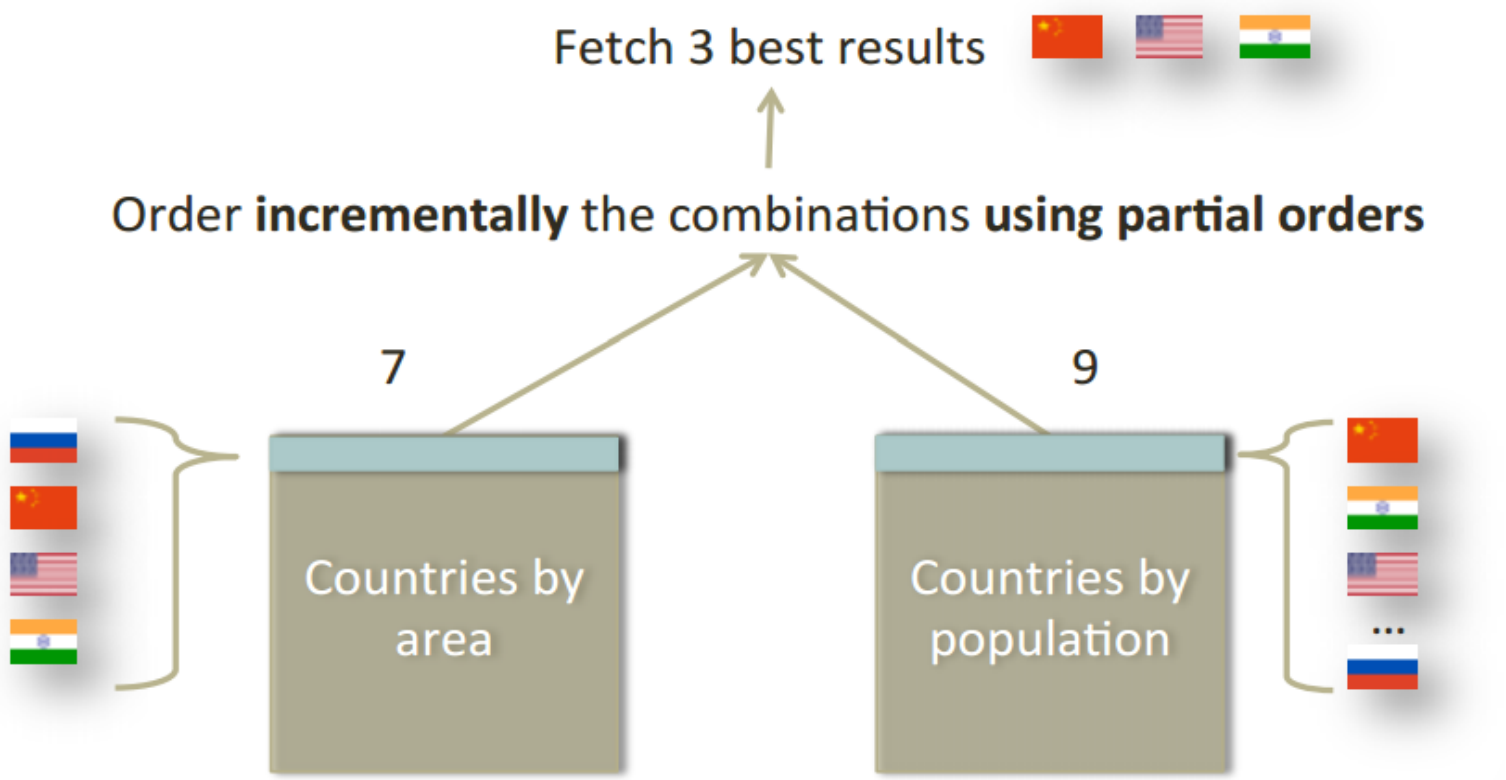


The standard way: Materialize then sort



How to make it efficient

Exploit the **available sorted access** (by area and population)



The(split and interleave) scheme

- The intuition of the previous example is formalized with the split and interleave scheme from RDBMS
  a. Split the evaluation of the scoring function into a single criteria
  b. Interleave them with other operators
  c. Use partial orders to construct *incrementally* the final order
- Standard assumptions
  a. Monotone scoring function

- Optimized for the case of fast sorted access

Top-k queries in SPARQL 1.1

- The top 10 offers ordered by the product ratings and the offer price:

```
SELECT ?product ?offer
(norm1(?avgRat1) + norm2(?avgRat2) + norm3(?price)
AS ?score)
WHERE {
   ?product hasAvgRat1 ?avgRat1 .
   ?product hasAvgRat2 ?avgRat2 .
   ?product hasName ?name .
   ?product hasOffers ?offer .
   ?offer hasPrice ?price
}
ORDER BY DESC (?score)
LIMIT 10
```

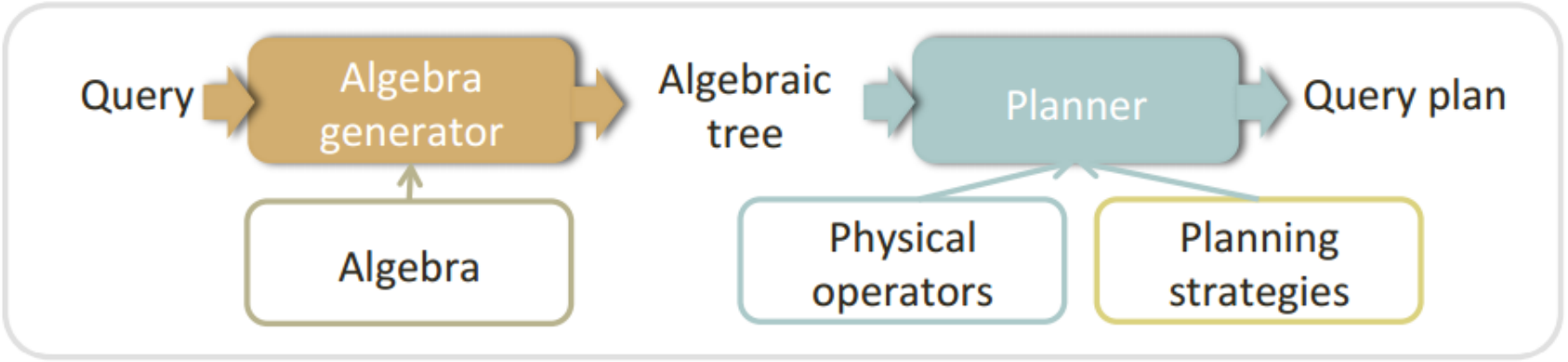
- Tens of seconds on 5M triples (scope for improvement to milliseconds)

Split & interleave in SPARQL - Related work

- A possible solution
  - Rewrite SPARQL into SQL
  - Use existing optimized RDBMS

- Disadvantages
  - Works if data is already in RDBMS

Challenges for native SPARQL split and interleave solutions



Differences with SQL and RDBMS	Proposed solution
Different algebra	STEP 1: New algebra (algebraic operators and algebraic equivalences)
Different cost of data access in native RDF triplestores (sorted access is slow)	STEP 2: New algorithms for physical operators, possibly using less sorted access
Additional optimization dimensions	STEP 3: New planning strategies

The SPARQL Rank algebraic operators

New operator rank ρ

Rank: ρ, with a ranking predicate p
- $\mu \in \rho_p(\Omega_P)$ iff $\mu \in \Omega_P$
- $\mu_1 <_{\rho_p(\Omega_P)} \mu_2$ iff $\overline{\mathcal{F}}_{P\cup\{p\}}[\mu_1] < \overline{\mathcal{F}}_{P\cup\{p\}}[\mu_2]$

Selection: σ, with a boolean condition c
- $\mu \in \sigma_c(\Omega_P)$ iff $\mu \in \Omega_P$ and μ satisfies c
- $\mu_1 <_{\sigma_c(\Omega_P)} \mu_2$ iff $\mu_1 <_{\Omega_P} \mu_2$, i.e. $\overline{\mathcal{F}}_P[\mu_1] < \overline{\mathcal{F}}_P[\mu_2]$

Union: $\cup$
- $\mu \in \Omega_{P_1} \cup \Omega'_{P_2}$ iff $\mu \in \Omega_{P_1}$ or $\mu \in \Omega'_{P_2}$
- $\mu_1 <_{\Omega_{P_1}\cup\Omega'_{P_2}} \mu_2$ iff $\overline{\mathcal{F}}_{P_1\cup P_2}[\mu_1] < \overline{\mathcal{F}}_{P_1\cup P_2}[\mu_2]$

Join: $\bowtie$
- $\mu \in \Omega_{P_1} \bowtie \Omega'_{P_2}$ iff $\mu = \mu_1 \cup \mu 2$, such that $\mu_1 \in \Omega_{P_1}$, $\mu_2 \in \Omega'_{P_2}$ and μ_1, μ_2 are compatible mappings
- $\mu_1 <_{\Omega_{P_1}\bowtie\Omega'_{P_2}} \mu_2$ iff $\overline{\mathcal{F}}_{P_1\cup P_2}[\mu_1] < \overline{\mathcal{F}}_{P_1\cup P_2}[\mu_2]$

Difference:
- $\mu \in \Omega_{P_1} - \Omega'_{P_2}$ iff $\mu \in \Omega_{P_1}$ and for all $\mu' \in \Omega'_{P_2}$, μ and μ' are not compatible
- $\mu_1 <_{\Omega_{P_1}-\Omega'_{P_2}} \mu_2$ iff $\mu_1 <_{\Omega_{P_1}} \mu_2$, i.e. $\overline{\mathcal{F}}_{P_1}[\mu_1] < \overline{\mathcal{F}}_{P_1}[\mu_2]$

Left Join: $\bowtie$
- $\mu \in \Omega_{P_1} \bowtie \Omega'_{P_2}$ iff $\mu \in (\Omega_{P_1} \bowtie \Omega'_{P_2}) \cup (\Omega_{P_1} - \Omega'_{P_2})$
- $\mu_1 <_{\Omega_{P_1}\bowtie\Omega'_{P_2}} \mu_2$ iff $\overline{\mathcal{F}}_{P_1\cup P_2}[\mu_1] < \overline{\mathcal{F}}_{P_1\cup P_2}[\mu_2]$

The Rank operator

Rank: ρ, with a ranking predicate p

- $\mu \in \rho_p(\Omega_P)$ iff $\mu \in \Omega_P$
- $\mu_1 <_{\rho_p(\Omega_P)} \mu_2$ iff $\overline{\mathcal{F}}_{P\cup\{p\}}[\mu_1] < \overline{\mathcal{F}}_{P\cup\{p\}}[\mu_2]$

	?x	?y	?p1	?p2
μ1	1	8	0.8	0.8
μ2	3	3	0.3	0.6
μ3	3	4	0.4	0.6

Ω

ρ_{p1}

	?x	?y	?p1	$\mathcal{F}_{p1}$
μ1	1	8	0.8	1.8
μ3	3	4	0.4	1.4
μ2	3	3	0.3	1.3

$\rho_{p1}(\Omega)$

The SPARQL Rank algebraic operators

Redefined standard operators

Rank: ρ, with a ranking predicate p
- $\mu \in \rho_p(\Omega_P)$ iff $\mu \in \Omega_P$
- $\mu_1 <_{\rho_p(\Omega_P)} \mu_2$ iff $\overline{\mathcal{F}}_{P\cup\{p\}}[\mu_1] < \overline{\mathcal{F}}_{P\cup\{p\}}[\mu_2]$

Selection: σ, with a boolean condition c
- $\mu \in \sigma_c(\Omega_P)$ iff $\mu \in \Omega_P$ and μ satisfies c
- $\mu_1 <_{\sigma_c(\Omega_P)} \mu_2$ iff $\mu_1 <_{\Omega_P} \mu_2$, i.e. $\overline{\mathcal{F}}_P[\mu_1] < \overline{\mathcal{F}}_P[\mu_2]$

Union: $\cup$
- $\mu \in \Omega_{P_1} \cup \Omega'_{P_2}$ iff $\mu \in \Omega_{P_1}$ or $\mu \in \Omega'_{P_2}$
- $\mu_1 <_{\Omega_{P_1}\cup\Omega'_{P_2}} \mu_2$ iff $\overline{\mathcal{F}}_{P_1\cup P_2}[\mu_1] < \overline{\mathcal{F}}_{P_1\cup P_2}[\mu_2]$

Join: $\bowtie$
- $\mu \in \Omega_{P_1} \bowtie \Omega'_{P_2}$ iff $\mu = \mu_1 \cup \mu 2$, such that $\mu_1 \in \Omega_{P_1}$, $\mu_2 \in \Omega'_{P_2}$ and μ_1, μ_2 are compatible mappings
- $\mu_1 <_{\Omega_{P_1}\bowtie\Omega'_{P_2}} \mu_2$ iff $\overline{\mathcal{F}}_{P_1\cup P_2}[\mu_1] < \overline{\mathcal{F}}_{P_1\cup P_2}[\mu_2]$

Difference:
- $\mu \in \Omega_{P_1} - \Omega'_{P_2}$ iff $\mu \in \Omega_{P_1}$ and for all $\mu' \in \Omega'_{P_2}$, μ and μ' are not compatible
- $\mu_1 <_{\Omega_{P_1}-\Omega'_{P_2}} \mu_2$ iff $\mu_1 <_{\Omega_{P_1}} \mu_2$, i.e. $\overline{\mathcal{F}}_{P_1}[\mu_1] < \overline{\mathcal{F}}_{P_1}[\mu_2]$

Left Join: $⟕$
- $\mu \in \Omega_{P_1} ⟕ \Omega'_{P_2}$ iff $\mu \in (\Omega_{P_1} \bowtie \Omega'_{P_2}) \cup (\Omega_{P_1} - \Omega'_{P_2})$
- $\mu_1 <_{\Omega_{P_1}⟕\Omega'_{P_2}} \mu_2$ iff $\overline{\mathcal{F}}_{P_1\cup P_2}[\mu_1] < \overline{\mathcal{F}}_{P_1\cup P_2}[\mu_2]$

The Join Operator

Join: $\bowtie$

- $\mu \in \Omega_{P_1} \bowtie \Omega'_{P_2}$ iff $\mu = \mu_1 \cup \mu 2$, such that $\mu_1 \in \Omega_{P_1}$, $\mu_2 \in \Omega'_{P_2}$ and μ_1, μ_2 are compatible mappings
- $\mu_1 <_{\Omega_{P_1} \bowtie \Omega'_{P_2}} \mu_2$ iff $\overline{\mathcal{F}}_{P_1 \cup P_2}[\mu_1] < \overline{\mathcal{F}}_{P_1 \cup P_2}[\mu_2]$

	?x	?y	?p1	$\mathcal{F}_{p1}$
µ1	1	8	0.8	1.8
µ3	3	4	0.4	1.4
µ2	3	3	0.3	1.3

Ω_{p1}

$\bowtie$

	?x	?z	?p2	$\mathcal{F}_{p2}$
µ4	1	9	0.8	1.8
µ5	3	0	0.6	1.6

Ω'_{p2}

	?x	?y	?z	?p1	?p2	$\mathcal{F}_{p1Up2}$
µ1 U µ4	1	8	9	0.8	0.8	1.6
µ3 U µ5	3	4	0	0.4	0.6	1.0
µ2 U µ5	3	3	0	0.3	0.6	0.9

SPARQL Rank algebraic equivalences

Split

Proposition 1: Splitting law for ρ

- $\Omega_{\{p_1,p_2,\ldots,p_n\}} = \rho_{p_1}(\rho_{p_2}(\ldots(\rho_{p_n}(\Omega))\ldots))$

Proposition 2: Commutative law for binary operators

- $\Omega_{P_1} \ \Theta \ \Omega'_{P_2} \equiv \Omega'_{P_2} \ \Theta \ \Omega_{P_1}, \ \forall \ \Theta \in \{\cup, \bowtie\}$

Proposition 3: Associative law

- $(\Omega_{P_1} \ \Theta \ \Omega'_{P_2}) \ \Theta \ \Omega''_{P_3} \equiv \Omega_{P_1} \ \Theta(\Omega'_{P_2} \ \Theta \ \Omega''_{P_3}), \ \forall \ \Theta \in \{\cup, \bowtie\}.$

Proposition 4 : Commutative laws for ρ

- $\rho_{p_1}(\rho_{p_2}(\Omega_P)) \equiv \rho_{p_2}(\rho_{p_1}(\Omega_P))$
- $\sigma_c(\rho_p(\Omega_P)) \equiv \rho_p(\sigma_c(\Omega_P))$

Proposition 5: Distributive law

- $(\Omega_{P_1} \ \bowtie \ (\Omega'_{P_2} \ \cup \ \Omega''_{P_3})) \equiv (\Omega_{P_1} \ \bowtie \ \Omega'_{P_2}) \cup (\Omega_{P_1} \ \bowtie \ \Omega''_{P_3})$

Proposition 6: Pushing ρ over binary operators

- $\rho_p(\Omega_{P_1} \bowtie \Omega'_{P_2})$
  $\equiv \rho_p(\Omega_{P_1}) \bowtie \Omega'_{P_2}$, if Ω' has variables in p
  $\equiv \rho_p(\Omega_{P_1}) \bowtie \rho_p(\Omega'_{P_2})$, if both Ω and Ω' have
- $\rho_p(\Omega_{P_1} \bowtie \Omega'_{P_2}) \equiv \rho_p(\Omega_{P_1}) \bowtie \Omega'_{P_2} \equiv \rho_p(\Omega_{P_1}) \bowtie \rho_p(\Omega'_{P_2})$
- $\rho_p(\Omega_{P_1} \cup \Omega'_{P_2}) \equiv \rho_p(\Omega_{P_1}) \cup \Omega'_{P_2} \equiv \rho_p(\Omega_{P_1}) \cup \rho_p(\Omega'_{P_2})$
- $\rho_p(\Omega_{P_1} - \Omega'_{P_2}) \equiv \rho_p(\Omega_{P_1}) - \Omega'_{P_2} \equiv \rho_p(\Omega_{P_1}) - \rho_p(\Omega'_{P_2})$

SPARQL Rank algebraic equivalences

Proposition 1: Splitting law for ρ

- $\Omega_{\{p_1,p_2,\ldots,p_n\}} = \rho_{p_1}(\rho_{p_2}(\ldots(\rho_{p_n}(\Omega))\ldots))$

- Allows the splitting of monolithic scoring function into several rank operators

SPARQL Rank algebraic equivalences

Interleave

Proposition 1: Splitting law for ρ

- $\Omega_{\{p_1,p_2,\ldots,p_n\}} = \rho_{p_1}(\rho_{p_2}(\ldots(\rho_{p_n}(\Omega))\ldots))$

Proposition 2: Commutative law for binary operators

- $\Omega_{P_1} \ \Theta \ \Omega'_{P_2} \equiv \Omega'_{P_2} \ \Theta \ \Omega_{P_1}, \forall \ \Theta \in \{\cup, \bowtie\}$

Proposition 3: Associative law

- $(\Omega_{P_1} \ \Theta \ \Omega'_{P_2}) \ \Theta \ \Omega''_{P_3} \equiv \Omega_{P_1} \ \Theta(\Omega'_{P_2} \ \Theta \ \Omega''_{P_3}), \forall \ \Theta \in \{\cup, \bowtie\}.$

Proposition 4 : Commutative laws for ρ

- $\rho_{p_1}(\rho_{p_2}(\Omega_P)) \equiv \rho_{p_2}(\rho_{p_1}(\Omega_P))$
- $\sigma_c(\rho_p(\Omega_P)) \equiv \rho_p(\sigma_c(\Omega_P))$

Proposition 5: Distributive law

- $(\Omega_{P_1} \ \bowtie \ (\Omega'_{P_2} \ \cup \ \Omega''_{P_3})) \equiv (\Omega_{P_1} \ \bowtie \ \Omega'_{P_2}) \cup (\Omega_{P_1} \ \bowtie \ \Omega''_{P_3})$

Proposition 6: Pushing ρ over binary operators

- $\rho_p(\Omega_{P_1} \bowtie \Omega'_{P_2})$
  $\equiv \rho_p(\Omega_{P_1}) \bowtie \Omega'_{P_2}$, if Ω' has variables in p
  $\equiv \rho_p(\Omega_{P_1}) \bowtie \rho_p(\Omega'_{P_2})$, if both Ω and Ω' have
- $\rho_p(\Omega_{P_1} \bowtie \Omega'_{P_2}) \equiv \rho_p(\Omega_{P_1}) \bowtie \Omega'_{P_2} \equiv \rho_p(\Omega_{P_1}) \bowtie \rho_p(\Omega'_{P_2})$
- $\rho_p(\Omega_{P_1} \cup \Omega'_{P_2}) \equiv \rho_p(\Omega_{P_1}) \cup \Omega'_{P_2} \equiv \rho_p(\Omega_{P_1}) \cup \rho_p(\Omega'_{P_2})$
- $\rho_p(\Omega_{P_1} - \Omega'_{P_2}) \equiv \rho_p(\Omega_{P_1}) - \Omega'_{P_2} \equiv \rho_p(\Omega_{P_1}) - \rho_p(\Omega'_{P_2})$

SPARQL Rank algebraic equivalences

Proposition 6: Pushing ρ over binary operators

- $\rho_p(\Omega_{P_1} \bowtie \Omega'_{P_2})$
  $\equiv \rho_p(\Omega_{P_1}) \bowtie \Omega'_{P_2}$, if Ω' has variables in p
  $\equiv \rho_p(\Omega_{P_1}) \bowtie \rho_p(\Omega'_{P_2})$, if both Ω and Ω' have
- $\rho_p(\Omega_{P_1} \bowtie \Omega'_{P_2}) \equiv \rho_p(\Omega_{P_1}) \bowtie \Omega'_{P_2} \equiv \rho_p(\Omega_{P_1}) \bowtie \rho_p(\Omega'_{P_2})$
- $\rho_p(\Omega_{P_1} \cup \Omega'_{P_2}) \equiv \rho_p(\Omega_{P_1}) \cup \Omega'_{P_2} \equiv \rho_p(\Omega_{P_1}) \cup \rho_p(\Omega'_{P_2})$
- $\rho_p(\Omega_{P_1} - \Omega'_{P_2}) \equiv \rho_p(\Omega_{P_1}) - \Omega'_{P_2} \equiv \rho_p(\Omega_{P_1}) - \rho_p(\Omega'_{P_2})$

- Allows to order incrementally the results by pushing the rank operator inside the query tree.

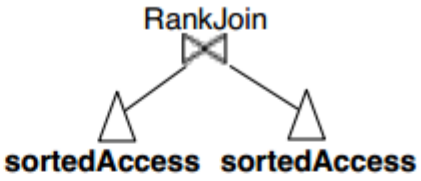
Step 2: Physical operators (top-k algorithms)

- Rank operator
  - If there is a sorted access index available on the ranking criterion, it is used. Otherwise other rank aggregation algorithms are used.

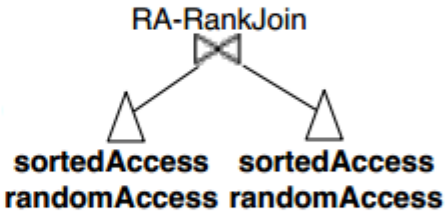
- Join operator
  - If the right operator does not influence the ranking then streaming index join is used. Otherwise a rank-join algorithm is used
- Other operators are straightforward.
  - Eg. the standard FILTER conserves the ordering of its input

Rank join algorithms

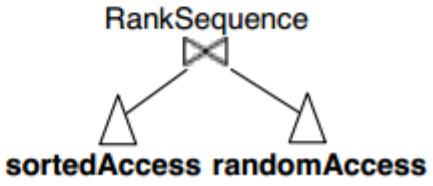
- Different algorithms based on available access in the inputs:
  - Hash Rank-Join
    - e.g. HRJN [Ilyas2004]



  - Random Access Rank-Join
    - e.g. RA-HRJN [Ilyas2004]



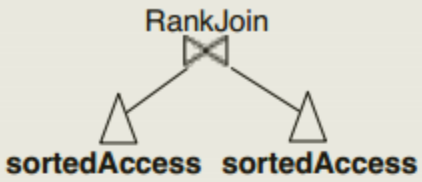
  - RankSequence (e,g, RSEQ)
    - Minimum sorted access
    - Leverages random access



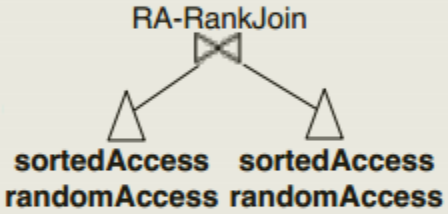
Rank join algorithms

Literature

- Hash Rank-Join
  - e.g. HRJN [Ilyas2004]

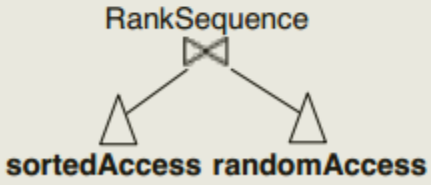


- Random Access Rank-Join
  - e.g. RA-HRJN [Ilyas2004]



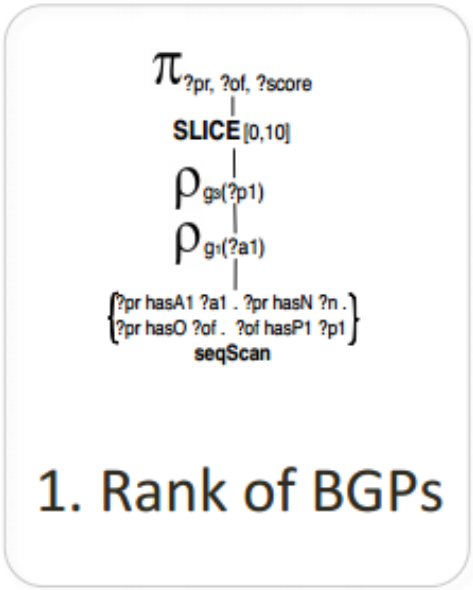
New

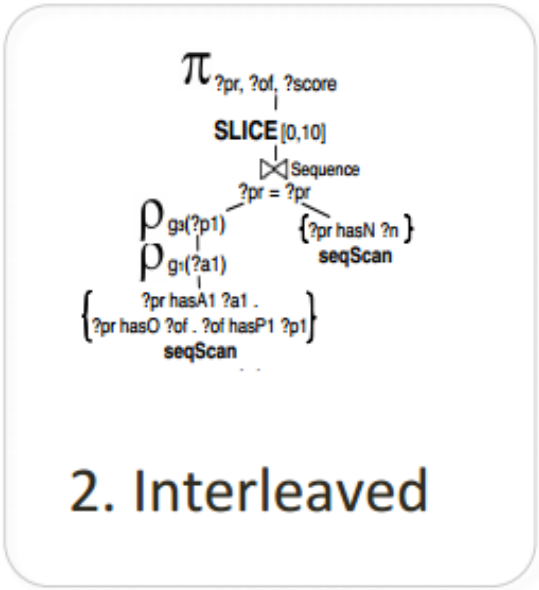
- RankSequence (e,g, RSEQ)
  - Minimum sorted access
  - Leverages random access

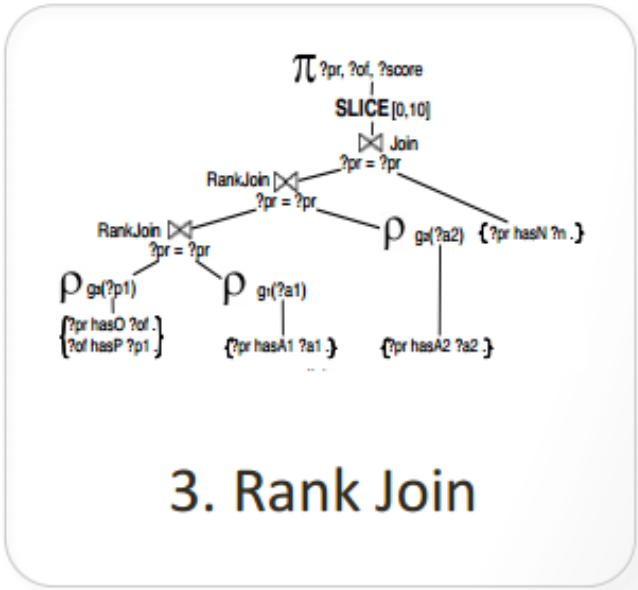


Step 3: Planning strategies

- Using the algebraic equivalences, several algebraic trees can be constructed
- The planner can use them to implement several planning strategies

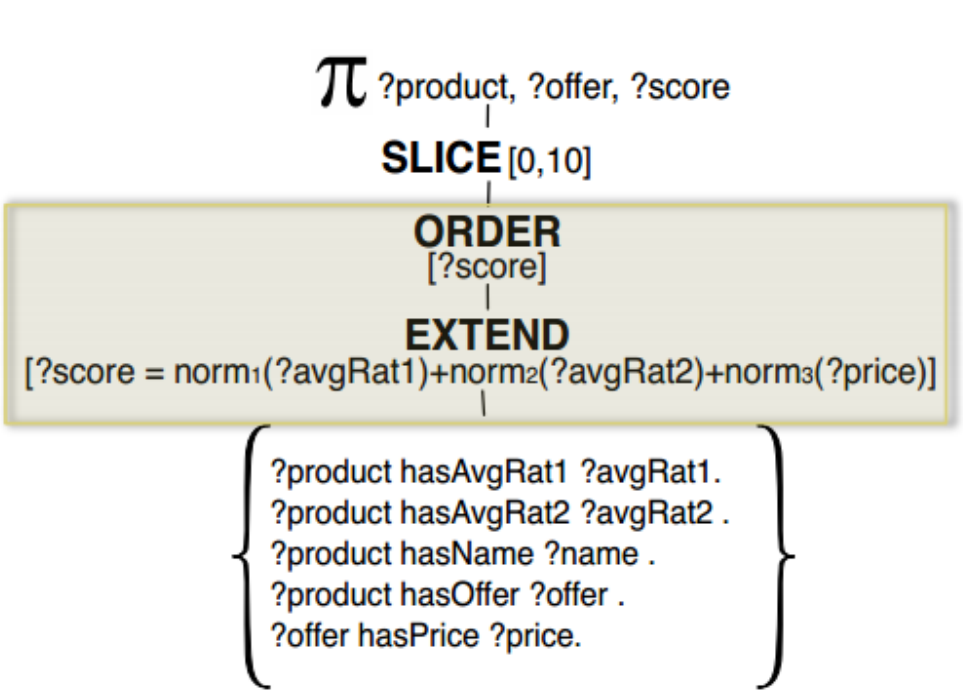




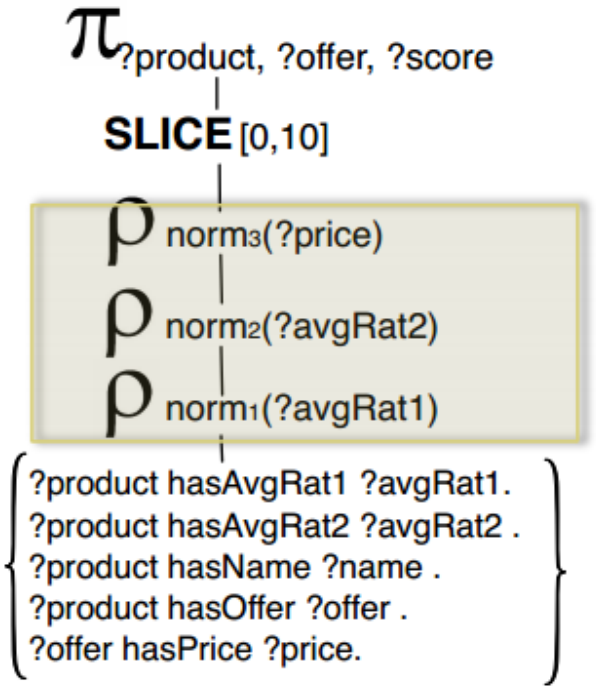


1. Rank of BGPs (ROB)

- Split the monolithic scoring function into several incremental rank operators (rho)



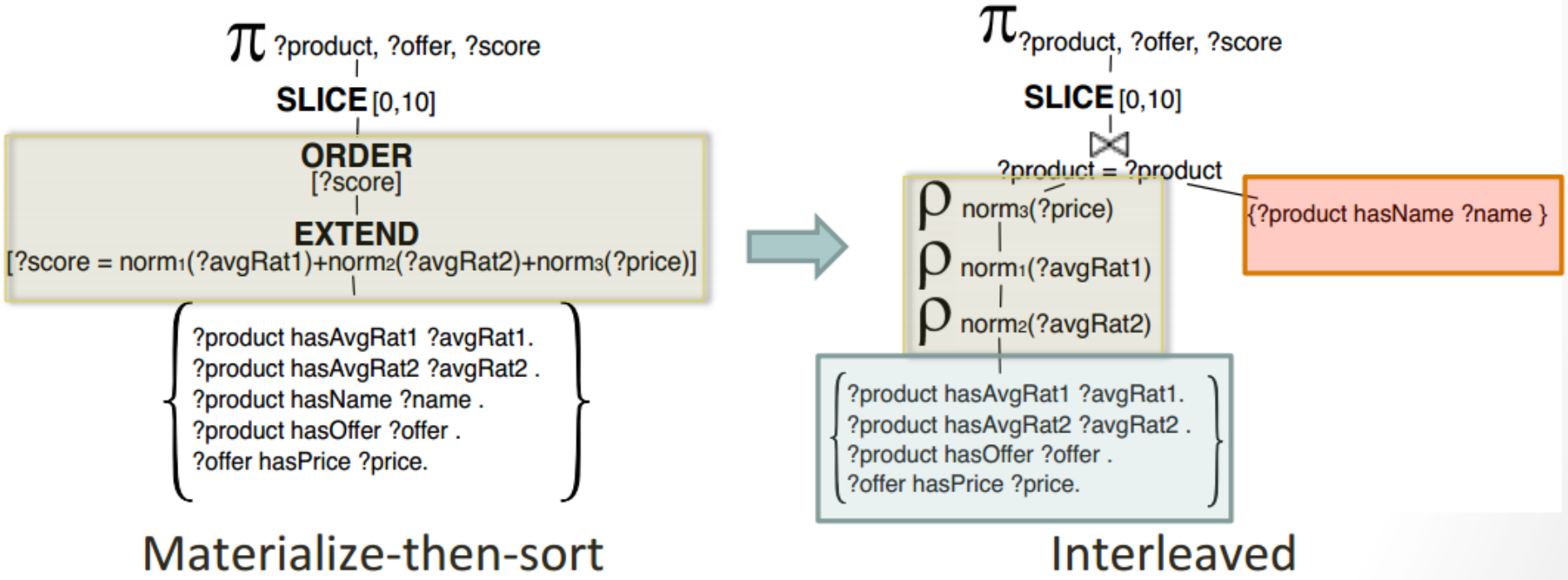
Materialize-then-sort



Rank of BGPs

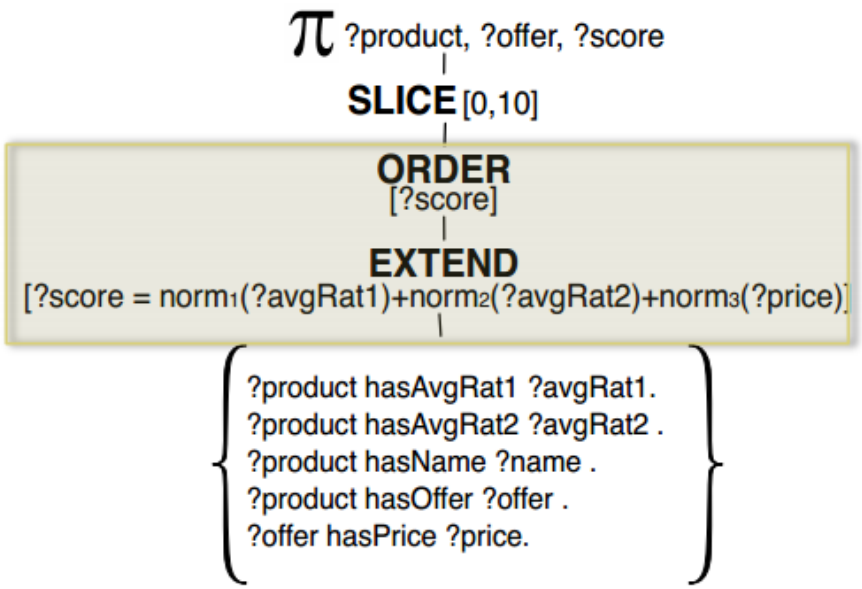
2. Interleaved (INTER)

- Separate the pattern into two groups
  - Triple patterns that influence ranking
  - Triple patterns that don't influence ranking

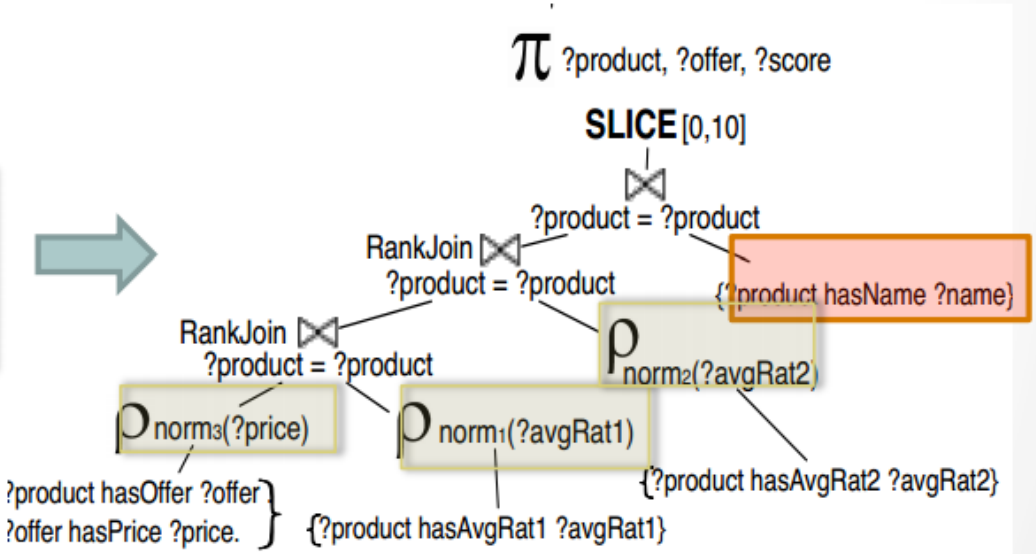


3. Rank Join (RJ)

- Split into one triple pattern for each criterion
- Most appropriate join algorithm based on available access



Materialize-then-sort



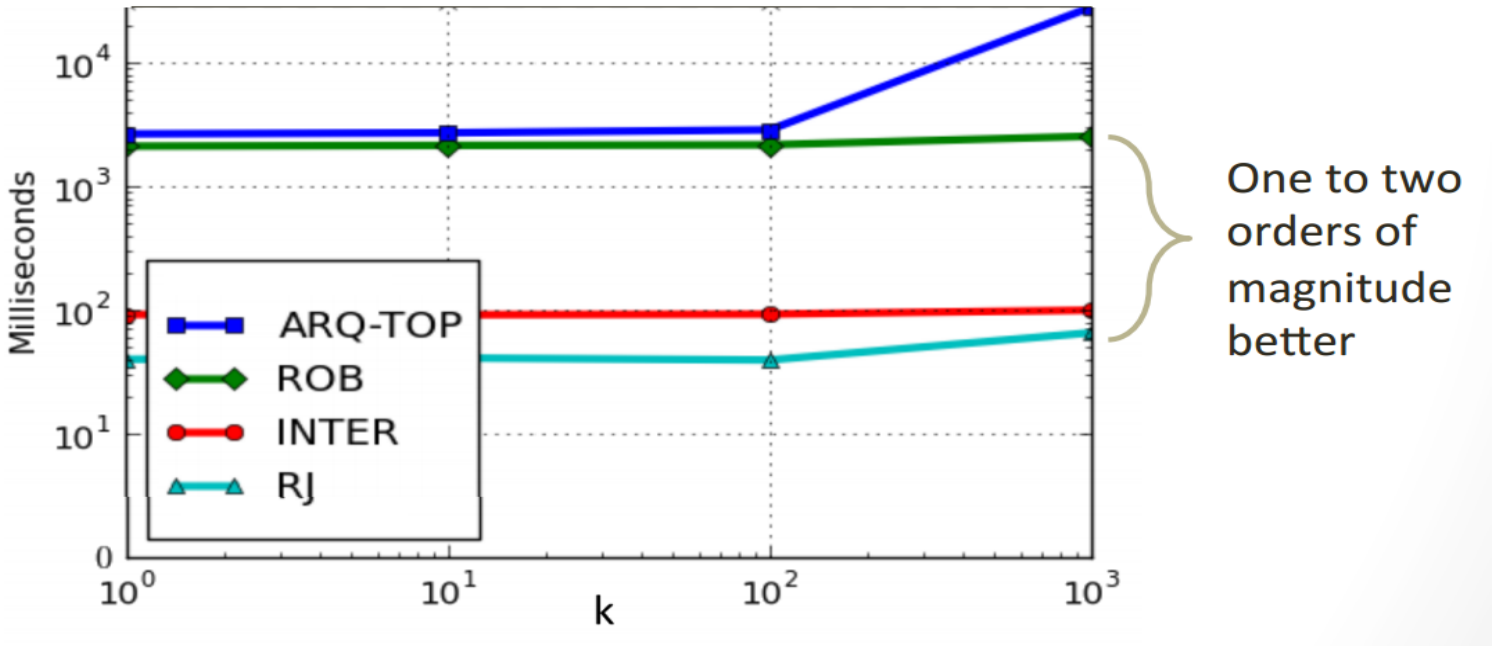
Rank-Join

Experimental Evaluation

- Prototype implementation
  - ARQ-Rank (Extends Jena ARQ 2.8.9)
- Extended version of Berlin SPARQL Benchmark
  - Added routing capabilities
  - Added top-k queries
- Jena TDB 0.8.11 as storage

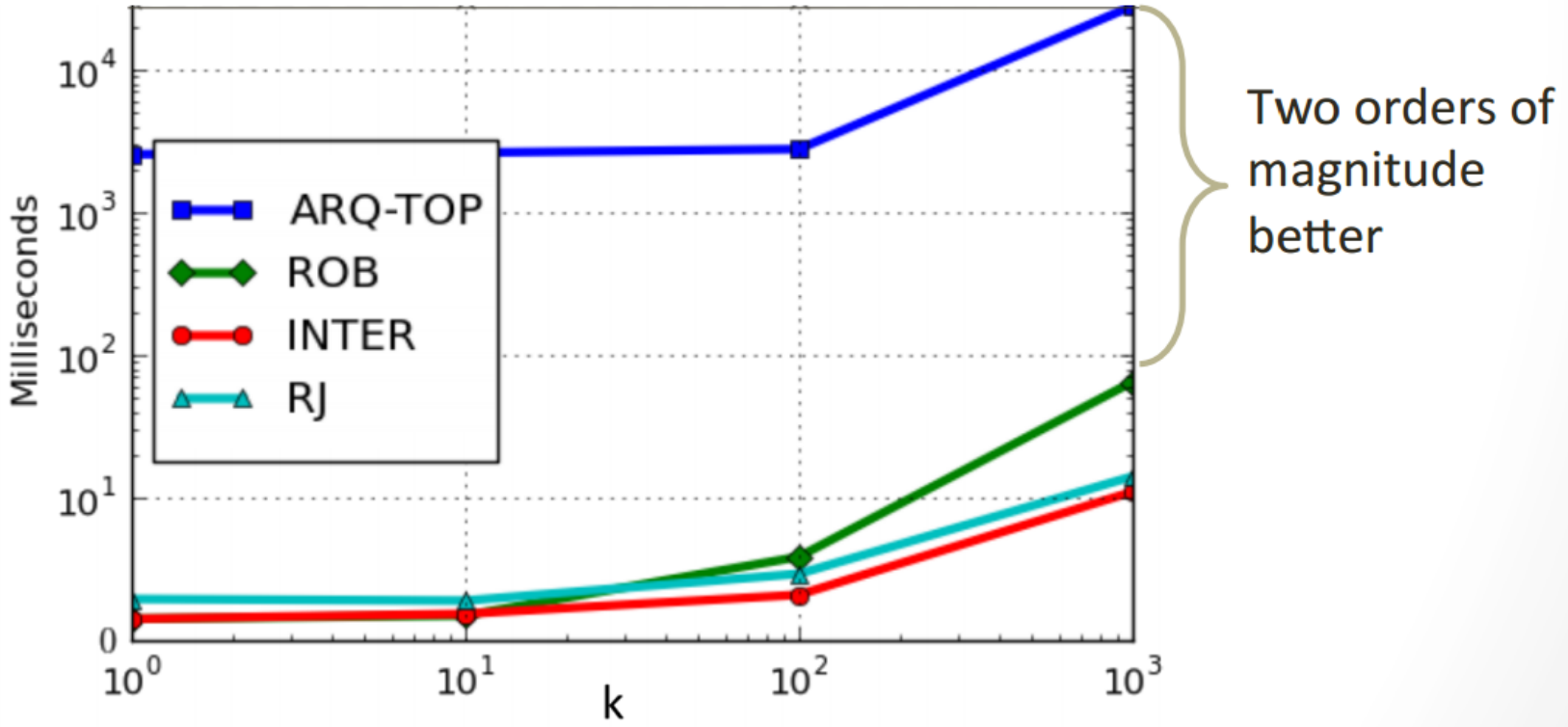
Experiment 1: Comparing planning strategies

- Example query, 5M triples dataset
- Worst case scenario: no sorted access indexes(slow sorted acccess)

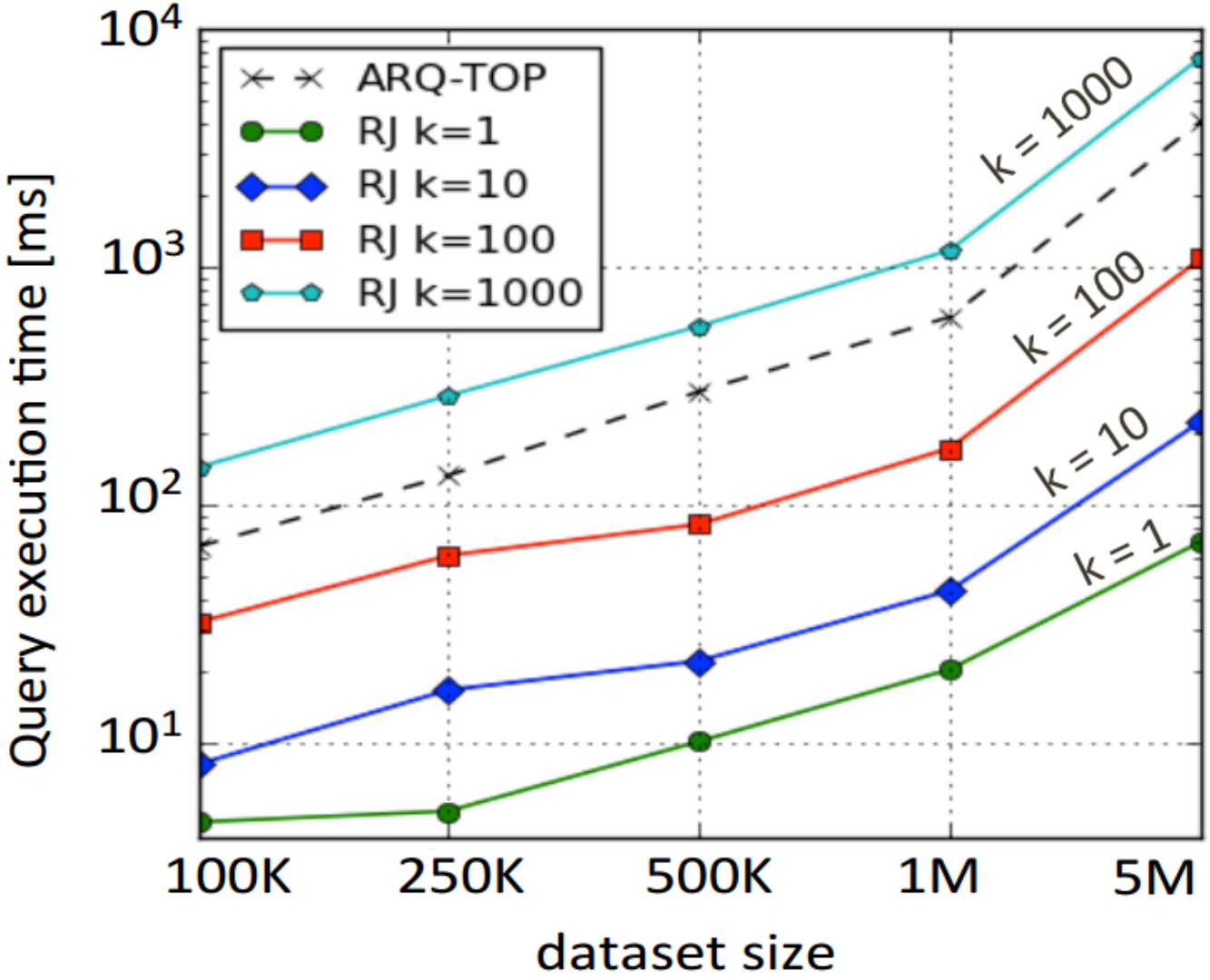


Experiment 1: Comparing planning strategies

- Example query, 5M triples dataset
- Standard scenario: sorted access indexes(fast sorted access)



Experiment 2: Small Benchmark (8 queries)



Conclusions

- A system that speeds up the execution of top-k queries in SPARQL by orders of magnitude:
  - STEP 1: A rank-aware SPARQL algebra (SPARQL-Rank algebra)
  - STEP 2: A rank-join algorithm (RSEQ)
  - STEP 3: Three planning strategies (ROB, INTER, RJ)
- ARQ-Rank, a rank aware extention of Jena ARQ

Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data

Motivation

- Semantic search
  - Access to semantically described documents
  - Support for expressive / precise information need
- **How to capture the user's information need?**
  - *Expressive queries* with difficult syntax (SQL, SPARQL) vs.limited but *intuitive queries* (Keywords)

Motivation

- **Expressive power** is crucial!
- Support the user in specifying information needs in an **intuitive way** is also crucial!

Goal: Interpreting Complex Information Needs by **Translating Keywords to Expressive Formal Queries**

Related work

- Translation of **NL questions**
  - Can the user specify a precise question when the information need is vague?
- **Relaxed-structure query models**
  - Require some knowledge about the query syntax and the structure of the underlying data
- In keyword search, the **user does not need to know about the query syntax and data schema**
  - Crucial for environment like the Web where most data sources to be queried are unknown to the user

Related work

- In most approaches, the exact mapping between keywords and labels of data elements is performed to obtain keyword elements.
- Algorithms for finding answer trees are backward search and to guarantee that computed answers have best scores, book keeping the information is required, which is costly. Hence most of the algorithms compute answers in an approximate way.
- *proposed to reach early termination* after obtaining the top-*k results, instead of searching* the data graph for all results.

Keyword Search – An Overview

- Mapping of keywords to ”labels” of data elements
  - Instead of presenting the top-k answers, which might belong to different queries, user is provided with top-k queries to retrieve all answers.
  - Keywords might also be mapped to edges.

- Data Graph exploration
  - Search for **substructures** (query graph) connecting keyword elements
  - **Query graph** vs. answer trees
  - Exploration of query graphs operates on **summary of data graph** only

Keyword Search – An Overview

- Top-k computation
  - Search guided by a **scoring function** to output only the top-k results
  - **Guaranteed top-k** vs. approximate top-k .

- Mapping query graph to conjunctive query
- Processing the conjunctive query using standard query engine

Keyword Search – An Overview

- Basic idea summarized
  - Map keywords to data elements (keyword elements), search for substructures on the data graph that connect the keyword elements, and output the top-k substructures computed on the basis of a scoring function.
  - Instead of presenting the top-k answers, which might actually belong to many distinct queries, the user select one of the top-k queries to retrieve all its answers.
  - Keywords do not necessarily correspond to answers exclusively, they might also be mapped to edges.

Scenario – Interpreting Information Needs

User Information Need

Query Specification

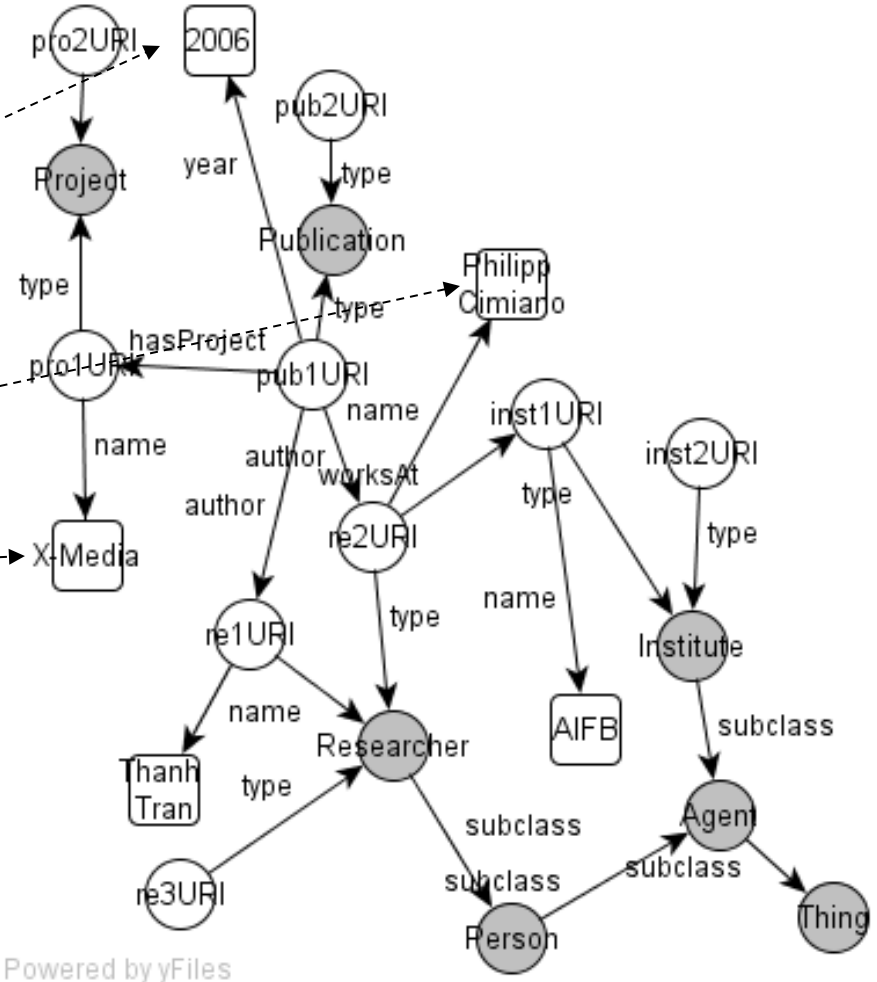
2006 Philipp Cimiano X-Media“

Query Translation

Query Processing

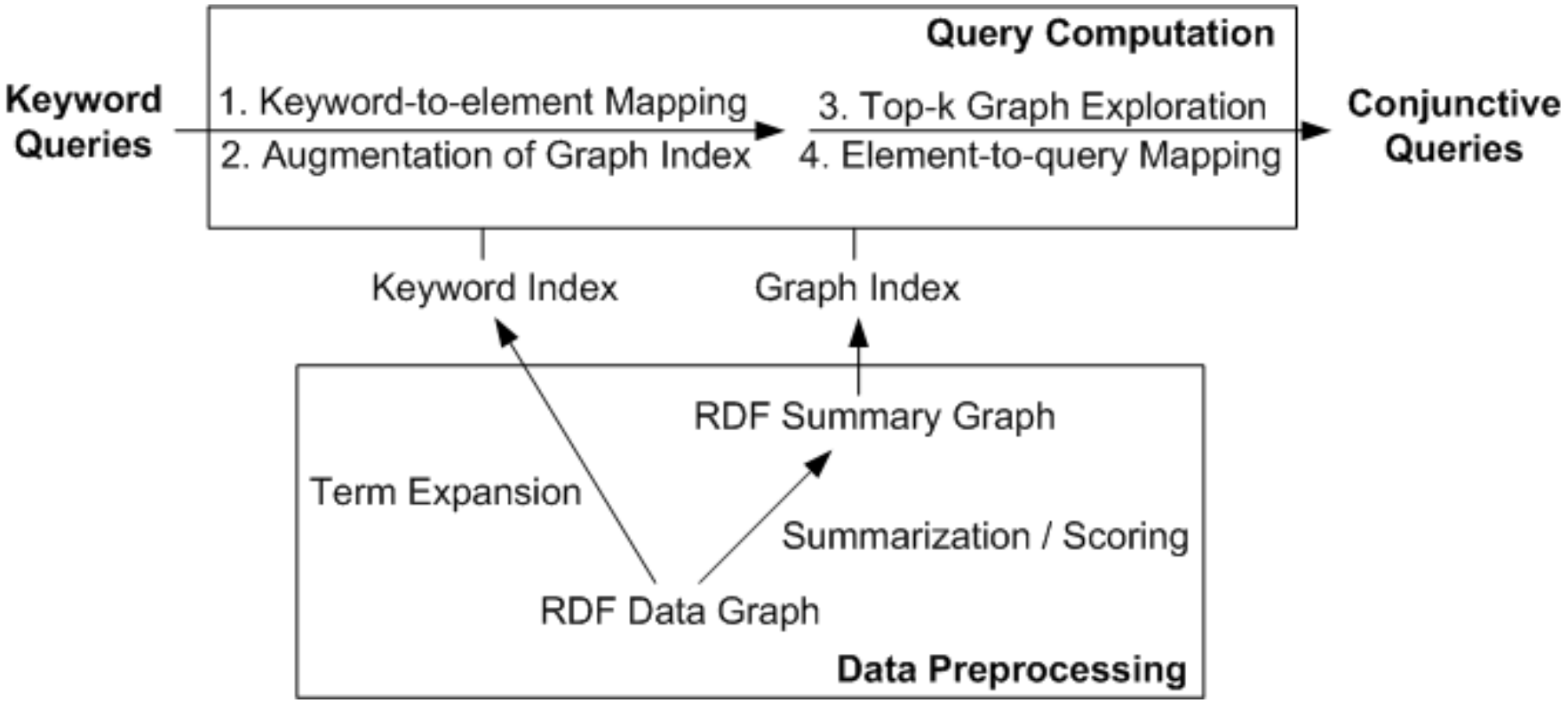
```
SELECT ?x , ? y , ? z WHERE {
? x type  Publication .  ? x  year 2006 .
? x  author ?y .  ? y name ’P . Cimiano
’ . ? y worksAt ? z .  ? z name ’AIFB’}
```

RDF Data Graph



Keyword Search – The Workflow

- **Offline**: Summarization, Scoring, Term Expansion
- **Online**: Query Computation, Query Processing



Query Computation

- In order to compute queries, the keywords are firstly mapped to elements of the data graph.
- From these *keyword elements, the graph is then explored* to find a *connecting element—a particular type of graph* element that is connected to all keyword elements.
- The paths between the connecting element and a keyword element are combined to construct a *matching subgraph. For each subgraph,* a conjunctive query is derived.
- vertices are mapped to variables or terms, and edges are mapped to predicates. The process continues until the top-*k queries have been computed.*

Preprocessing

- In order to perform these steps in an efficient way, the data graph is preprocessed to obtain a *keyword index* that is used for the keyword-to-element mapping. Labels of data elements are indexed. Additionally, *semantically* related labels extracted from Wordnet are also incorporated in a process called term expansion.
- For graph exploration, a *graph index is constructed, which is basically* a *summary of the original RDF graph containing structural* (schema) elements only.

Preprocessing

- At the time of query processing, graph index is augmented with keyword elements obtained from the keyword-to-element mapping. The augmented index contains sufficient information to derive the structure as well as the predicates and terms of the query.
- In order to fetch the top-*k queries, graph elements are also augmented* with *scores. While scores associated with structure elements* can be computed off-line, scores of keyword elements are specific to the query and thus, can only be processed at query computation time.

Scoring

- Computed queries can result in many queries. Scoring functions assess the relevance of the computed queries.
- Path length - computed offline
  - Based on the assumption that the information need of the user can be modelled in terms of entities which are closely related. Thus, a shorter path between two entities (keyword element) is preferred.
- Popularity Score - computed offline
  - Several cost functions aim to capture the "popularity" of an element of the summary graph, measured by the relative number of data elements it actually represents.
- Keyword matching score are computed online

Graph Summarization

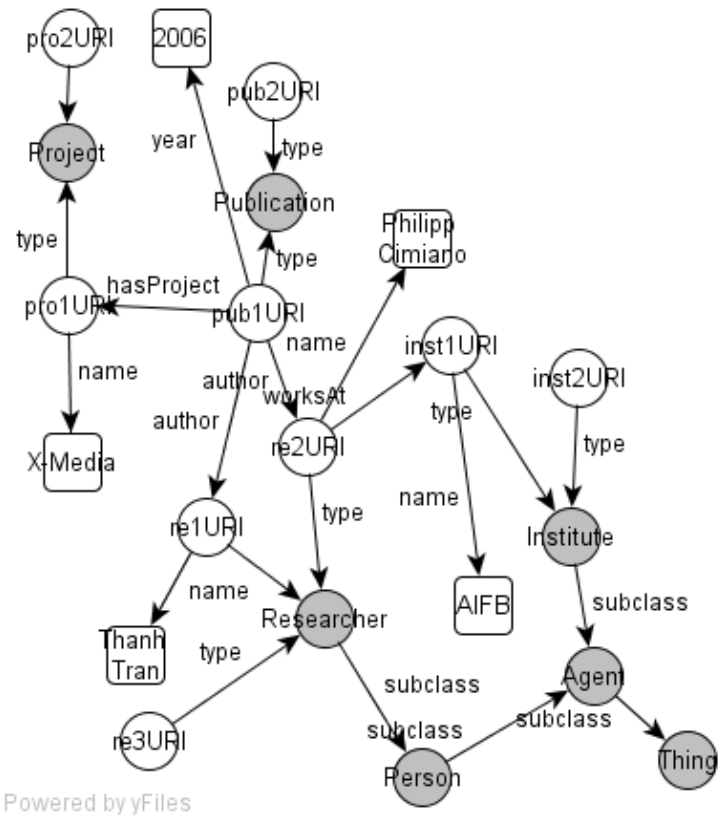
- A rdf *data graph G is a tuple (V; L;E) where*
  - *V is a finite set of vertices. V is conceived* as the disjoint union *VE union VC union VV with*
  - *E-vertices VE* (representing entities),
  - C-vertices *VC (classes),*
  - *V-vertices VV (data values).*

L is a finite set of edge labels,

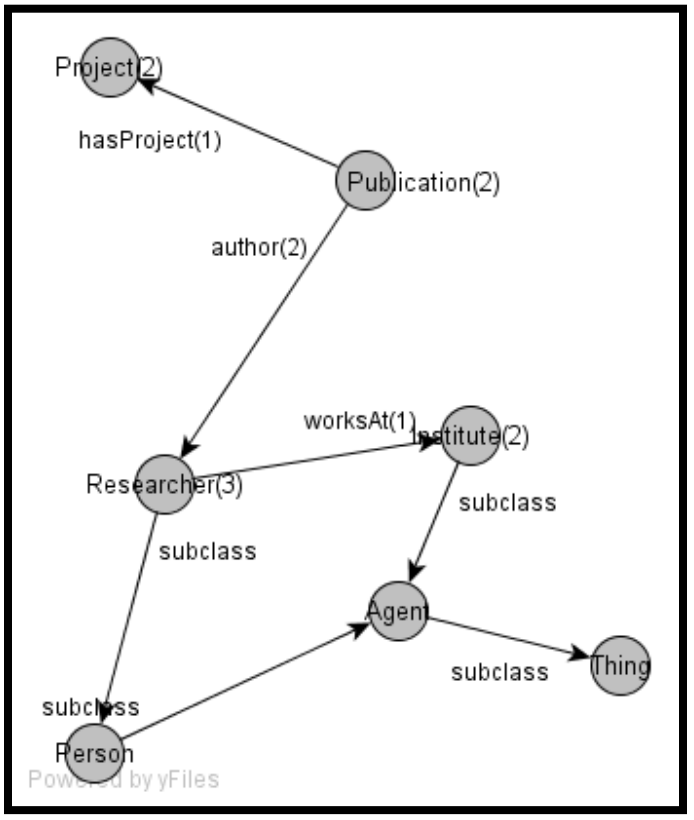
E is a finite set of edges of the form e(v1,v2) with v1,v2 in V and e in L.

Graph Summarization

- Goal: preserve sufficient information to compute elements and structure of the query, while **reducing the exploration space**
- Summary graph captures **relations between entity classes**, thus preserve structural information of the original data graph



Example RDF Graph



Summary Graph

Graph Summarization

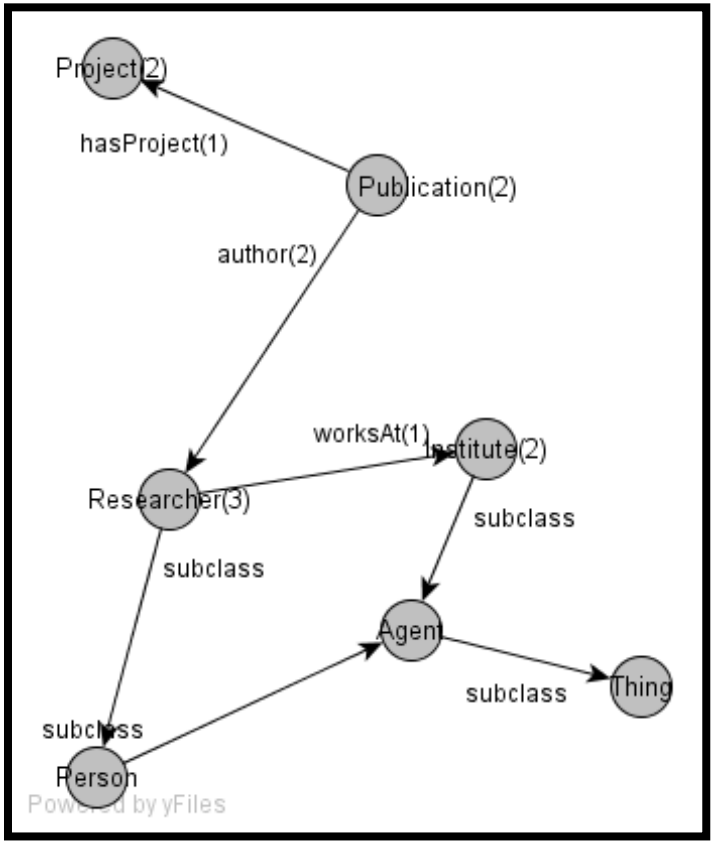
- The graph index is used for the exploration of substructures that connect keyword elements. (In most of the current approaches, this graph index is simply the entire data graph, thus, exploration might be very expensive when the data graph is large)
- Objective is to derive the query structure from the edges and the query terms from the vertices of the computed subgraphs.

Graph Summarization

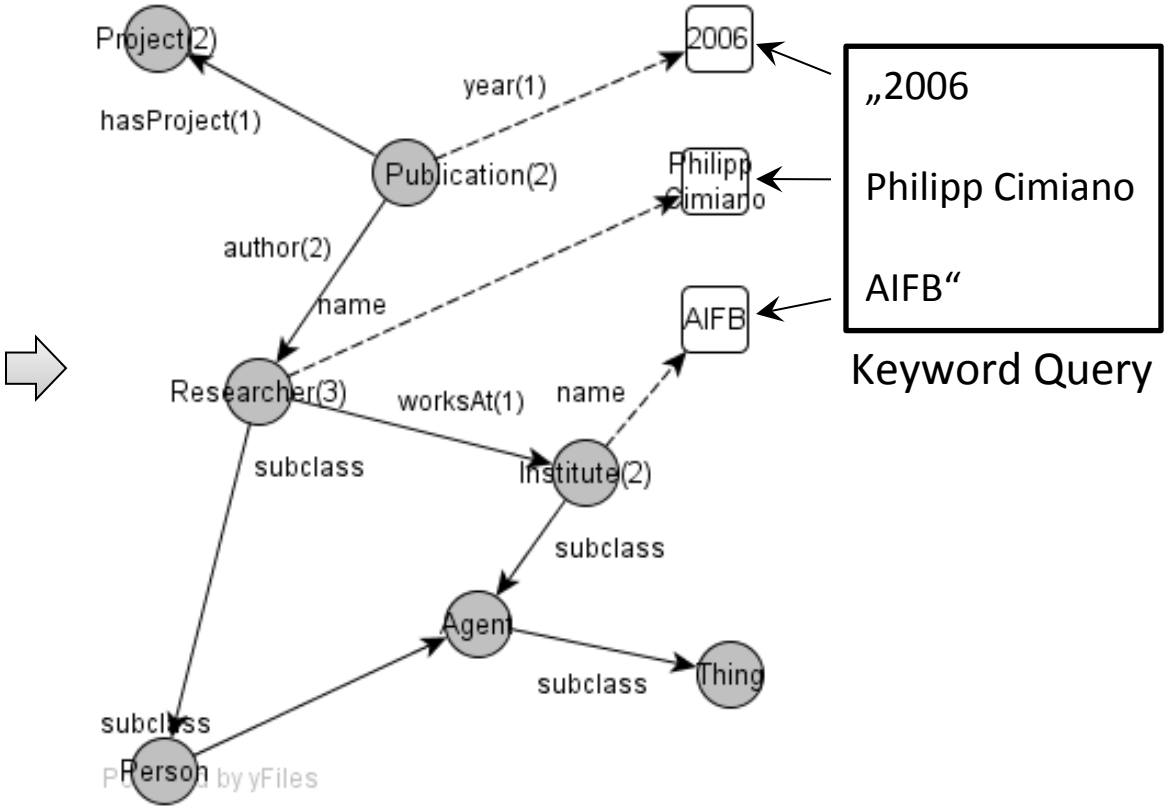
- URIs denoting E-vertices are verbose and thus, are unlikely to be used as keywords. Thus, if structural information can be preserved, this type of vertices can be omitted when building the graph index.
  - To achieve this, summary graph is used, which intuitively, captures only relations between classes of entities:
  - This summary graph can be seen as a RDF schema; however it might contain of relations that have not been specified in the RDF schema
  - It is similar to the dataguide concept; which is used for semi-structured data where there is no schema available or schema is incomplete

Keyword Mapping & Graph Augmentation

- Summary graph captures information for exploration of **query structure**
- Online augmentation with elements & scores obtained from keyword mapping
- Augmented graph contains further information for exploration of **query elements**



Summary Graph



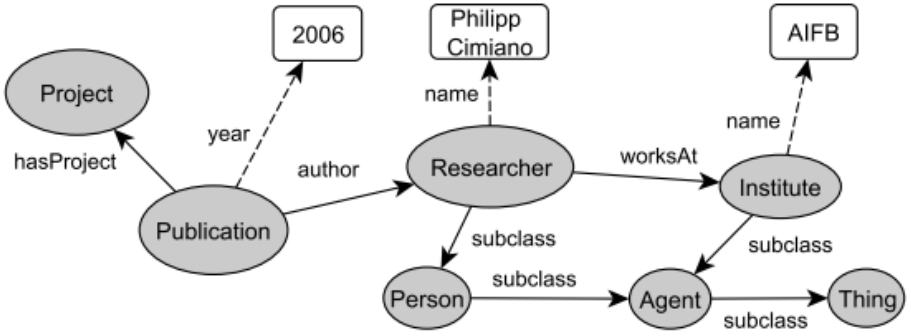
Augmented Summary Graph

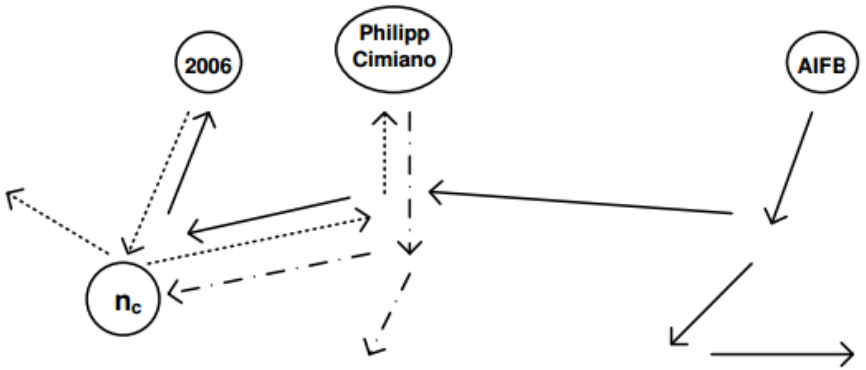
Keyword Mapping & Graph Augmentation

- Till now, A-edges and V-vertices have not been considered in the construction of the summary graph.
- They are relevant for query computation only when they are keyword elements themselves. In order to keep the search space minimal, the summary graph is augmented only with the A-edges and V-vertices that are obtained from the keyword-to-element mapping:
- The augmented summary graph contains both the structural information computed during preprocessing (rendered gray) and the query specific elements added at query computation time (shown in white).
  - *For each of the keyword elements, the score as computed* off-line is combined with the matching score obtained from the keyword-to-element mapping.

Top-k Graph exploration

- The graph exploration starts from the three keyword matching vertices, resulting in different paths.
- Among them, the following three paths meet at the connecting element *nc,*
  - *p1(AIFB; :::; nc),*
  - *p2(Philipp Cimiano; :::; nc)*
  - *p3(2006; year; nc)*
- *These three* paths are merged to obtain a matching subgraph





General idea of exploration

The exploration starts with a set of keyword elements. For this, exploration cursors with an “empty path history” are created for the keyword elements and placed into queues

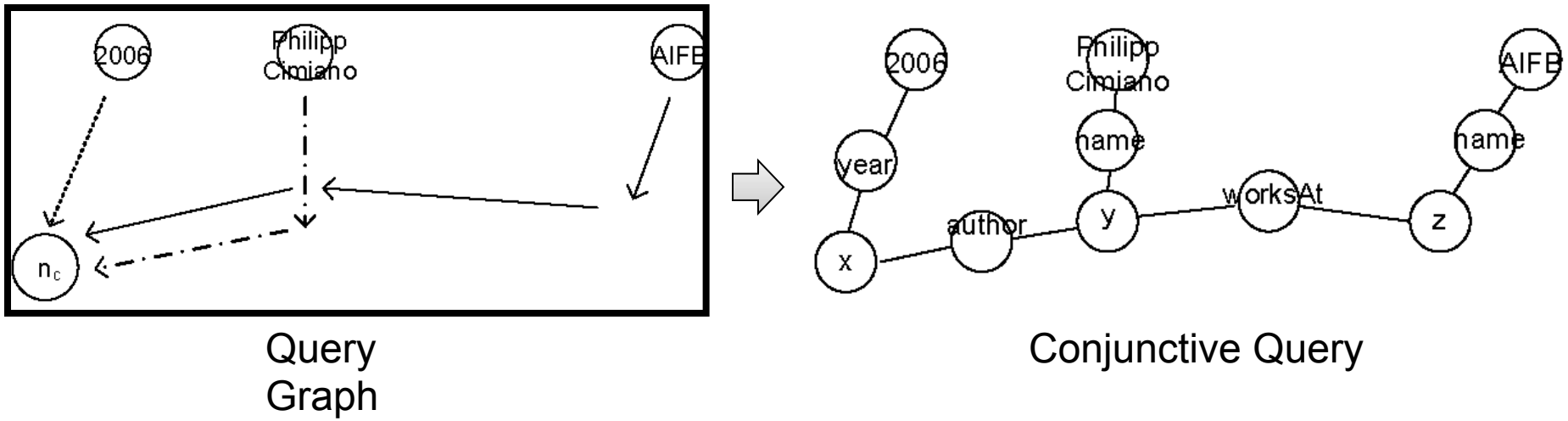
- *During exploration, the “cheapest” cursor* created so far is selected for further expansion.
- Every cursor expansion constitutes an exploration step, where new cursors are created for the neighbors of the element just visited by the current cursor.
- At every step of the exploration, top-*k* is invoked to check whether the element just visited is a *connecting element*, i.e. it connects every keyword elements (there is at least one path between this element and every keyword element)
- With information stored in this connecting element, a matching subgraph (query graph) is constructed by merging the paths between the connection element and the keyword elements
- During top-k, it is also checked whether it is safe to terminate the process.

Top-k Graph exploration

- *Proposed to reach early termination* after obtaining the top-*k results, instead of searching* the data graph for all results. The basic idea originates from the Threshold Algorithm (TA).
- Top-k algorithm finds the top-*k subgraphs with best* scores
- The score of the graph is computed from the individual scores of these paths
- Iteratively, graphs found during the exploration are added to the candidate list
- This process continues until the lower bound score (highest cost) of this candidate list (the score of the *k-ranked object) is found to* be higher (lower) than the upper bound score (lowest cost) of all remaining objects.
- The upper bound for the remaining objects is derived from the cursors waiting in the queues

Mapping Query Graph to Conjunctive Query

- Conjunctive query obtained by exhaustive application of **mapping rules**
  - Every value vertex v_{vertex} → a term
  - Every class vertex c_{vertex} → a distinct variable
  - Every A-edge $e(c_{vertex}, v_{vertex})$ → a query predicate $e[var(c_{vertex}), term(v_{vertex})]$
  - Every R-edge $e(c_{vertex1}, c_{vertex2})$ → a query predicate $e[var(c_{vertex1}), var(c_{vertex2})]$
- Treat all query variables as distinguished
- Specific mechanisms can be provided for the user to **choose distinguished variables**
- Query chosen by the user finally translated to query formalism supported by the query engine (SPARQL) for retrieving query answers



Query Graph

Conjunctive Query

Evaluation – Effectiveness

- 12 users provide 30 keyword queries on DBLP, along with the NL description of the information need
- **Reciprocal Rank = 1/r**, where r is the rank of the correct query
- A **query is correct if it matches the information need**
- Information need can be interpreted in most cases, in particular when **path length**, **matching score** as well as **popularity of graph elements** are incorporated into scoring function

Evaluation – Usability of Query Interpretation

- Standard approaches return top-k results
- This approach based on interpretation of keywords as queries, i.e. **compute top-k queries** instead of top-k answer trees
- Queries are then transformed to simple natural language and **presented to user**
- 90% of users prefer to obtain question first, since it **facilitates understanding of results**
- All user prefers to do refinement on the structured query, rather than on the keywords, since the **structured query can be manipulated in a more precise and predictable way**

Evaluation – Efficiency

- Comparison with bidirectional search [V. Kacholia et al.] and search based on graph indexing (1000 BFS, 1000 METIS, 300 BFS, 300 METIS in [H. He et al.])
- Measure time for **query computation + time for processing several queries until finding 10 answers**
- Outperforms bidirectional search by at least one order of magnitude
- Performs fairly well when compared to indexing based approaches

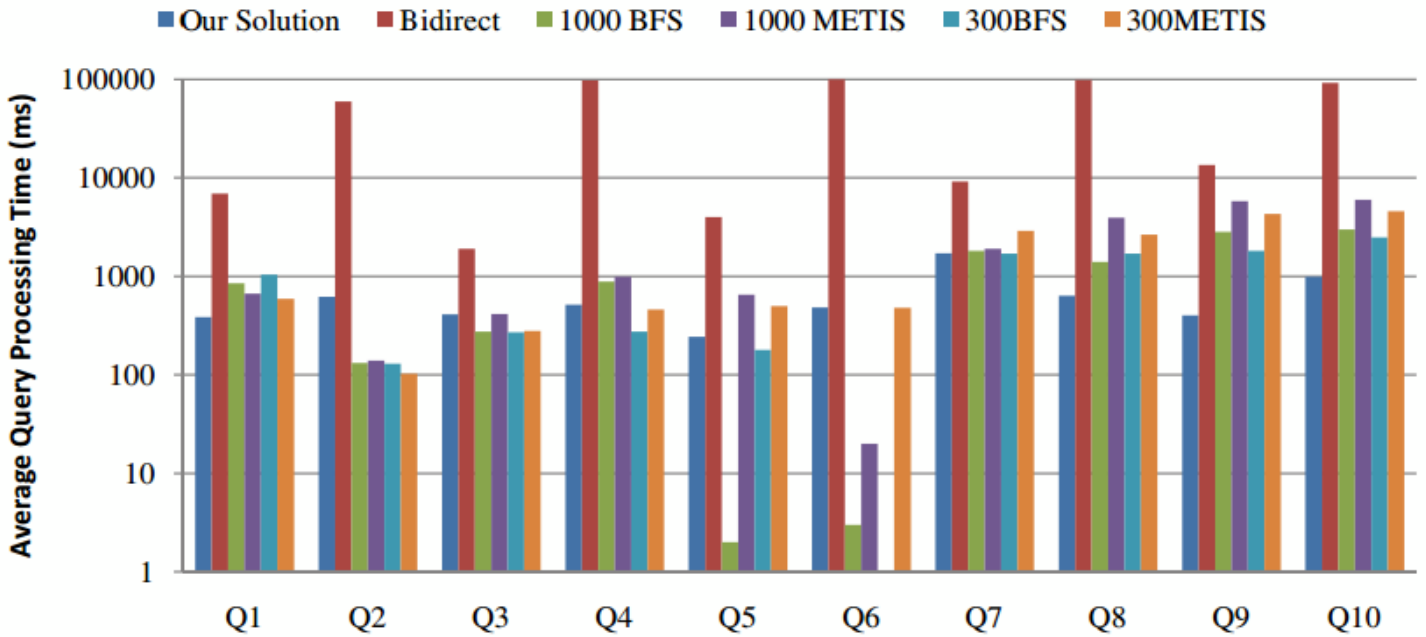


Fig. 5. Query Performance on DBLP Data.

Conclusions and Future Work

- Conclusions
  - A new approach for **keyword search on graph-structured data**, RDF in particular
  - Novel algorithms for the **top-k exploration of subgraphs** to **compute queries** as an additional intermediate step
  - **Query computing** is performed on **an aggregated graph** while **query processing** can leverage **optimization capability of the database**
- Future Work
  - **Indexing connectivity and scores** for further speed up
  - Consider **special query operations** (e.g. filters) as keywords

Thank you!

Definitions

Mapping μ … an intermediate SPARQL solution, equivalent to a SQL tuple

set of mappings

	?x	?y	?p1	?p2
μ1	1	8	0.8	0.8
μ2	3	3	0.3	0.6

Maximal possible score

Given a scoring function $\mathcal{F}(p_1, \ldots, p_n)$ and a set of predicates $P = \{p_1, \ldots, p_j\}$ the maximal possible score for a mapping μ is defined as:

$$\overline{\mathcal{F}}_P(p_1, \ldots, p_n)[\mu] = \mathcal{F}\left(\begin{array}{ll} p_i = p_i[\mu] & \text{if } p_i \in P \\ p_i = 1 & \text{otherwise} \end{array} \quad \forall i \right)$$

Definitions

Ranking principle

Given two mappings μ_1 e μ_2 with $\overline{\mathcal{F}}_P[\mu_1] > \overline{\mathcal{F}}_P[\mu_2]$, if we process μ_2 we need to process also μ_1.

Ranked set of mappings

Given a set of predicates P, a ranked set of mappings Ω_P is a set of mappings Ω augmented with the following properties:

- Score: for each mapping μ, the maximal possible score $\overline{\mathcal{F}}_P[\mu]$
- Order: the order relation $<_{\Omega^P}$ is defined on Ω_P based on the scores of the single mappings

The RESQ algorithm

Algorithm 1 The RSEQ getNext method

```
S: sorted access input
R: random access input
R_top: maximal possible value for R
loop
    if Q is not empty then
        topScore ← Q.topScore();
        if topScore ≥ Threshold then
            return Q.topMapping();
        end if
    end if
    μ_S ← S.getNext();
    probe R with μ_S;
    for each retrieved join combination
        insert the join combination in Q;
    end for
    Threshold ← f(μ_S, R_top);
end loop
```