

DATA SUMMARIES FOR ON-DEMAND QUERIES OVER LINKED DATA

- **Authors:**

Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, Jurgen Umbrich

- **Conference:**

19th International conference on World Wide Web

- **Presented by:**

Vishal Chaudhary

NEED OF THIS PAPER:

- Conventionally, Web data is first collected(crawl) and pre-processed(index) in central data repository before query answering starts.
- In Linked Data, even after pre-processing has been completed, structured data is accessible live and up-to-date.
- Theoretically, a Query answering system for linked data should return current answers in a reasonable amount of time.
- But, for this query processor requires knowledge of the contents of data sources.

- So, this paper:
 - proposes an index structure by putting graph-structured content into Linked data principles
 - Provide an algorithm for query answering over linked data.
 - Evaluate the system using synthetically generated queries.



INTRODUCTION:

- Linked data source use RDF(Resource Description Format) for encoding graph-structured data.
- Linked Data can be seemed to be a highly distributed system, with thousands and millions of independent data sources.
- Now, evaluating queries in this environment can be done in 2 ways:
 - **Data Warehousing or Materialisation-based approaches(MAT):** Collect data in advance > pre-process > Store results in Central database > Query local DB
 - **Distributed Query Processing(DQP) approach:** Parse, Normalize and split into sub-queries > determine sources for each sub-query > Execute sub-query against the source directly.
- But applying DQP directly is no good. Coz first, Data in different sources may have different schema. Second, queries cannot be *dispatched*, source site might not have query processing capabilities.




INTRODUCTION(CONT'D):

- So, in this paper our aim is to narrow the gap between 2 different approaches and find a balanced middle ground for processing queries over Linked Data directly.
- Since all the sources don't have query processing capabilities, paper propose a multidimensional indexing structure(a QTree) to store description of the content of data sources.(smaller index)
- Also, Qtree can be dynamically extended by user submitted sources or undiscovered sources.
- It works like this:
 - Prime an approximate index structure(a QTree) with a seed dataset.
 - Use the Qtree to determine the sources that contribute partial results for SAPRQL query Q.
 - Fetch the content of the sources into memory(usually only top-k sources)
 - Perform join processing locally, given that sources don't provide join functionality.
- Problem is:
 - Finding right sources that contain answer for the query
 - Efficient parallel fetching of content



QUERYING LINKED DATA:

- Linked Data is RDF published on the web according to these principles:
 - Use URI as names for things
 - Use HTTP URIs
 - Provide useful content at these URIs encoded in RDF form
 - Include links to other URIs for discovery
 - Linked Data is RDF published on the web according to these principles:
 - MAT(Materialisation based approach) provide excellent query response time due to pre-processing carried out during load and indexing step. But, it has few drawbacks:
 - Aggregate data is never current as collecting and indexing data is a time consuming process.
 - From a single query requester point of view, there is lots of unnecessary data gathering. Processing and storage.
 - Coz of replicated data storage, they cannot restrict or log access any more since queries are answered against a copy of the data.
- 

QUERYING LINKED DATA: (CONT'D)

- Possible approaches to evaluate queries over Web resources:
 - Direct Lookups:
 - Contains source addresses and identifiers.
 - Query processor(QP) performs lookups on the sources that contains identifiers mentioned in the query.
 - Drawback: Incompleteness problem
 - Schema-Level Indexes:
 - Based on distributed query processing, relies on schema-based indexes
 - Keeps a index structure with properties and classes that occur at certain sources and uses that structure to guide query processing.
 - Drawback: Only queries which contain schema-level elements can be answered; Very common properties might fetch huge amount of data.
 - Direct Approach:
 - Combination of instance and schema-level elements to summarize the contents of data sources.
 - Uses more resources than Schema level but has ability to handle instance level queries as well.



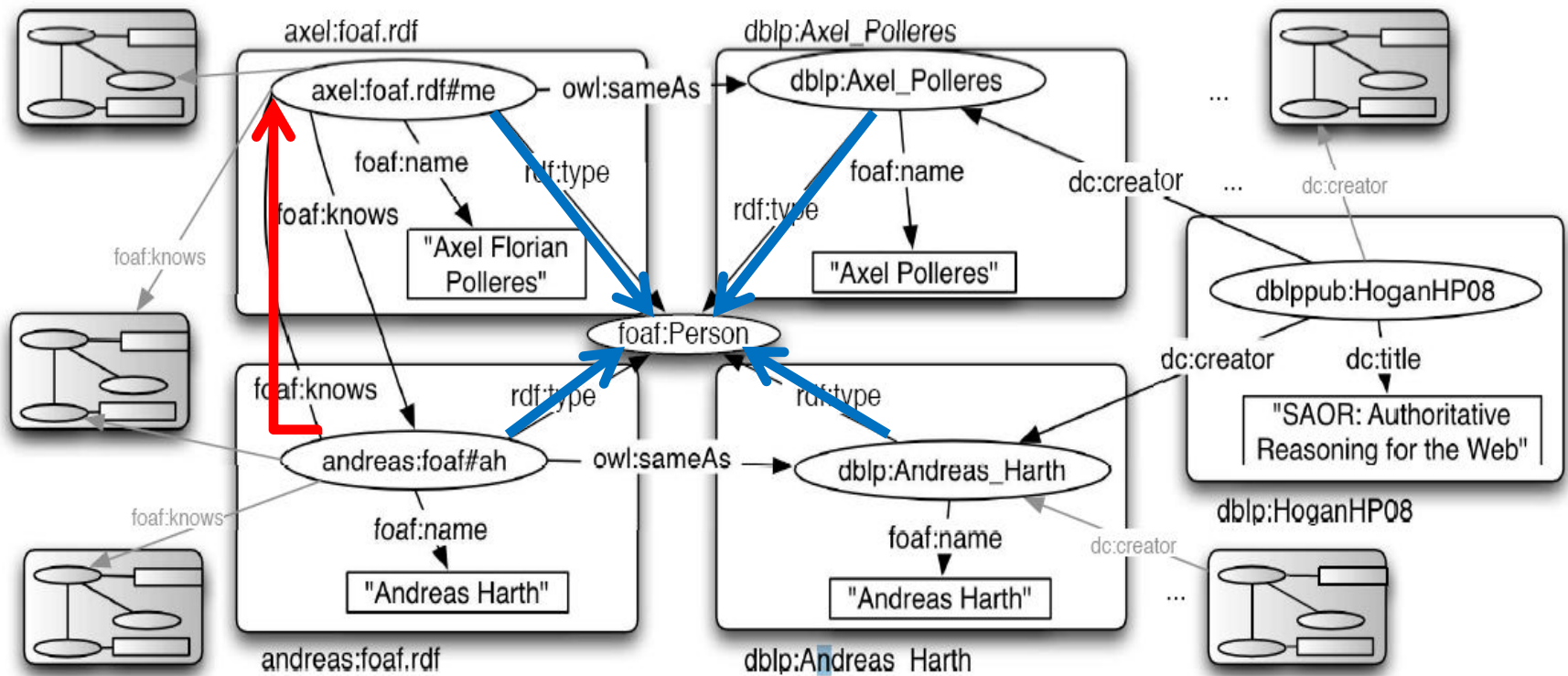


Figure 1: Linked Data in RDF about persons and their publications



SOURCE SELECTION USING DATA SUMMARIES: SOURCE INDEXING USING QTREE

- Combination of Histogram and R-Trees(for Spatial locations)
- Indexing multidimensional data, capturing attribute corelation, efficient lookups and supporting incremental construction.
- Just like R-tree, Qtree is a tree structure consisting of nodes defined by minimal bounding boxes(MBBs).
- MBBs describe multidimensional regions in the data space.
- To limit memory and disk consumption, we replace subtrees with buckets. Buckets are always the leaf nodes.
- Bucket has a number of triples in RDF components form(S,P,O)

SOURCE SELECTION USING DATA SUMMARIES: SOURCE INDEXING USING QTREE

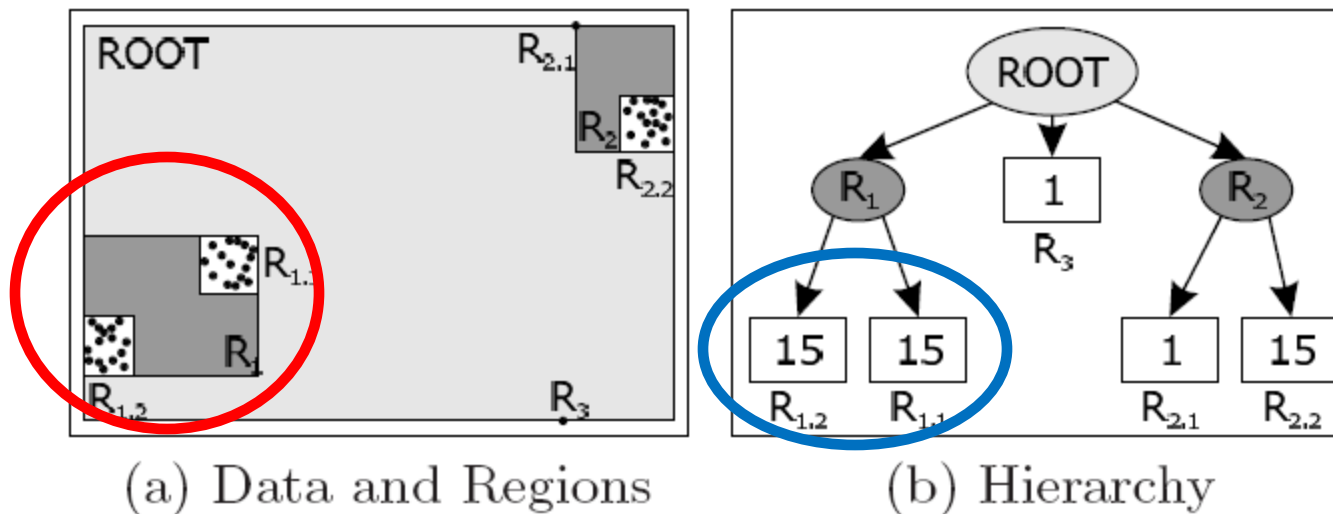


Figure 2: Two-dimensional QTree example



SOURCE SELECTION USING DATA SUMMARIES: SOURCE SELECTION

- Once the Qtree has provided the information, next important job is to identify how to use the information to answer the queries.
- There are 2 ways:
- **Triple pattern source selection:** While joining multiple triple patterns and variables, this selection method implies a prerequisite for the identification of relevant sources for a given triple pattern. For this:
 - first identify region R >
 - Use Hash function on each URI of a triple to get minimum and maximum coordinates >
 - Identify other regions that overlap with region R >
 - Find percentage of overlap amongst all the overlapped buckets(B) >
 - Cardinality of O(Overlapped region of B & R) is calculated as $[c_B \cdot \text{size}(O)/\text{size}(B)]$ where c_B is number of data items in the bucket B.



SOURCE SELECTION

- **Join Source Selection:** Here, we consider the overlaps between the sets of obtained relevant buckets for the Triple patterns(BGPs) with respect to the defined join dimensions and determine the expected result cardinality of the join. Here sources can be discarded only if whole bucket is discarded.

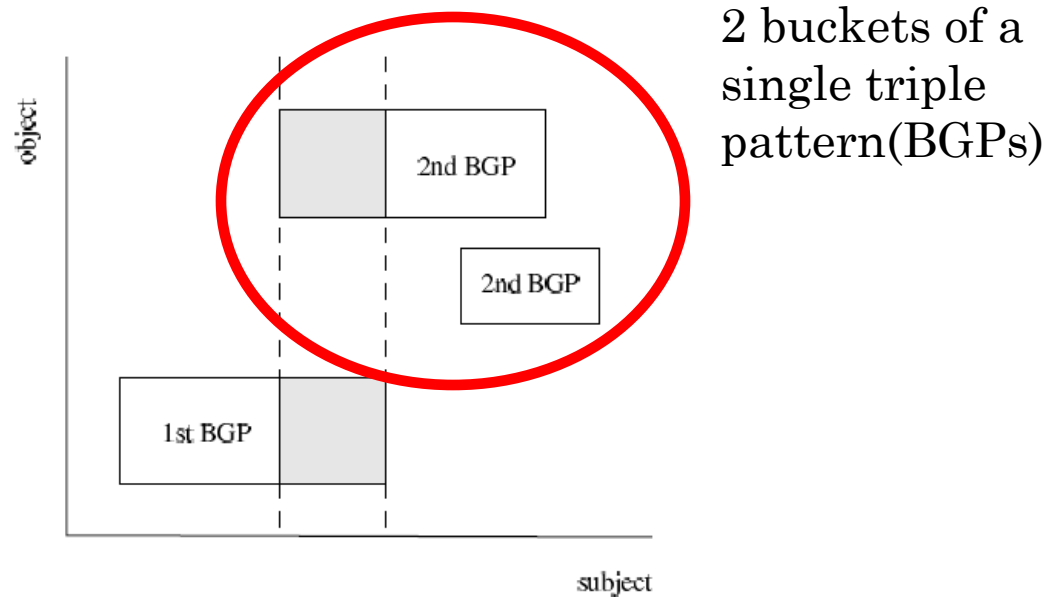


Figure 3: QTree join between first and second BGP

- Here, join is processed over the triples for subjects



SOURCE SELECTION(CONT'D)

- **Join Source Selection:** Joining results of first 2 BGPs with 3rd BGP.

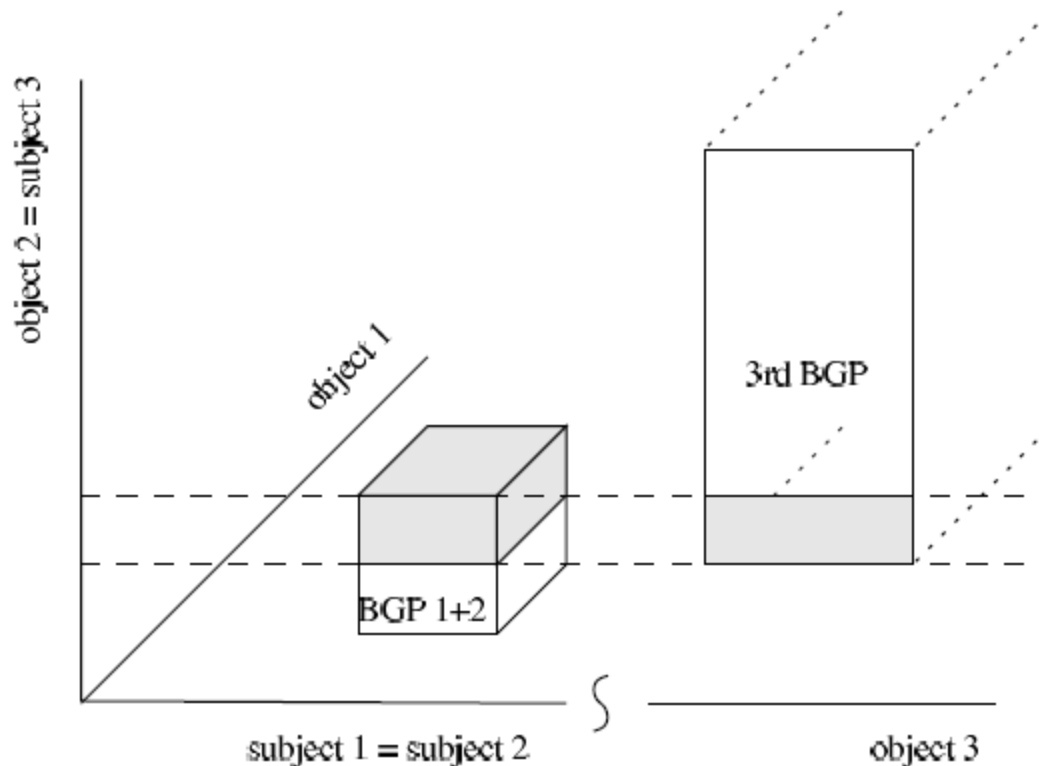


Figure 4: QTree join with third BGP

- This is processed on object-subject join between 2nd and 3rd BGP



SOURCE RANKING:

- There are 2 kinds of approach used to rank sources
 - External ranking: Ranking based on an independent or externally calculated source's relevance.
 - Cardinality ranking: Ranking based on cardinality. No external data required.

Input: Query q , QTree QT
 Output: list of relevant sources

```

1 forall buckets  $B \in QT.getBuckets(q.BGP[0])$  do
2    $O = B.overlap(q.BGP[0]);$ 
3    $join_0.insert(O, c_B \cdot \frac{size(O)}{size(B)}, \mathcal{S}_B);$ 
4 end
5 for  $i = 1$  to  $|q.BGP| - 1$  do
6   forall buckets  $L \in join_{i-1}$  do
7     forall buckets  $R \in QT.getBuckets(q.BGP[i])$  do
8        $d_L = q.joindim[i - 1]; d_R = q.joindim[i];$ 
9       if  $\exists O_L = L[d_L].overlap(R[d_R])$  then
10         $O_R = R[d_R].overlap(L[d_L]);$ 
11         $c_{O_R \oplus O_L} =$ 
12           $\frac{c_L \cdot \frac{size(O_L)}{size(L)} \cdot c_R \cdot \frac{size(O_R)}{size(R)}}{\max(L[d_L].hi - L[d_L].low, R[d_R].hi - R[d_R].low)};$ 
13         $join_i.insert(O_L \oplus O_R, c_{O_R \oplus O_L}, \mathcal{S}_L \cup \mathcal{S}_R);$ 
14      end
15    end
16  end
17 end
18 return  $\bigcup_{B \in join_{|q.BGP|-1}} \mathcal{S}_B$ 
    
```

Algorithm 1: *identifyRelevantSources(Query, QTree)*



DATA SUMMARY CONSTRUCTION & MAINTENANCE:

- **Initial Phase:**

- There are 2 different approaches for Initial phase

- **Pre-Fetching:**

- To fetch seed sources for the QTree from the Web using a Web crawler. The quality of query answers will depend on the selection of the seed sources and depth/exhaustiveness of the crawl.

- **SPARQL queries:**

- Starts with an empty Qtree and uses initial SPARQL query to collect the initial sources for the Qtree build. Index expands on further queries(next subsections). Given an SPARQL query, an agent iteratively fetches the content of URIs selected from bound variables of the query.



DATA SUMMARY CONSTRUCTION & MAINTENANCE:

○ Expansion Phase:

- Given a SPARQL query, it is very likely that the initialised QTree may contain information about dereferenceable URIs that are not (yet) indexed.
- In this case, the Qtree should be updated with the newly discovered URIs to increase the completeness of answer sets for the next time a query is executed.
- **Push of sources** is a passive approach to get new data indexed into the QTree. With passive expansion we refer to all methods that involve users or software agents notifying the QTree about new sources.
- **Pull of sources** is an active approach to index new data from the Web. One way to achieve this is to perform lazy fetching during query execution. Lazy fetching refers to the process of dereferencing all new URIs needed to answer a query.



EVALUATION:

- We expect the QTree approach to be a lightweight but efficient and effective method to limit the search for query answers to only a subset of relevant sources.
- Setup: Using a breadth first crawl of depth four starting at Tim Berners-Lee's FOAF file⁵, we collected about 3 million triples from about 16,000 sources.
- The data set represents a heterogeneous and well-linked collection of documents hosted on various domains and with different numbers of RDF triples.
- All experiments are performed on a local copy of the gathered data using Java 1.5 and a maximum of 3 GB main memory.
- We experimented with queries corresponding to two general classes. The first class of sample query is star-shaped queries with one variable at the subject position.
- The second type of query is path queries with join variables at subject and object positions.



EVALUATION: (CONT'D)

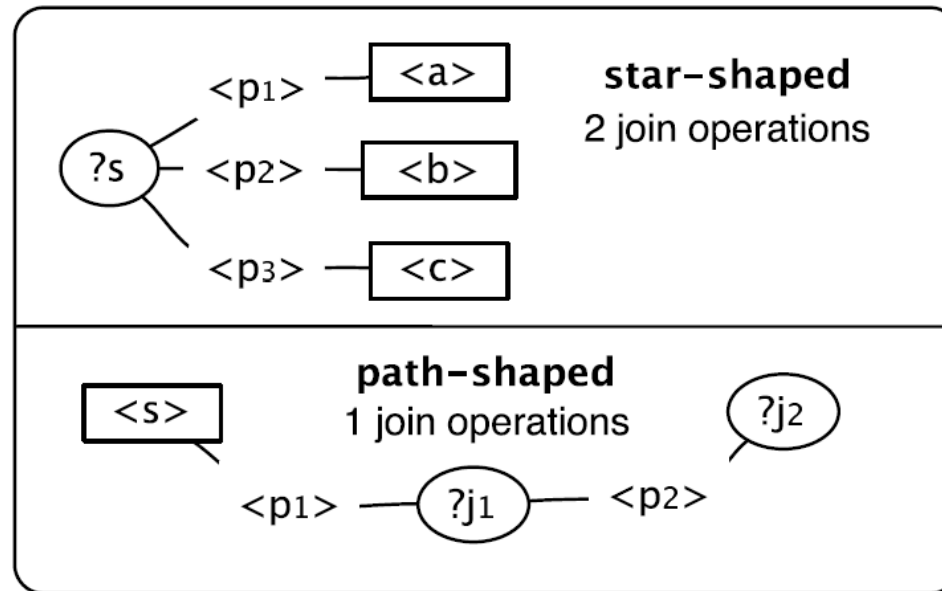


Figure 5: Abstract illustration of used query classes



RESULTS:

- Index construction: measured time to insert one triple into the QTree is 4ms on average.
- The final QTree requires a disk size of around 22 MB in serialised form.
- As the original data is of size 561 MB, this corresponds to a compression ratio of 96%.
- Results for 4 different evaluation aspects:
 - quality of source selection,
 - impact of ranking,
 - query execution time,
 - and comparison with other approaches



QUALITY OF SOURCE SELECTION:

- For Star-shaped queries, benefit of above 80% was observed.
- For path queries, we achieve benefits of about 20%, 40% and 60%

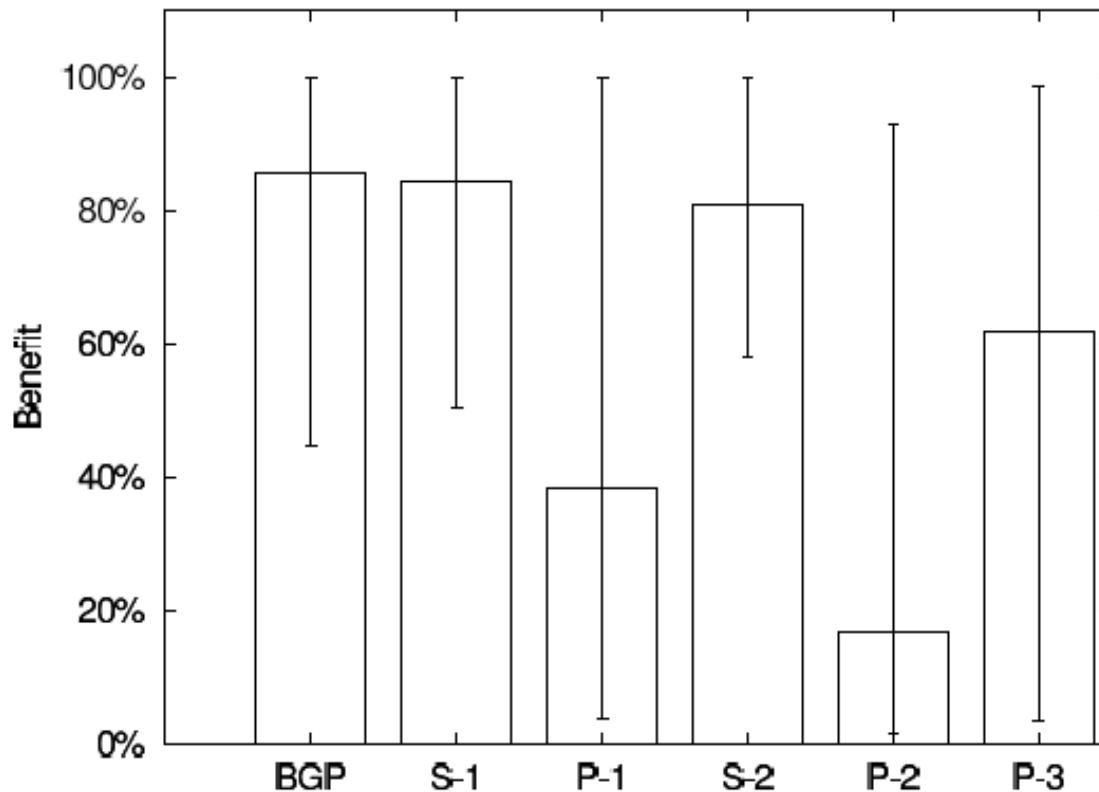


Figure 6: Benefit of source selection



IMPACT OF RANKING:

- To show the impact of the ranking, we measured how many result triples we can determine and how many queries we can completely answer when querying only top-k ranked sources.

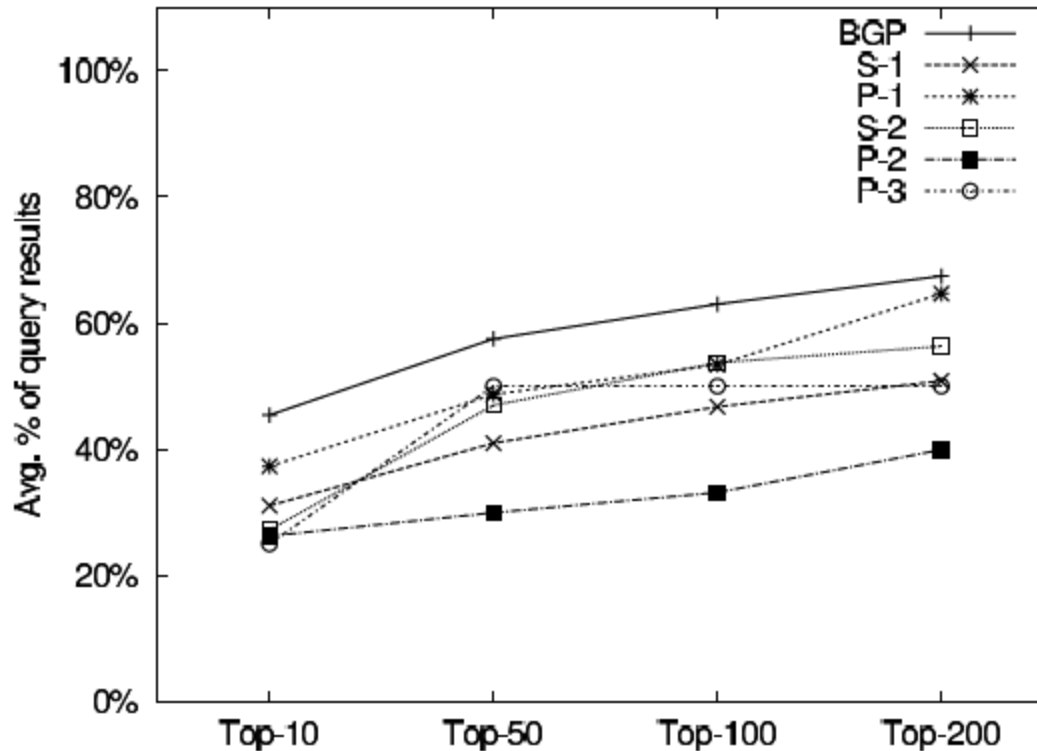


Figure 7: Impact of ranking, recall of triples



IMPACT OF RANKING: (CONT'D)

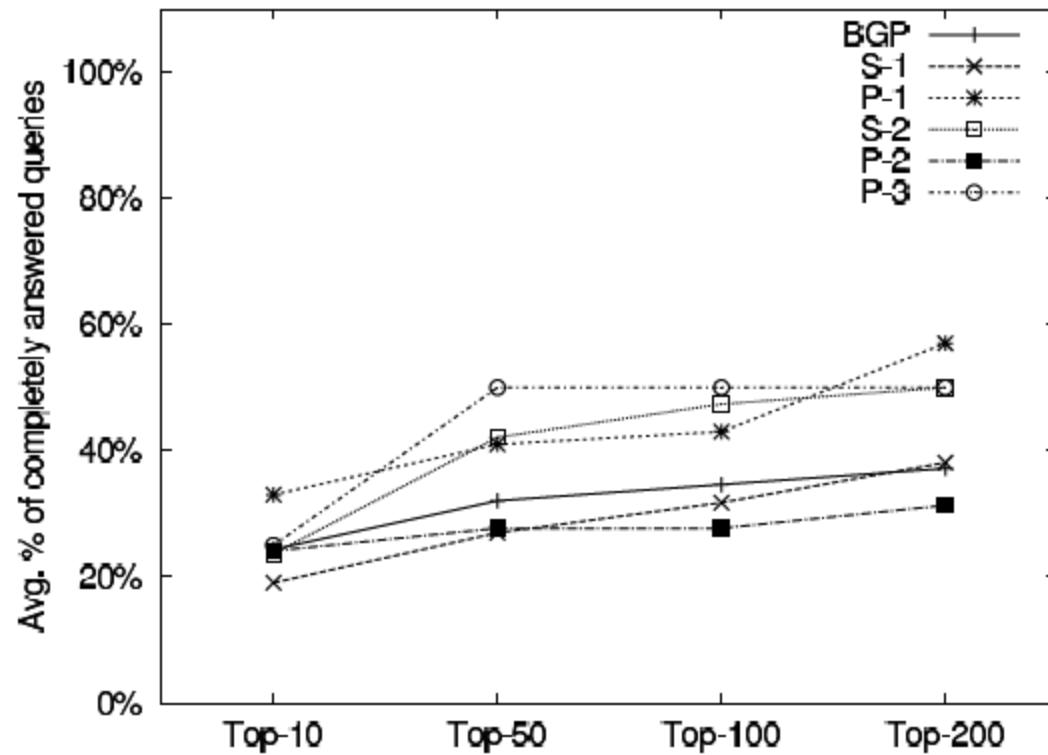


Figure 8: Impact of ranking, answer completeness



IMPACT OF RANKING: (CONT'D)

- This shows the average maximal k that would be required to answer a query completely

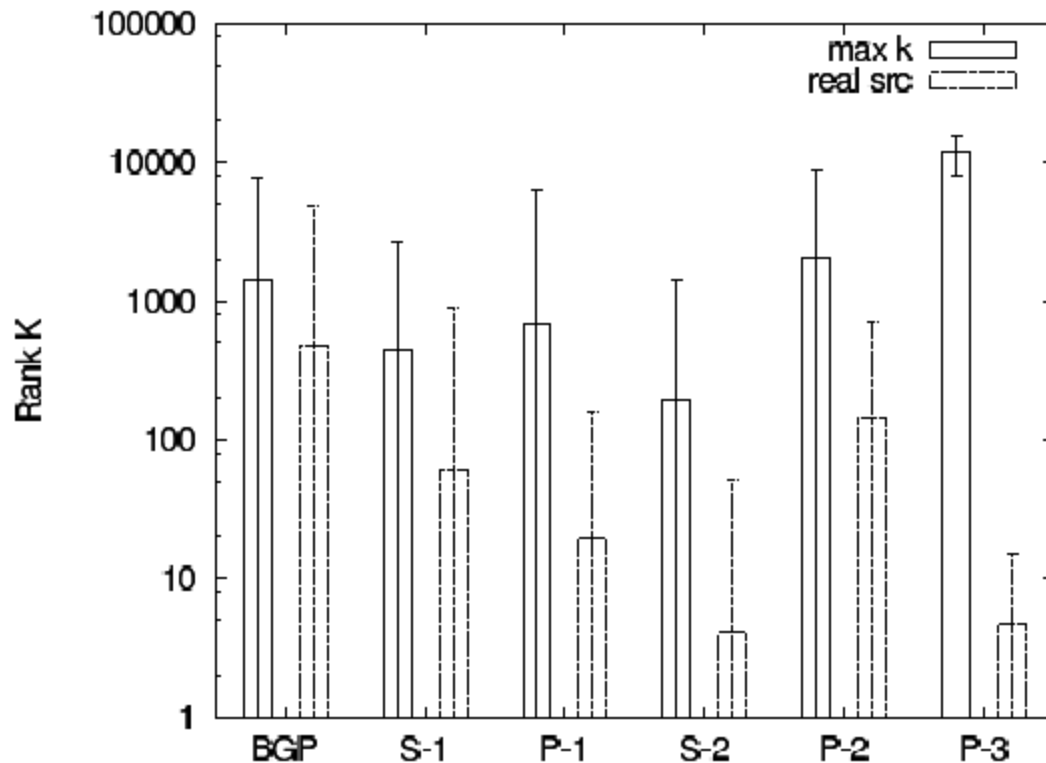


Figure 9: Impact of ranking, maximal k



QUERY EXECUTION TIME:

- It shows the average time required to estimate relevant sources (Qtree) and to actually evaluate the query afterwards on the content stored in memory (query).
- The average query time for all queries is below 10 seconds, with some outliers of maximum 100 seconds.

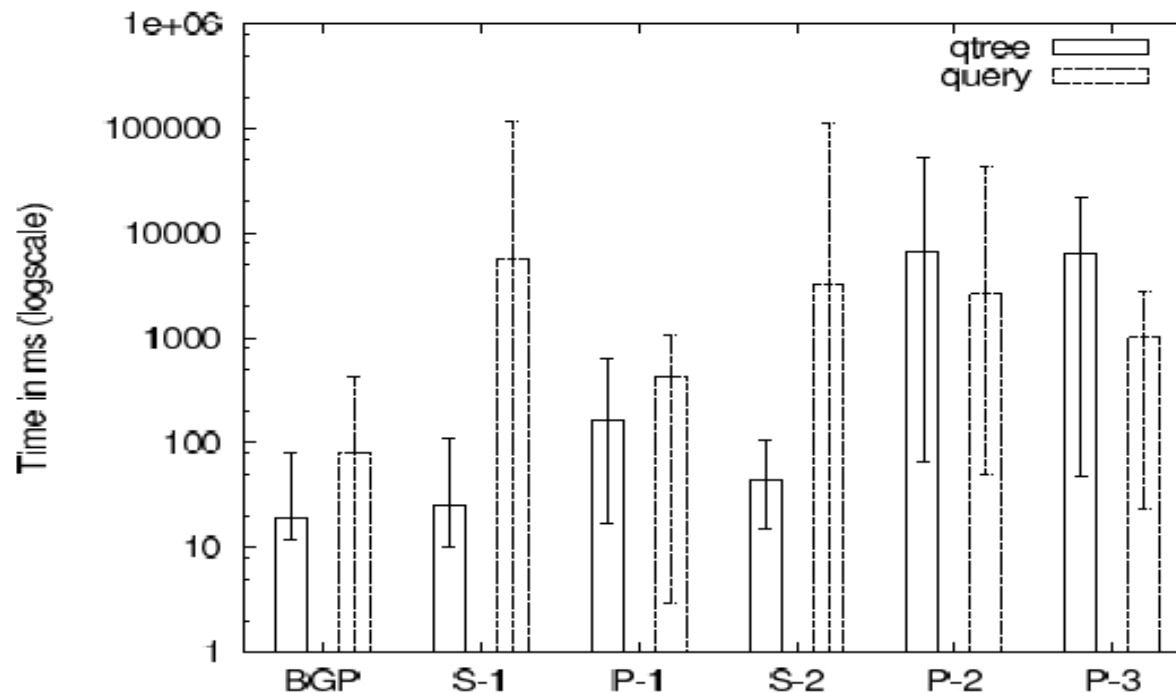


Figure 10: Query time



COMPARISON WITH OTHER APPROACHES:

- Compared our proposed solution with an alternative approach, namely the DL approach.
- We cannot expect the results to be completely accurate since the DL approach performs, by design, live HTTP lookups.
- Despite this difference, an evaluation based on crawl data reflects the general limitations of the DL approach.

| | BGP | S-1 | P-1 | S-2 | P-2 | P-3 |
|---------|-------|-------|-----|-------|-----|-----|
| Average | 32.8% | 20% | - | 9.64% | - | - |
| Maximum | 100% | 39.8% | - | 27.8% | - | - |

Table 1: Completeness of results with the DL approach

- DL approach is capable of returning results only for star-shaped queries with less than 2 joins, for path queries the DL approach returned no results.

DISCUSSION:

- Evaluation shows that our novel approach is very promising and practical for efficiently querying the Linked Data Web.
- But as expected, this is only practical if an accurate ranking is applied.
- A client, able to perform multithreaded lookups and set up with an appropriate timeout for fetching the content of the estimated sources, can answer queries with live results in less than a minute using an index of 4% size of the original data.
- All of our expectations were met by the evaluation.
- In summary, the proposed approach represents a novel, efficient and effective way of supporting source selection for live queries over the Linked Data Web.

QUESTIONS ?

