Efficient Management of Spatial RDF Data

John Liagouris[1,3], Nikos Mamoulis[1], Panagiotis Bouros[2], Manolis Terrovitis[3]

[1]University of Hong Kong
[2]Humboldt Universität zu Berlin
[3]IMIS - Research Center “Athena”

Contents

- Overview of RDF
- Problem Definition
- An Encoding Scheme for Spatial RDF
- Query Evaluation & Optimization
- Experiments
- Conclusions

Resource Description Framework (RDF)

- A very simple graph model

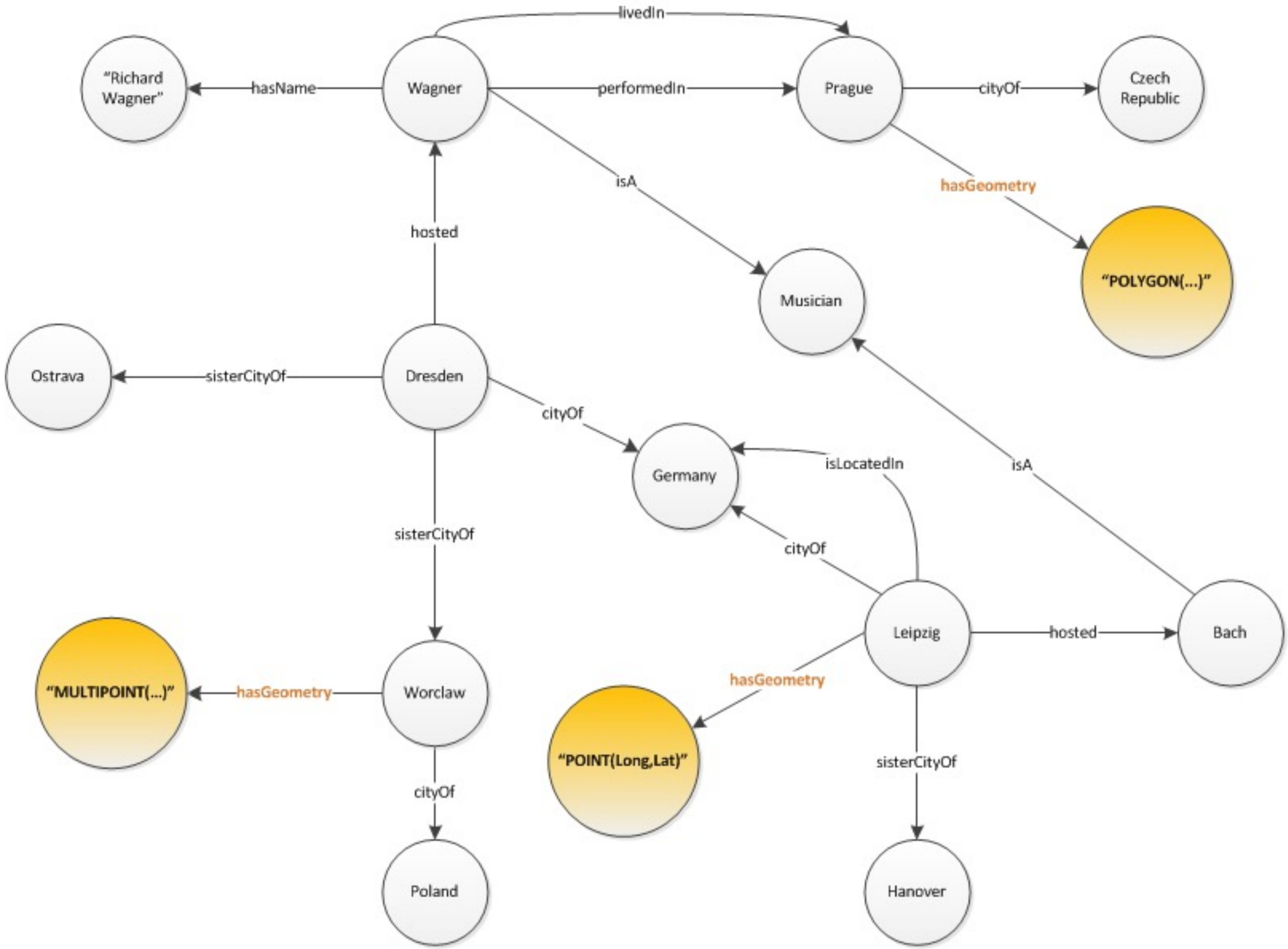
- Data are modeled as triples: < *subject predicate object* >

- *Subjects* and *Objects* are resources (entities)

- *Predicates* (aka *properties*) are relations between subjects and objects
  E.g., < Berlin isCapitalOf Germany >

- Good for data that do not have a crisp schema
  - Each subject can have its own set of properties

RDF Graph Example



Querying RDF Data

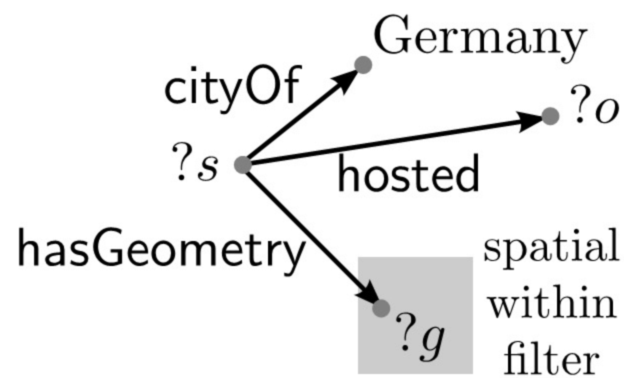
- SPARQL Language

- Queries are expressed in SQL-like syntax:

  `Select` [projection clause]
  `From` [graph model]
  `Where` [graph pattern]
  `Filter` [condition]

- We focus on queries having a spatial predicate in the `Filter` condition

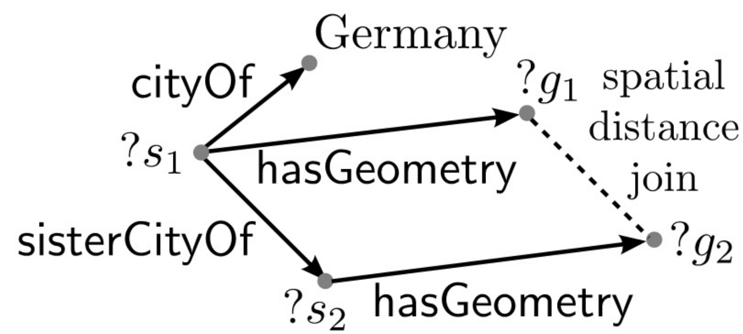
Queries with Spatial Range Filters




```
Select ?s ?o
From dataset
Where
{
    ?s cityOf Germany .
    ?s hosted ?o .
    ?s hasGeometry ?g .
    Filter WITHIN(?g, “POLYGON(...)”)
}
```

Queries with Spatial Distance Filters

Select ?s ?o
From dataset
Where
{
  $?s_1$ cityOf “Germany” .
  $?s_1$ sisterCityOf $?s_2$.
  $?s_1$ hasGeometry $?g_1$.
  $?s_2$ hasGeometry $?g_2$.
  Filter DISTANCE($?g_1$,$?g_2$) < “300km”
}



Problems in Query Evaluation

- In existing systems the spatial predicates are evaluated with the use of an R-tree combined with traditional spatial join algorithms

- The previous approach has the following drawbacks:
  - The spatial predicate is evaluated separately from the rest of the query
  - The results of the R-tree scan and the spatial join operators are not sorted on the entities' IDs
  - Query evaluation cannot benefit from the particular physical design of the native RDF stores

RDF Stores

- Create a Dictionary from strings to IDs
- Store triples in a single table

subject	*property*	*object*
Dresden	cityOf	Germany
Prague	cityOf	CzechRepublic
Leipzig	cityOf	Germany
Wrocław	cityOf	Poland
Dresden	sisterCityOf	Wrocław
Dresden	sisterCityOf	Ostrava
Leipzig	sisterCityOf	Hannover
Dresden	hosted	Wagner
Leipzig	hosted	Bach
Wagner	hasName	“Richard Wagner”
Wagner	performedIn	Leipzig
Wagner	performedIn	Prague
Dresden	hasGeometry	“POINT (...)”
Prague	hasGeometry	“POINT (...)”
Leipzig	hasGeometry	“POINT (...)”
...	...	...

Input Dataset

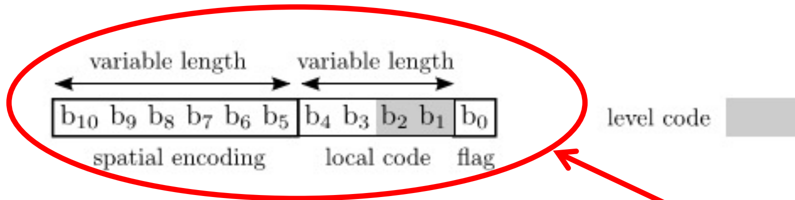
ID	*URI/literal*
1	Dresden
2	cityOf
3	Germany
4	Prague
5	CzechRepublic
6	Leipzig
...	...

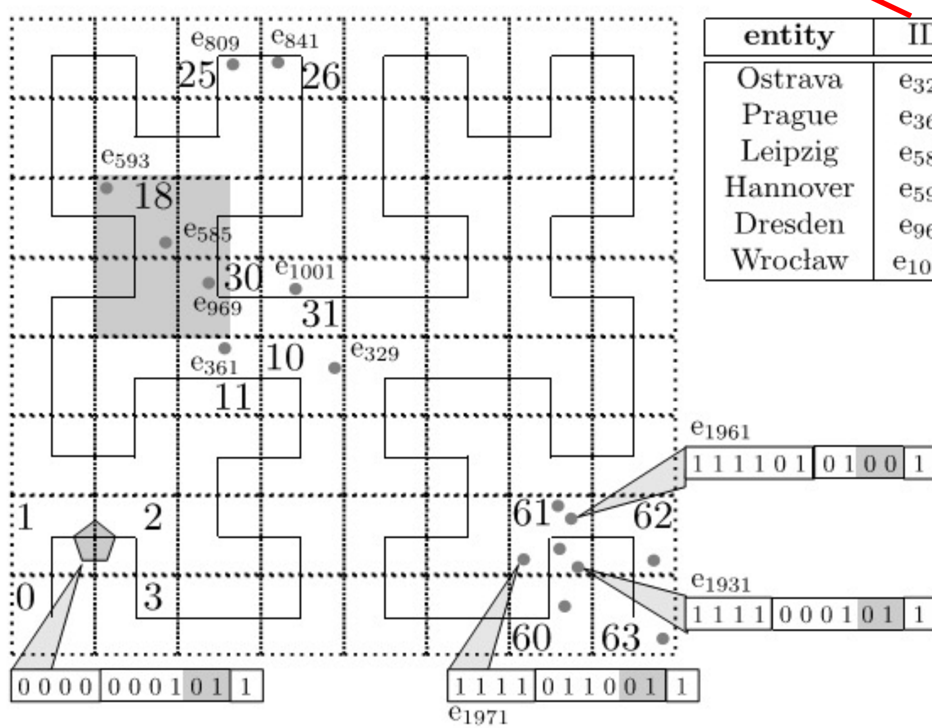
Dictionary

subject	*property*	*object*
1	2	3
4	2	5
6	2	3
...	...	...

Triples’ Table

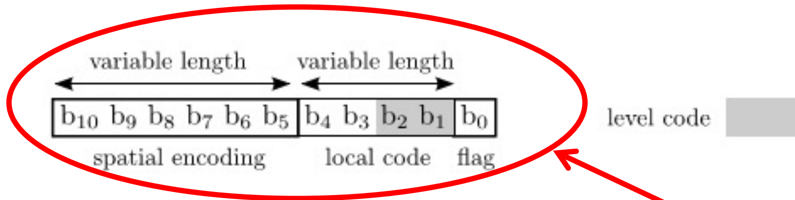
Encoding Scheme

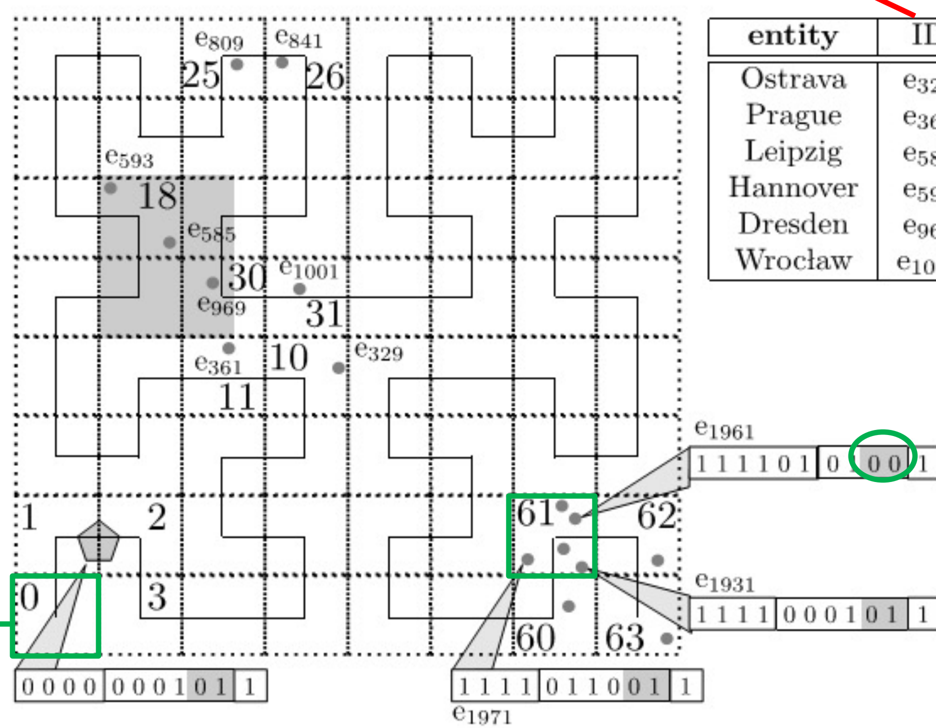




entity	ID
Ostrava	e_{329}
Prague	e_{361}
Leipzig	e_{585}
Hannover	e_{593}
Dresden	e_{969}
Wrocław	e_{1001}

Encoding Scheme

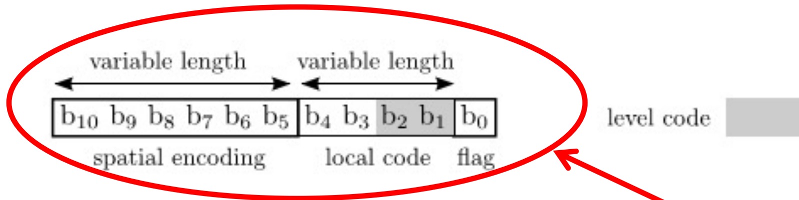


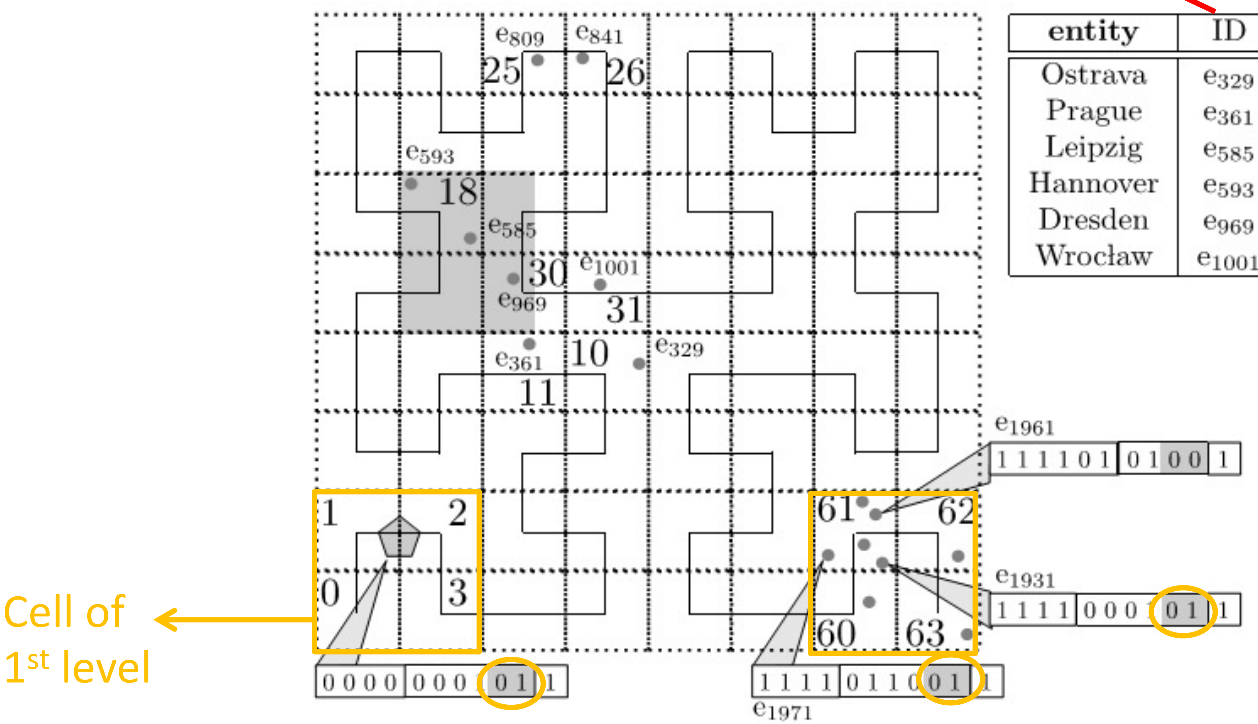


entity	ID
Ostrava	e_{329}
Prague	e_{361}
Leipzig	e_{585}
Hannover	e_{593}
Dresden	e_{969}
Wrocław	e_{1001}

Cell of bottom level

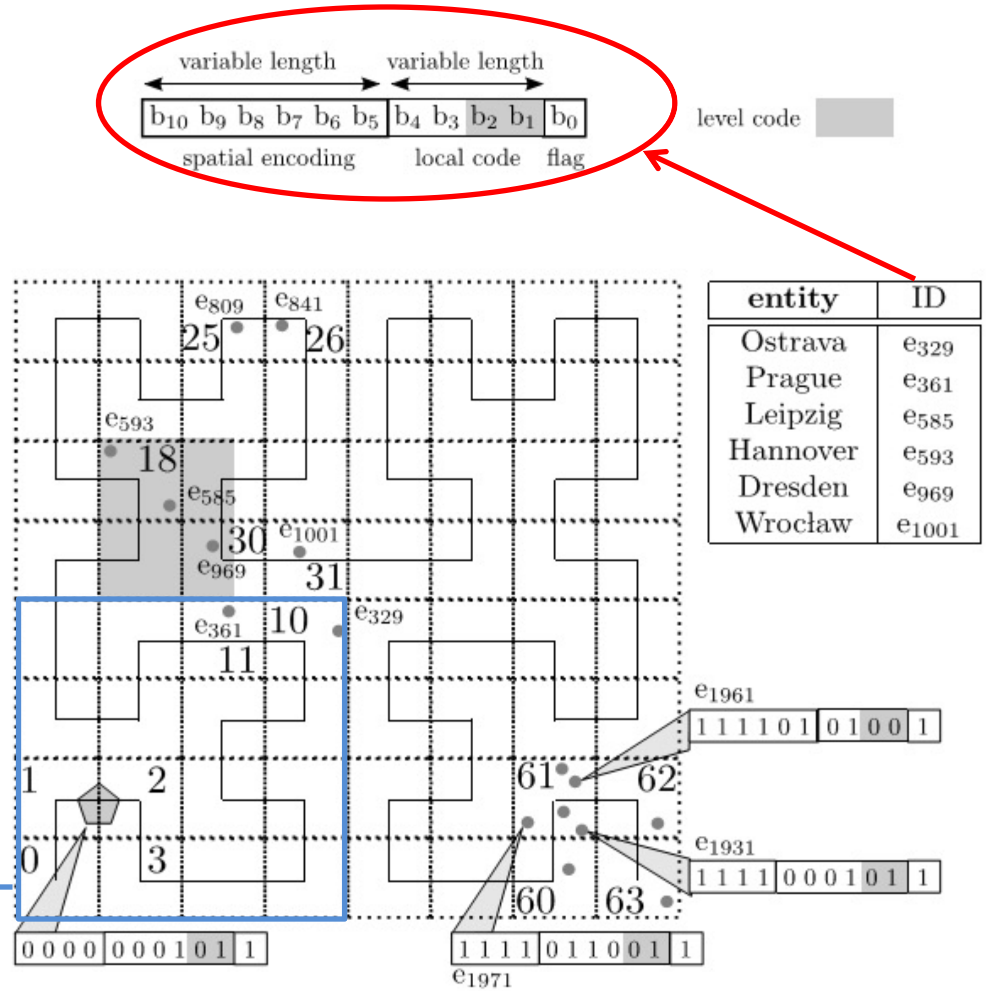
Encoding Scheme





entity	ID
Ostrava	e_{329}
Prague	e_{361}
Leipzig	e_{585}
Hannover	e_{593}
Dresden	e_{969}
Wrocław	e_{1001}

Encoding Scheme

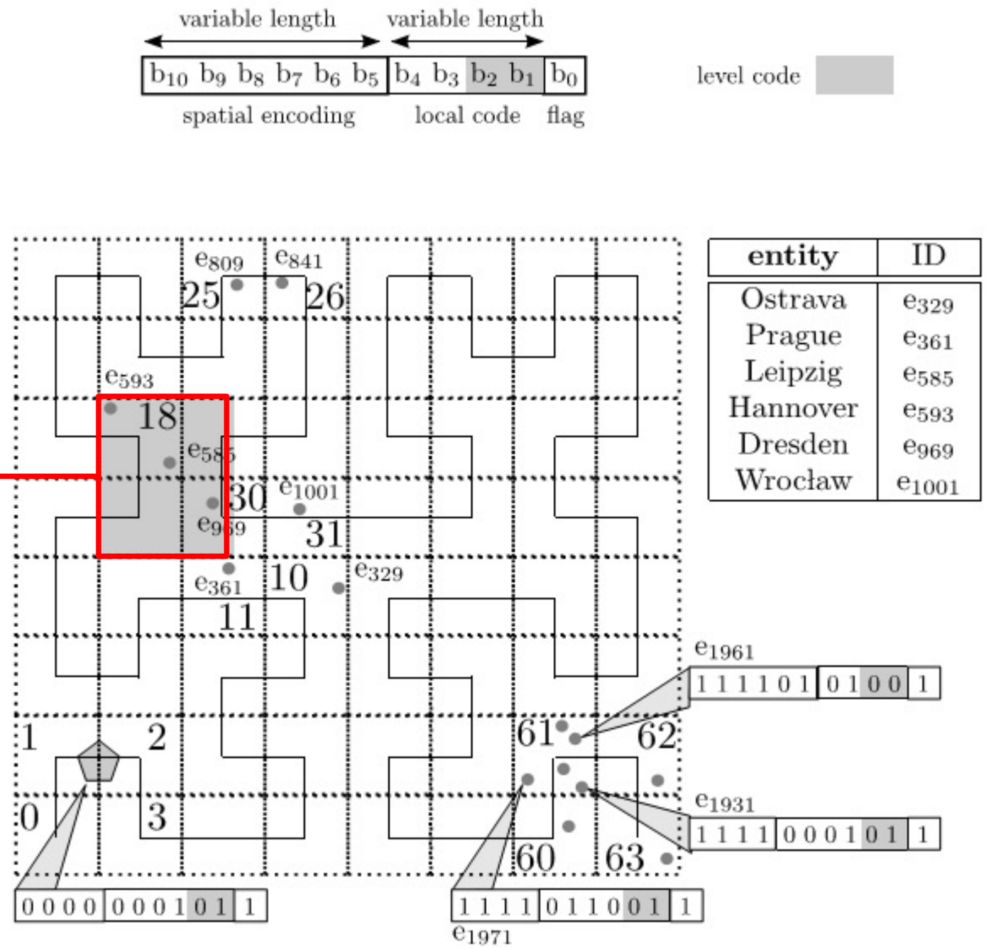


entity	ID
Ostrava	e_{329}
Prague	e_{361}
Leipzig	e_{585}
Hannover	e_{593}
Dresden	e_{969}
Wrocław	e_{1001}

Cell of 2nd level

Spatial Range Filtering on the Encoding

Spatial Range

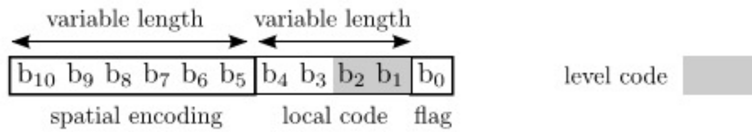


entity	ID
Ostrava	e_{329}
Prague	e_{361}
Leipzig	e_{585}
Hannover	e_{593}
Dresden	e_{969}
Wrocław	e_{1001}

Steps:

1. Extract cell ID from subject ID
2. Get cell's coordinates from the grid
3. Check if cell is contained in the given window

Spatial Range Filtering on the Encoding



Verified entities:

e_{593}

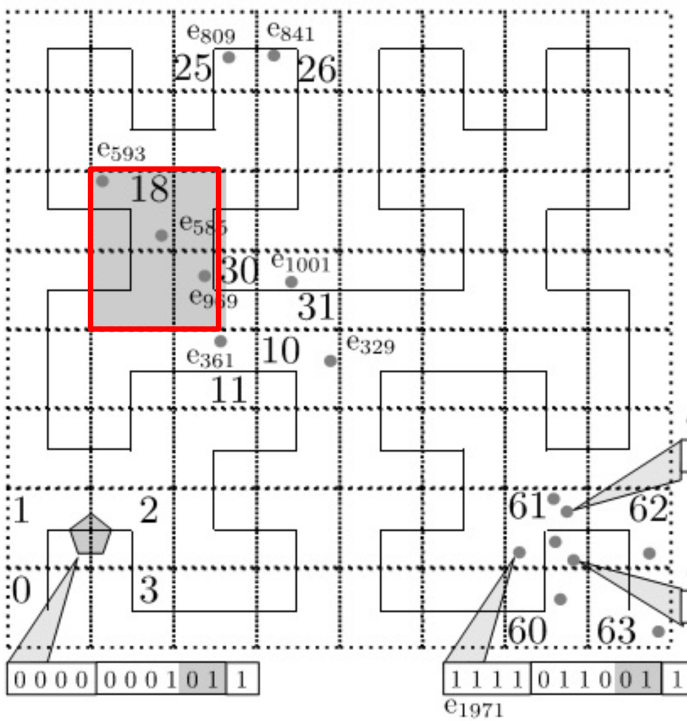
e_{585}

Non-verified entities:

e_{969}

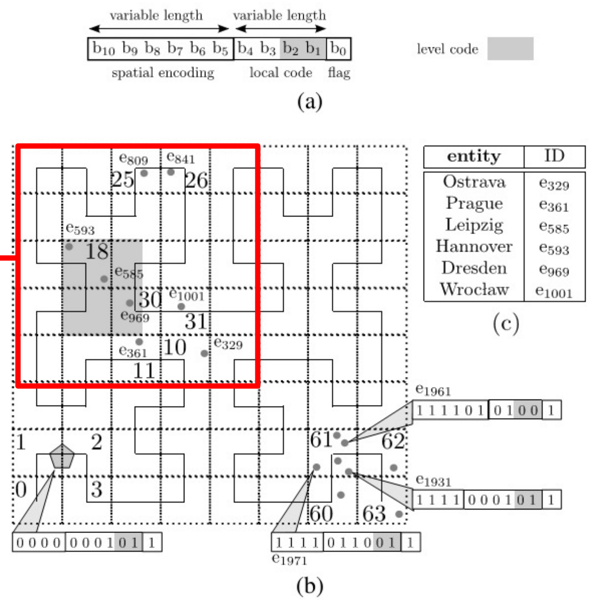
Filtered entities:

All input entities whose cells are out of the given range



entity	ID
Ostrava	e_{329}
Prague	e_{361}
Leipzig	e_{585}
Hannover	e_{593}
Dresden	e_{969}
Wrocław	e_{1001}

Spatial Merge Join on the Encoding



Neighbours cells of e_{585}

Steps:

1. Extract cell ID from subject ID on the left
2. Compute neighbour cells using the grid
3. Spool tuples in the buffer till we reach an entity that is out of the neighbours' range
4. Output the join pairs
5. Check if tuples from the buffer can be discarded
6. Continue with the next tuple on the left

Neighbours of e_{585}

spatial merge-join

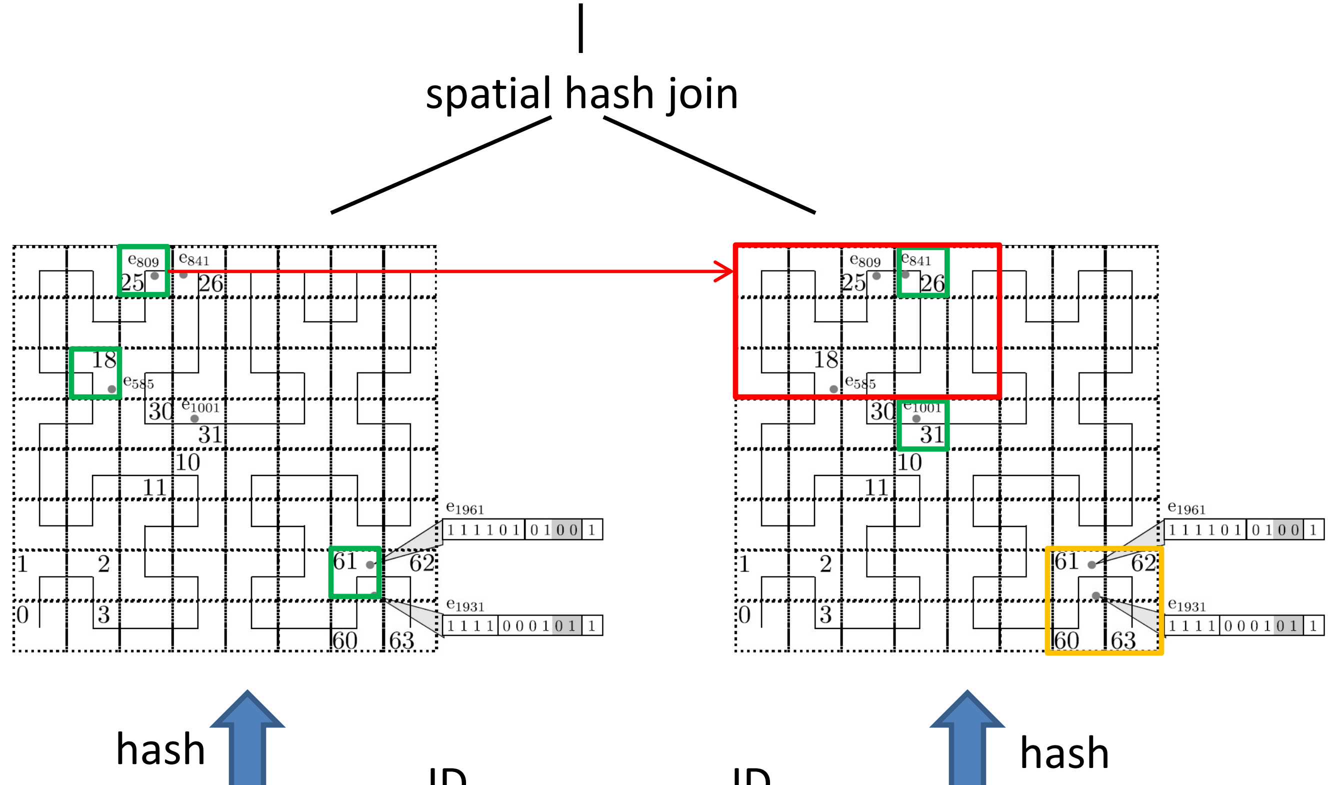
Range of possible neighbours in the right buffer

	minNeighborID	*maxNeighborID*	*minChildID*	*ID*
t_1	10	31	18	585
t_2	18	37	25	809
t_3	48	63	60	1961
	…	…	…	…

Buffer

minChlidID	*maxChlidID*	*maxNeighborID*	*ID*	
26	26	39	841	t'_1
31	31	55	1001	t'_2
60	63	63	1931	t'_3
…	…	…	…	

Spatial Hash Join on the Encoding



					ID
t_1	…	…	…	…	809
t_2	…	…	…	…	585
…	…	…	…	…	1961
	…	…	…	…	…

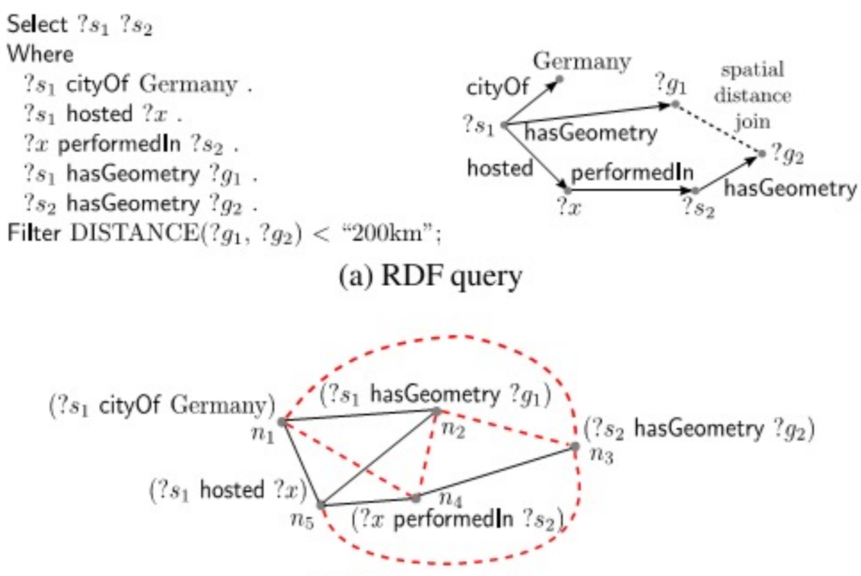
Left Relation

	ID				
t'_1	841	…	…	…	…
t'_2	1931	…	…	…	…
…	1001	…	…	…	…
	…	…	…	…	…

Right Relation

Query Optimization

- Use the grid for selectivity estimation
- Assume
  - Uniform spatial distribution inside each cell
  - Independence between the spatial and the RDF parts of the query
- Expand query graph with the additional edges that denote a join based on the spatial encoding

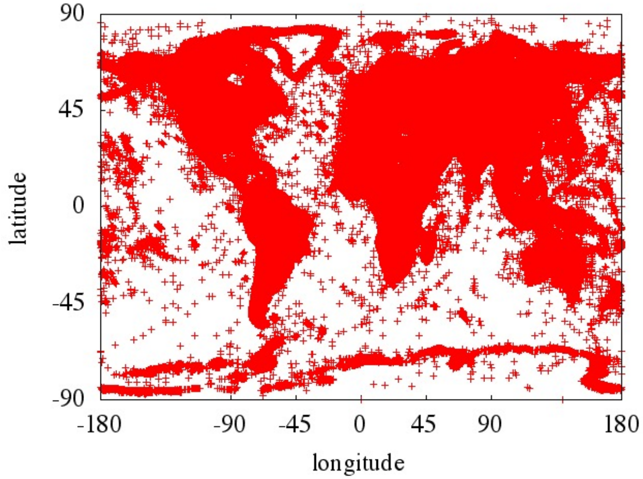


(a) RDF query

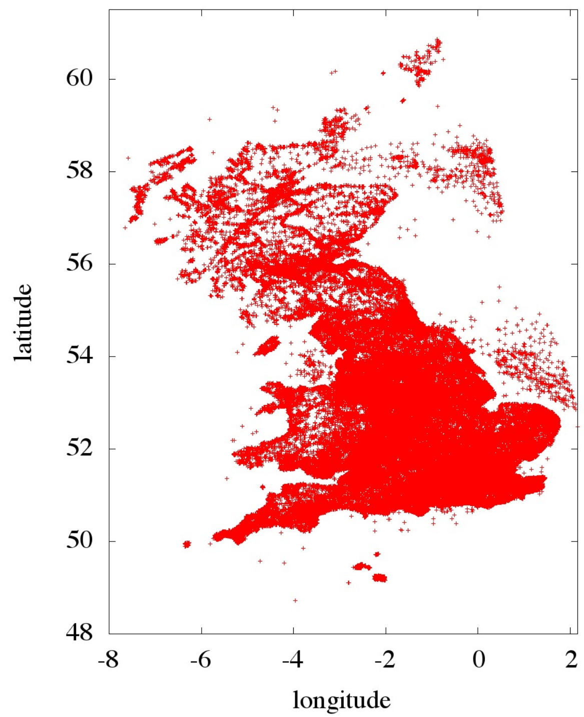
(b) Join graph G_Q

Experimental Evaluation

YAGO2



Linked Geodata (OpenStreetMap)



Dataset	Triples	Entities	Points	Polygons	Lines	Multipoints
LGD (3 Gb)	15.4M	10.6M	590K	264K	2.6M	0
YAGO2 (22 Gb)	205.3M	108.5M	4M	0	0	780K

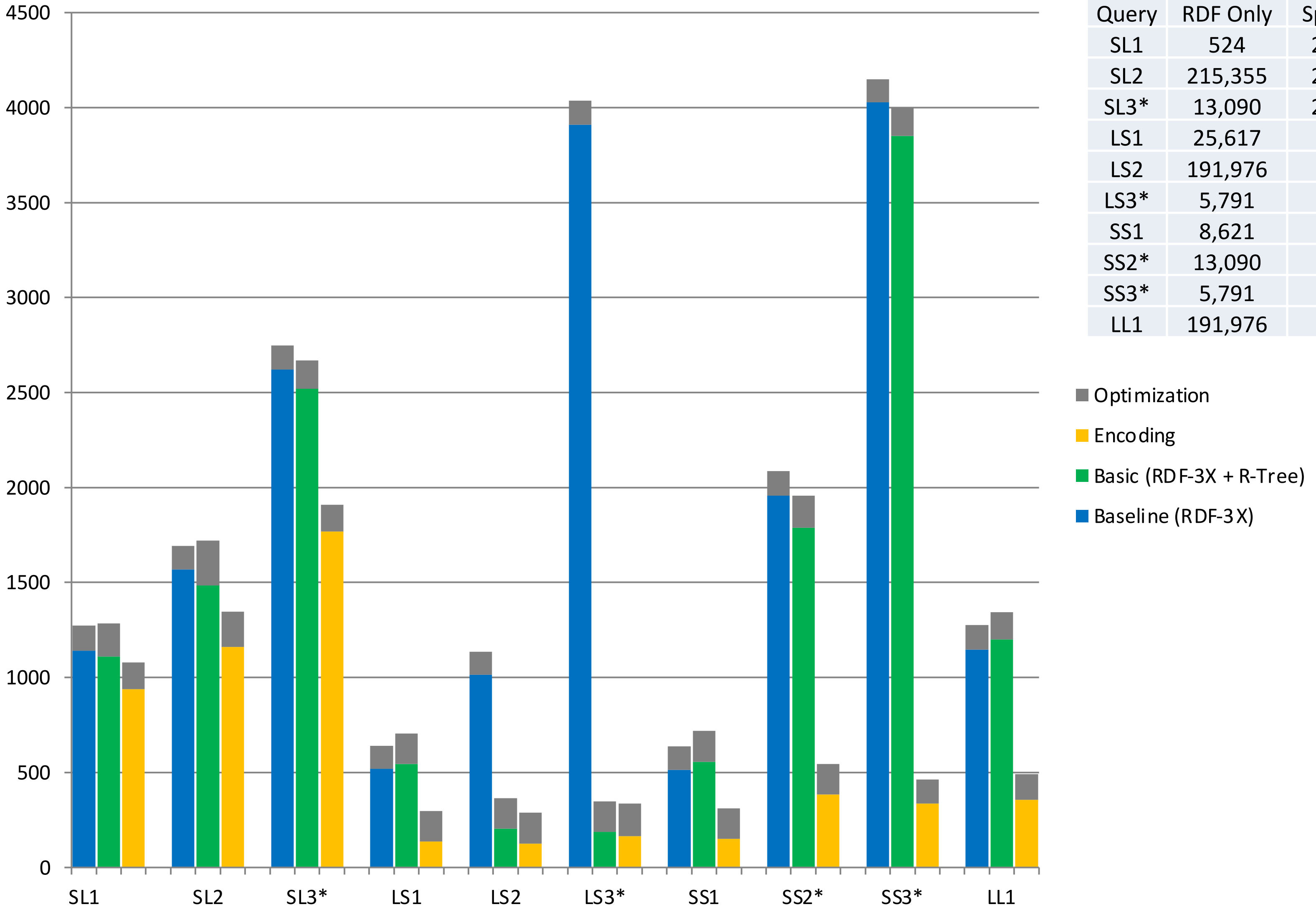
Level	0 (bottom)	1	2	3	4	5	6	≥ 7
LGD	42.7	13.7	13.2	11.1	7.9	5.1	3.0	3.3
YAGO2	50.3	19.2	8.1	4.5	3.0	2.4	1.9	10.6

Grid used: 8192 x 8192
Number of cells: ~89M

Queries Used in the Experiments

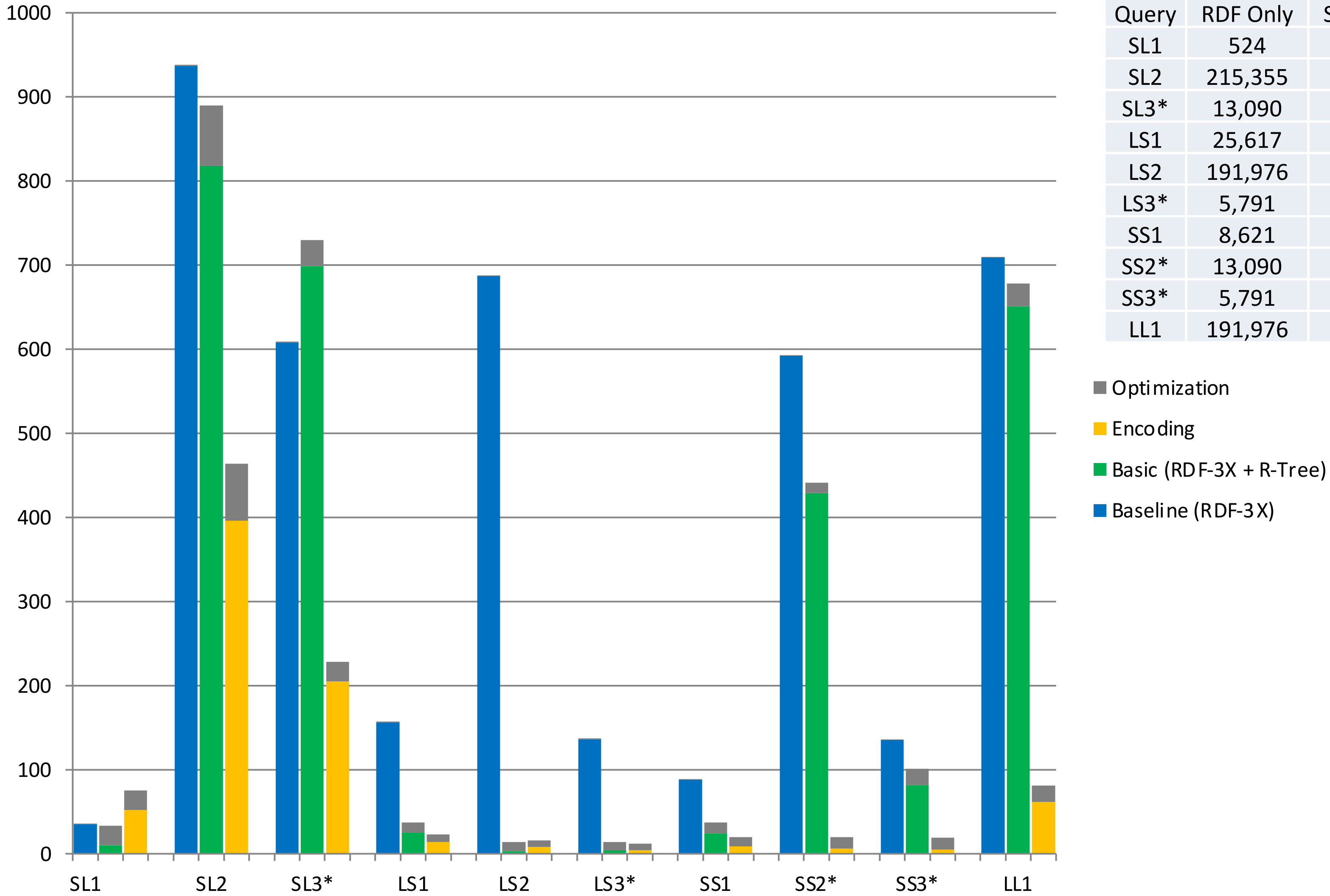
- Queries with WITHIN predicates:
  - **SL** (RDF part is selective – spatial part is not)
  - **LS** (Spatial part is selective – RDF part is not)
  - **SS** (RDF and spatial parts are both selective)
  - **LL** (RDF and spatial parts are both not selective)
- Queries with DISTANCE predicates:
  - Varying distance thresholds from ten to thousands kms
  - Connected and not connected graph patterns

Queries with Spatial Range Filters on LGD (cold cache)



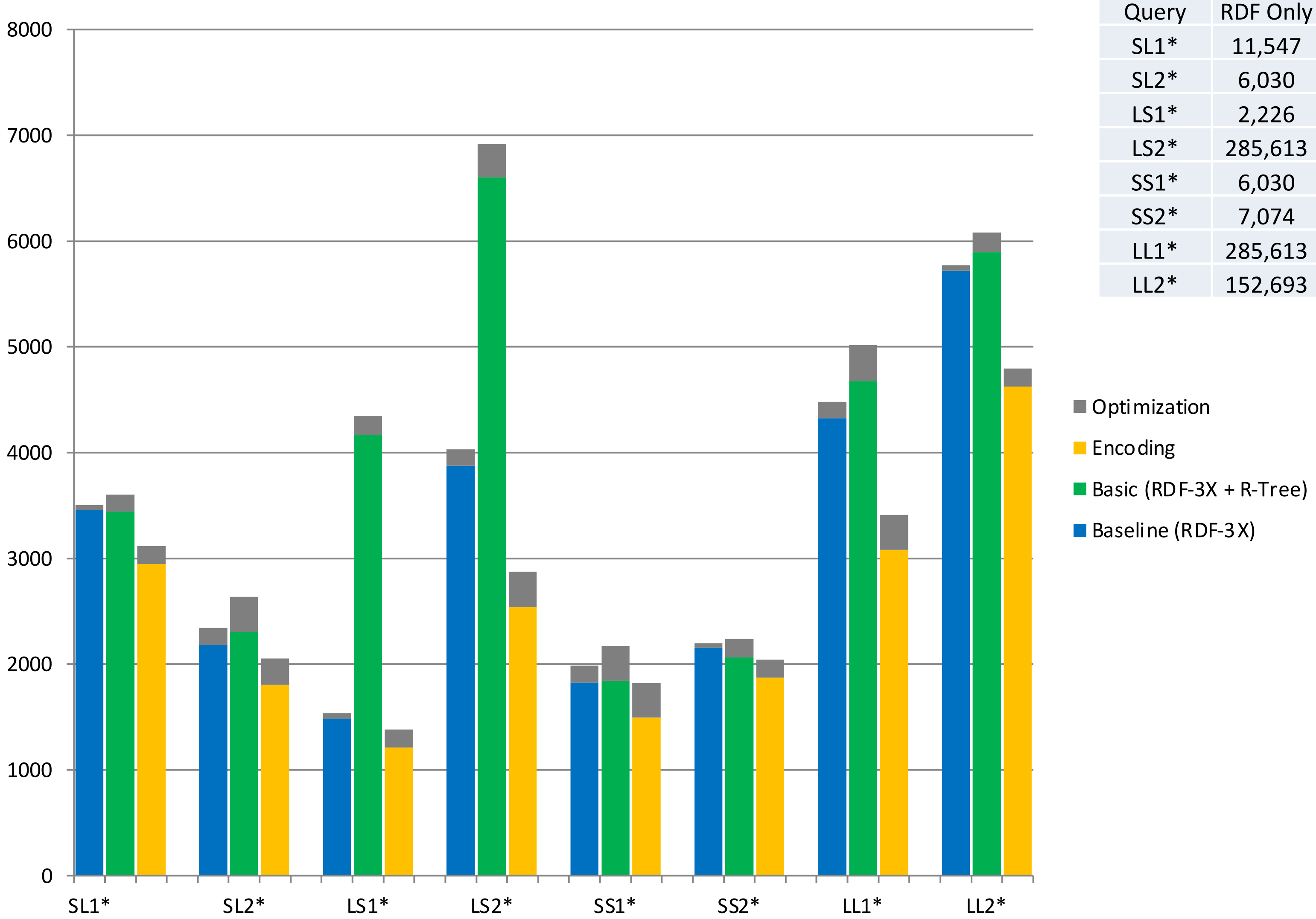
Query	RDF Only	Spatial Only	Combined
SL1	524	2,537,757	411
SL2	215,355	2,943,209	186,302
SL3*	13,090	2,537,757	9814
LS1	25,617	9,002	86
LS2	191,976	908	3
LS3*	5,791	908	9
SS1	8,621	9,002	69
SS2*	13,090	9,002	120
SS3*	5,791	9,002	7
LL1	191,976	350,405	13,416

Queries with Spatial Range Filters on LGD (warm cache)



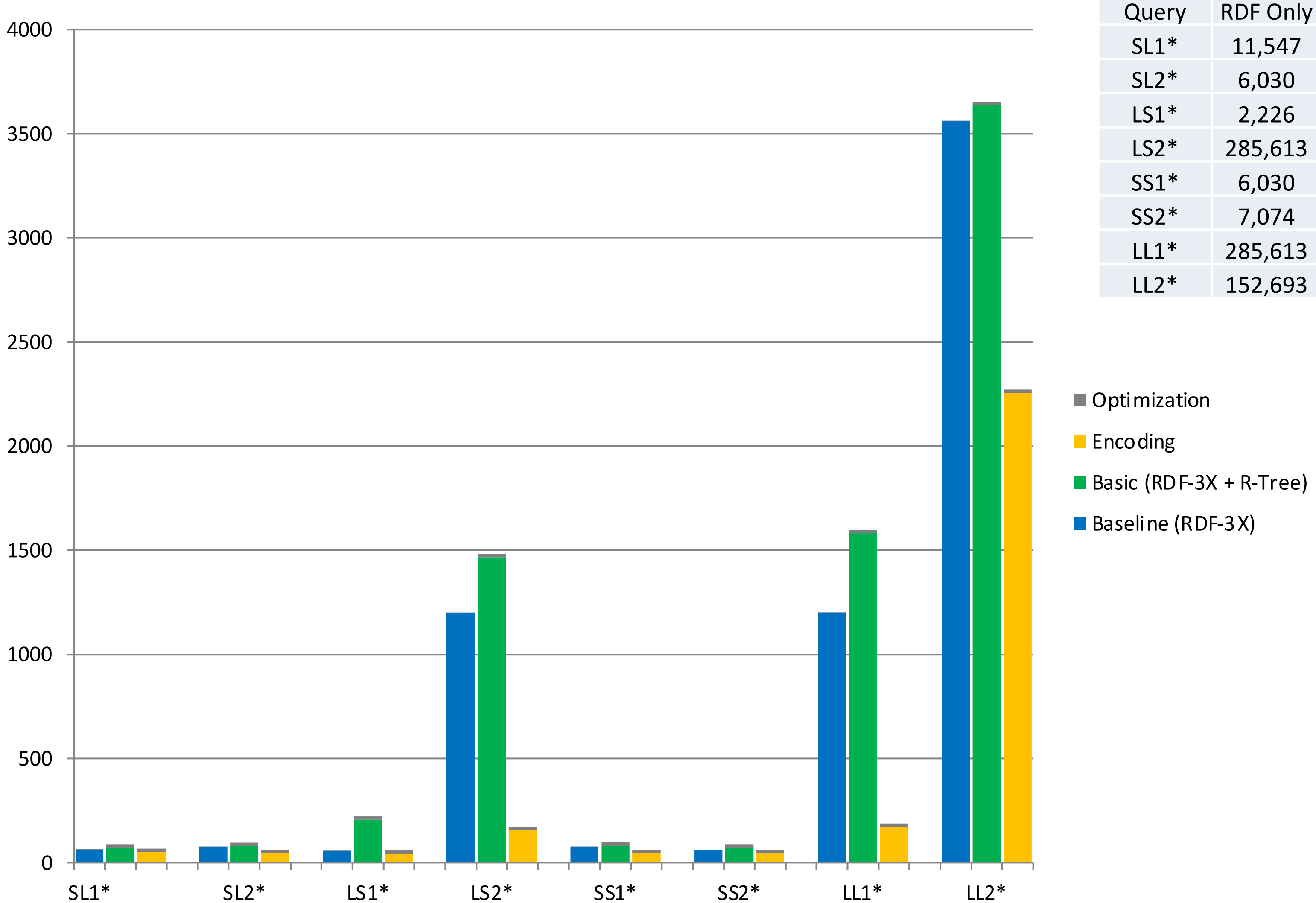
Query	RDF Only	Spatial Only	Combined
SL1	524	2,537,757	411
SL2	215,355	2,943,209	186,302
SL3*	13,090	2,537,757	9814
LS1	25,617	9,002	86
LS2	191,976	908	3
LS3*	5,791	908	9
SS1	8,621	9,002	69
SS2*	13,090	9,002	120
SS3*	5,791	9,002	7
LL1	191,976	350,405	13,416

Queries with Spatial Range Filters on YAGO2 (cold cache)



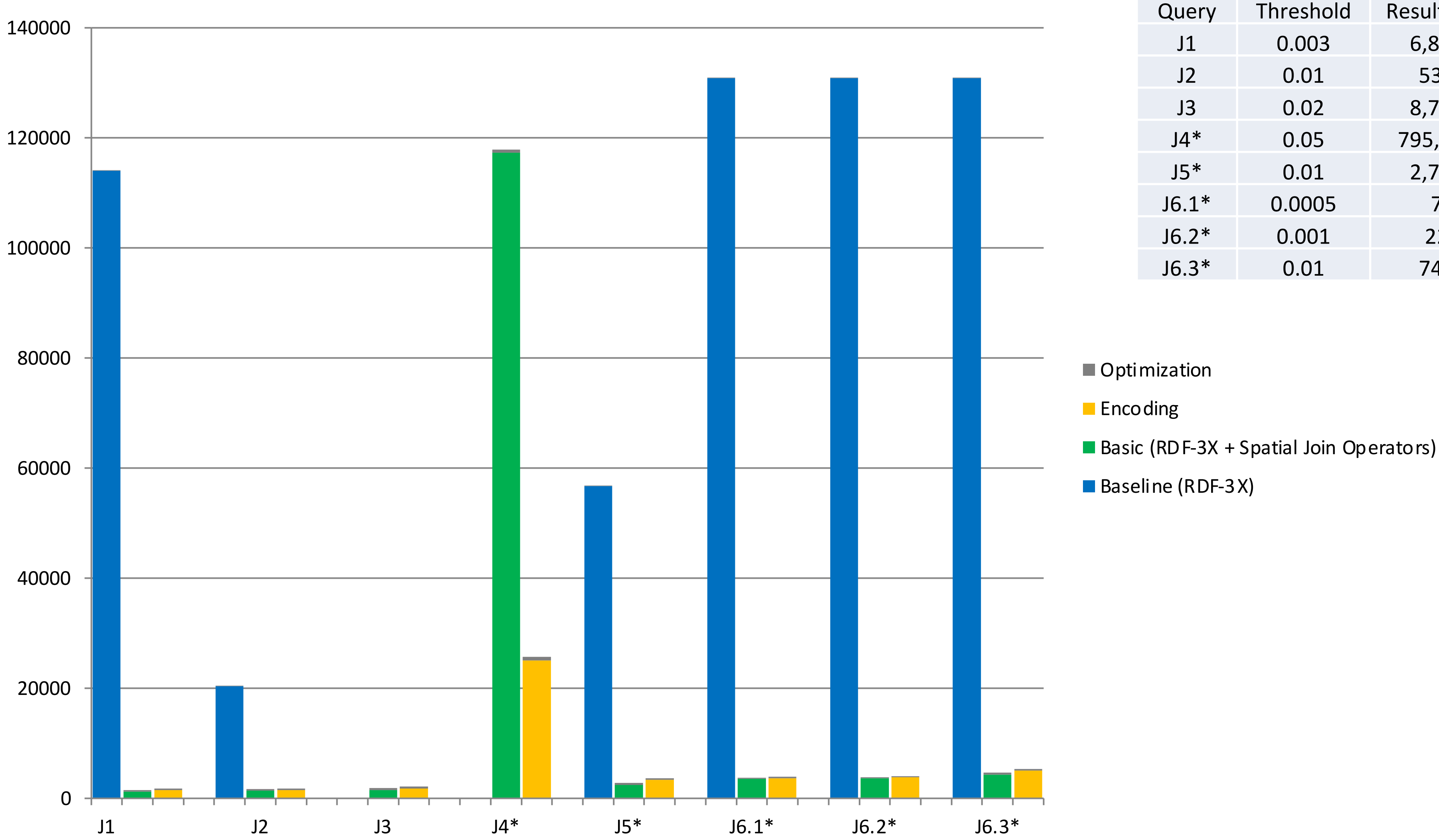
Query	RDF Only	Spatial Only	Combined
SL1*	11,547	364,992	891
SL2*	6,030	31,260	69
LS1*	2,226	138	7
LS2*	285,613	41,945	4,471
SS1*	6,030	8,440	3
SS2*	7,074	7,042	32
LL1*	285,613	184,743	10,454
LL2*	152,693	107,625	88

Queries with Spatial Range Filters on YAGO2 (warm cache)



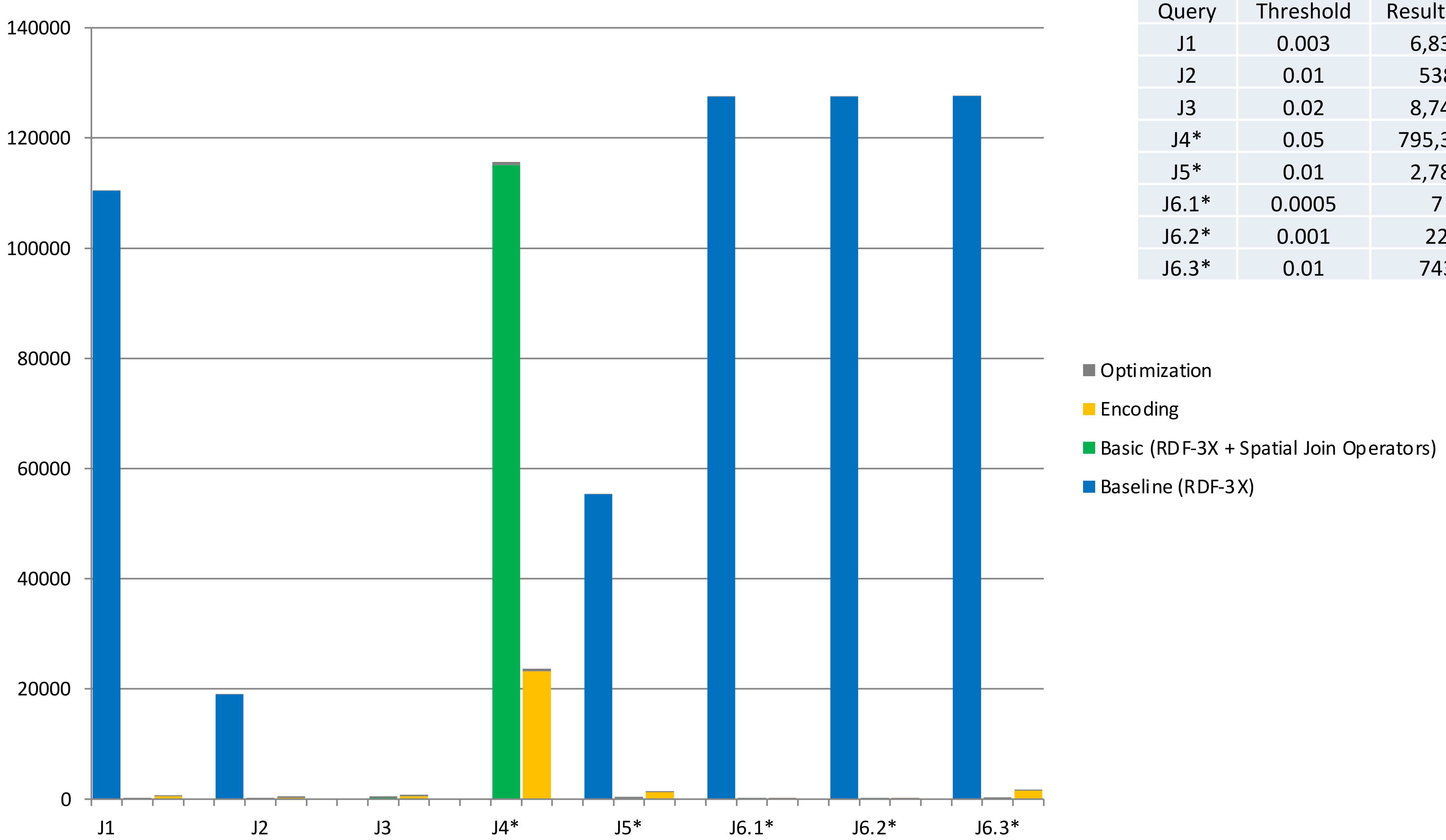
Query	RDF Only	Spatial Only	Combined
SL1*	11,547	364,992	891
SL2*	6,030	31,260	69
LS1*	2,226	138	7
LS2*	285,613	41,945	4,471
SS1*	6,030	8,440	3
SS2*	7,074	7,042	32
LL1*	285,613	184,743	10,454
LL2*	152,693	107,625	88

Queries with Spatial Distance Filters on LGD (cold cache)



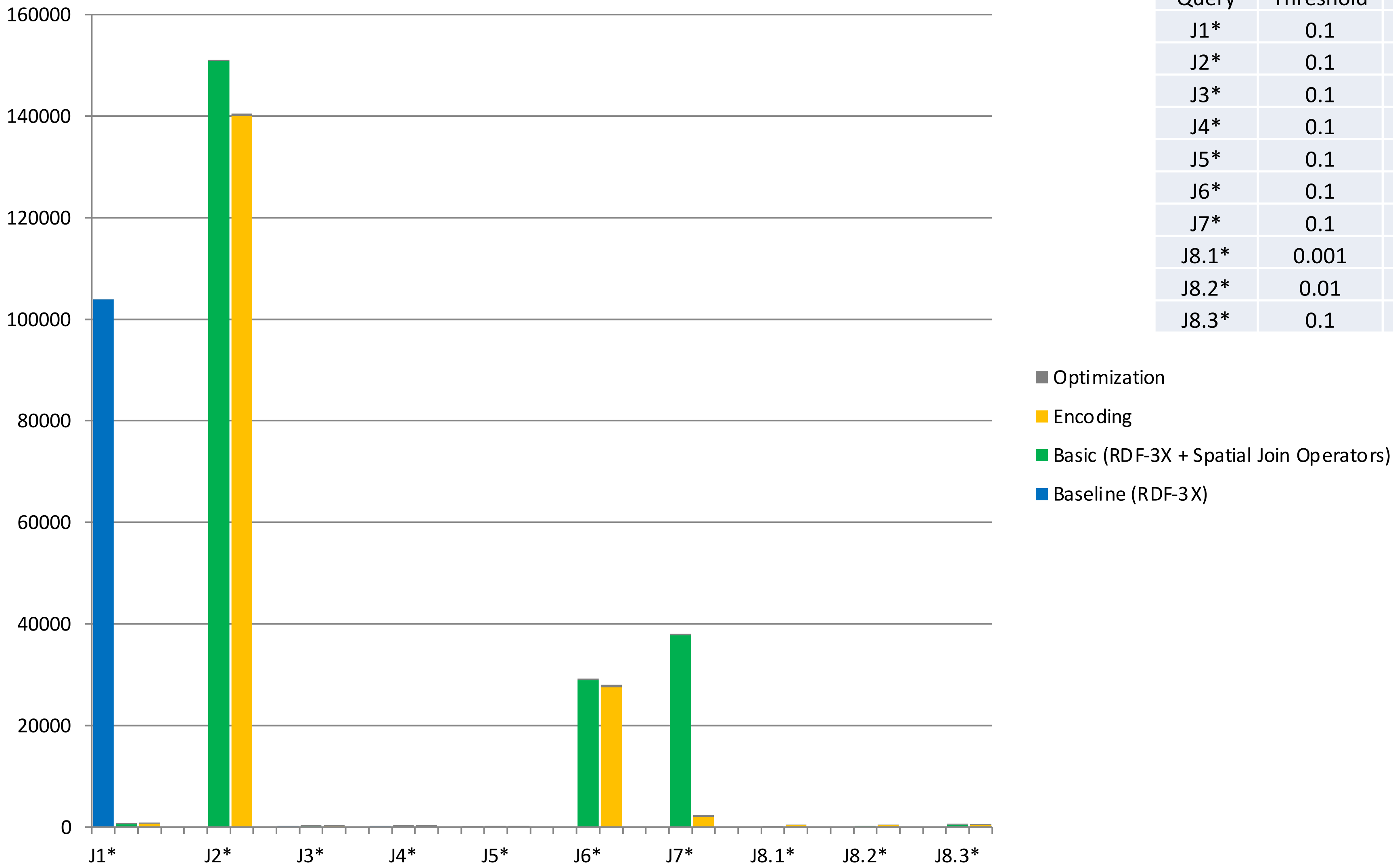
Query	Threshold	Result Size
J1	0.003	6,831
J2	0.01	538
J3	0.02	8,742
J4*	0.05	795,322
J5*	0.01	2,782
J6.1*	0.0005	7
J6.2*	0.001	22
J6.3*	0.01	743

Queries with Spatial Distance Filters on LGD (warm cache)



Query	Threshold	Result Size
J1	0.003	6,831
J2	0.01	538
J3	0.02	8,742
J4*	0.05	795,322
J5*	0.01	2,782
J6.1*	0.0005	7
J6.2*	0.001	22
J6.3*	0.01	743

Queries with Spatial Distance Filters on YAGO2 (warm cache)



Query	Threshold	Result Size
J1*	0.1	2,635
J2*	0.1	6,799,189
J3*	0.1	832
J4*	0.1	451
J5*	0.1	113
J6*	0.1	664,613
J7*	0.1	4,204,184
J8.1*	0.001	85,188
J8.2*	0.01	86,222
J8.3*	0.1	131,828

Conclusions

- The encoding-based approach can be easily incorporated into any triple store
- The average performance gains of the Encoding-based approach with respect to the Basic approach are:
  - Queries with WITHIN predicates:
    - LGD: **53%** with cold cache and **68%** with warm cache
    - YAGO2: **35%** with cold cache and **60%** with warm cache
  - Queries with DISTANCE predicates:
    - LGD: **65%** with cold cache and **75%** with warm cache
    - YAGO2: **19%** with cold cache **21%** with warm cache
- The overhead in the optimization time is negligible with respect to the overall response time