An Efficient Query Planner for the QLever SPARQL Engine

Mahmoud Khalaf

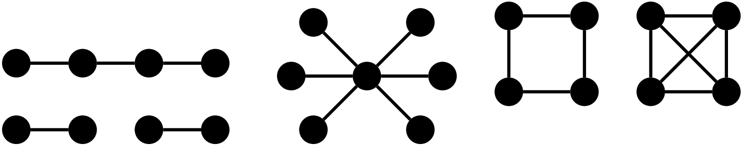
Albert-Ludwigs-Universität Freiburg
Algorithms and Data Structures

04.04.2025

Qlever's Query Planner is optimal

but can be very slow when the query graph is very large.

Query Graph



(a) Chain (b) Star (c) Cycle (d) Clique

Figure: Query Graph Shapes[1]

[1] Moerkotte, *Building Query Compilers*, ch. 3.2.

SPARQL Star Query

SPARQL query composed of six triple patterns grouped in two star-shaped blocks[2]

```
SELECT ?person ?city WHERE {
    ?person :name ?name .
    ?person :birthDate ?bday .
    ?person :birthPlace ?city .
    ?city :label ?label .
    ?city :population ?population .
    ?city :country :Germany .
}
```

Listing: SPARQL star query

[2] Galkin et al., "SMJoin", P. 2.

SPARQL Star Query

SPARQL query composed of six triple patterns grouped in two star-shaped blocks[2]

```
SELECT ?person ?city WHERE {
    ?person :name ?name .
    ?person :birthDate ?bday .
    ?person :birthPlace ?city .
    ?city :label ?label .
    ?city :population ?population .
    ?city :country :Germany .
}
```

Listing: SPARQL star query

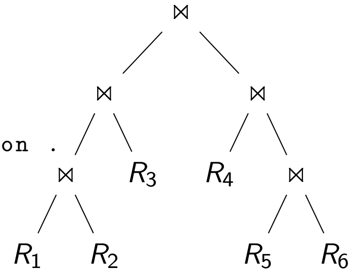


Figure: $((R_1 \bowtie R_2) \bowtie R_3) \bowtie (R_4 \bowtie (R_5 \bowtie R_6))$

[2] Galkin et al., "SMJoin", P. 2.

Query Planner

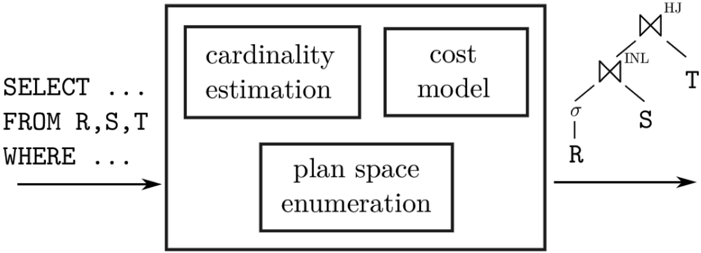


Figure: Query Planner Architecture[3]

[3] Leis et al., "How Good Are Query Optimizers, Really?"

Cardinality Estimation

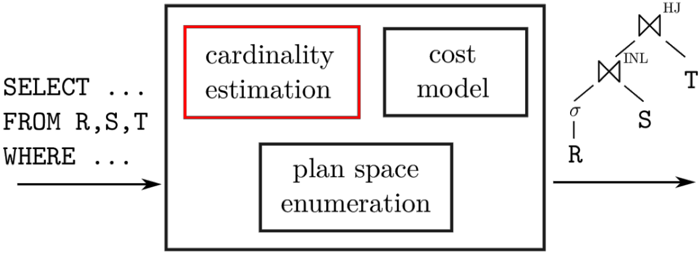


Figure: Query Planner Architecture[4]

[4]Leis et al., "How Good Are Query Optimizers, Really?"

Database Catalogue

- Histograms
- Sketches
- Sampling
- ...

Qlever way

For Join operations Qlever[5] compute the size as:

Size of intermediate results

$$s := \alpha \cdot m_a \cdot m_b \cdot \mathtt{min}(d_a, d_b)$$

[5] Bast and Buchhold, "QLever".

DocumentDB way

```c
/*-------------------------------------------------------------------------
* Copyright (c) Microsoft Corporation.  All rights reserved.
*
* src/planner/selectivity.c
*
* Implementation of selectivity functions for BSON operators.
*
*-------------------------------------------------------------------------
*/
#include <postgres.h>
#include <fmgr.h>


PG_FUNCTION_INFO_V1(bson_operator_selectivity);


/*
* bson_operator_selectivity returns the selectivity of a BSON operator
* on a relation.
*/
Datum
bson_operator_selectivity(PG_FUNCTION_ARGS)
{
    /* dumbest possible implementation: assume 1% of rows are returned */
    PG_RETURN_FLOAT8(0.01);
}
```

Listing: selectivity.c (documentdb)

[5] https://github.com/microsoft/documentdb/blob/main/pg_documentdb_core/src/planner/selectivity.c

Cost Model

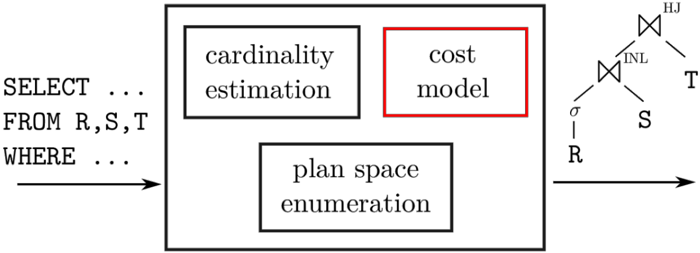


Figure: Query Planner Architecture[6]

[6] Leis et al., "How Good Are Query Optimizers, Really?"

Cost Function: Cout

$$C_{out}(R_i \bowtie R_j) = |R_i||R_j|f_{i,j} \tag{1}$$

$$|T| = \begin{cases} |R_i| & \text{if } T \text{ is a leaf } R_i \\ (\prod_{R_i \in T_1, R_j \in T_2} f_{i,j})|T_1||T_2| & \text{if } T = T_1 \bowtie T_2 \end{cases} \tag{2}$$

$$C_{out}(T) = \begin{cases} 0 & \text{if } T \text{ is a leaf of } R_i \\ |T| + C_{out}(T_1) + C_{out}(T_2) & \text{if } T = T_1 \bowtie T_2 \end{cases} \tag{3}$$

Plan Space Enumeration

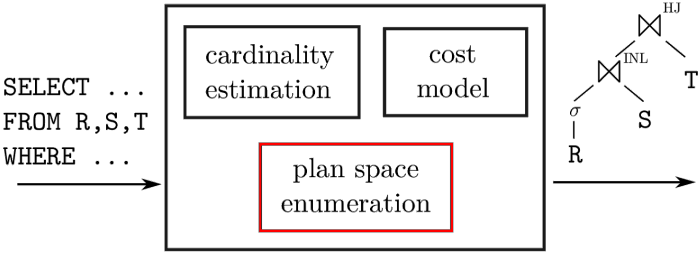


Figure: Query Planner Architecture[7]

[7] Leis et al., "How Good Are Query Optimizers, Really?"

IKKBZ I

Algorithm IKKBZ[8][9]

Require: G, C_H ▷ *an acyclic query graph, cost function*

$S \leftarrow \emptyset$
for $R_i \in R$ **do**
$G_i \leftarrow$ Precedence tree derived from G rooted at R_i
$S_i \leftarrow$ IKKBZ-Sub(G_i, C_H)
$S \leftarrow S \cup \{S_i\}$
end for

return $\text{argmin}_{S_i \in S} C_H(S_i)$ ▷ *optimal left-deep tree*

[8] Moerkotte, *Building Query Compilers*, p. 54.
[9] Neumann, "Query Optimization", p. 120-123.

IKKBZ II

Algorithm `IKKBZ-Sub`

Require: G, C_H ▷ *precedence graph G, cost function C_H*

while G_i is not a chain **do**

  $r \leftarrow$ a subtree of G_i whose subtrees are chains

  `IKKBZ-Normalize`(r)

  ▷ merge chains under r according to rank function

end while

`IKKBZ-Denormalize`(G_i) ▷ *optimal left-deep tree under G_i*

IKKBZ III

Algorithm `IKKBZ-Normalize`

Require: R ▷ *a subtree R of precedence graph G*

while $\exists r, c \in R, \texttt{rank}(r) > \texttt{rank}(c)$ **do**

▷ replace r, c by a compound relation r' that replace rc

end while ▷ *normalized subtree*

IKKBZ IV

Algorithm `IKKBZ-Denormalize`

Require: G ▷ *precedence graph (compound relations)*

while $\exists r \in R : r$ is a compund relation **do**

▷ replace r sequence of relations it represent

end while ▷ *denormalized precedence graph G*

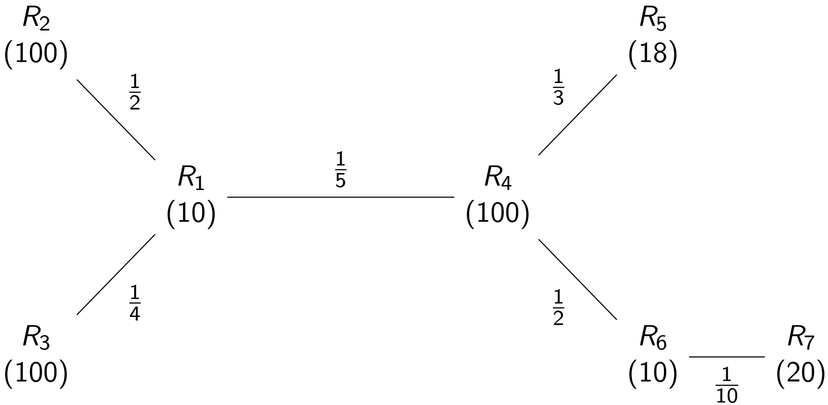


Figure: Query Graph

R	**n**	**s**	**C**	**T**	**rank**
R_1	10	0.20	2.00	2.00	0.50
R_2	100	0.50	50.00	50.00	0.98
R_3	100	0.25	25.00	25.00	0.96
R_4	100	0.20	20.00	20.00	0.95
R_5	18	0.33	6.00	6.00	0.83
R_6	10	0.50	5.00	5.00	0.80
R_7	20	0.10	2.00	2.00	0.50
$R_6 R_7$	200	0.05	15.00	10.00	0.60
$R_4 R_6 R_7$	20000	0.01	320.00	200.00	0.62

Table: Rank Computation

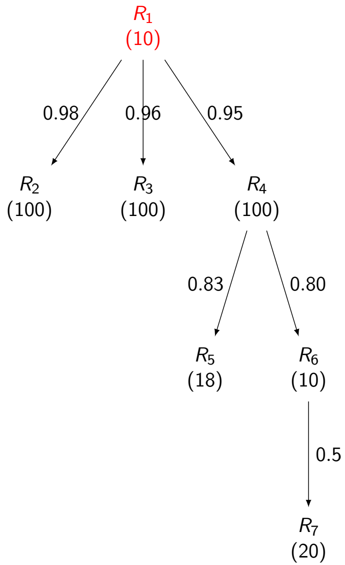


Figure: precedence tree rooted at R_1

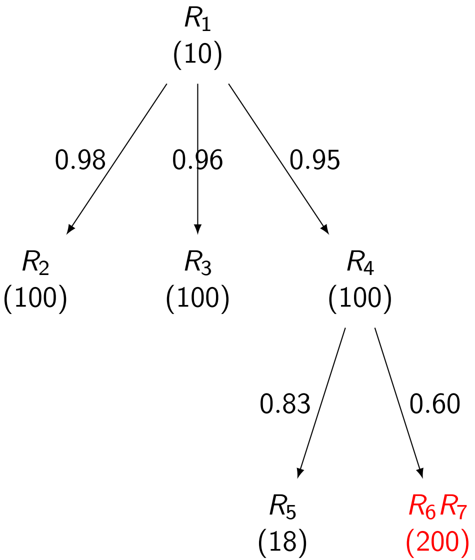


Figure: rank(R_6) $>$ rank(R_7), but $R_6 \rightarrow R_7$

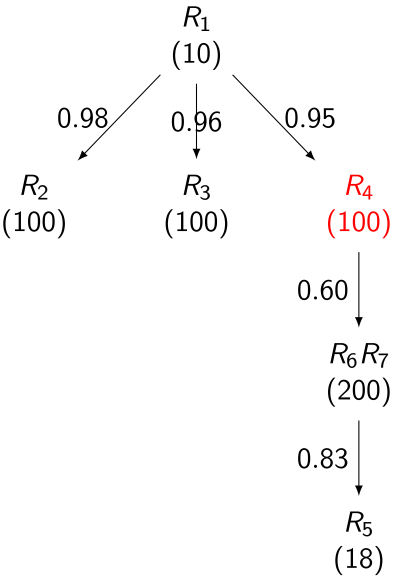


Figure: merging subchains of R_4

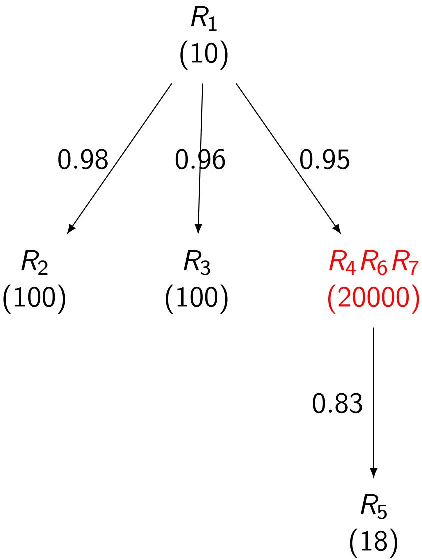


Figure: $\text{rank}(R_4) > \text{rank}(R_6, R_7)$, but $R_4 \rightarrow R_6 R_7$

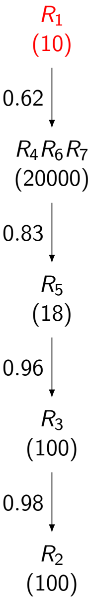


Figure: merging subchains of R_1

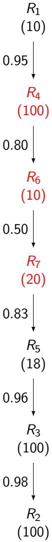


Figure: denormalized graph

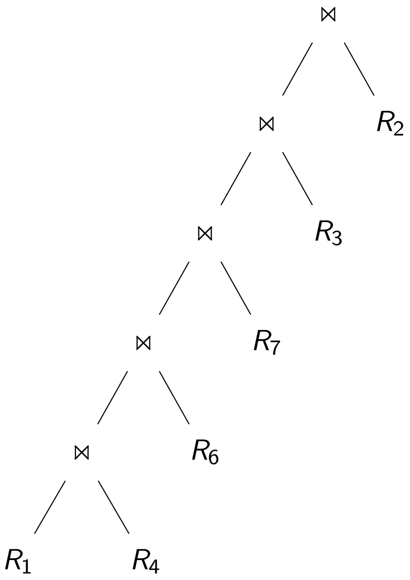


Figure: equivalent join tree

State Space Linearization

Algorithm linDP[10]

Require: $G(V, E), w, C_H$
$G' = \texttt{MST}(G, w)$
$O = \texttt{IKKBZ}(G', C_H)$
for $R_i \in R$ **do**
  $dp[i, i] \leftarrow R_i$ ▷ *init* n^2 *DPTable*
end for
for $s \in \{2, \ldots, |O|\}$ **do**
  for $i \in \{0, \ldots, |O| - s\}$ **do**
    for $j \in \{1, \ldots, |O| - s - 1\}$ **do**
      $L \leftarrow dp[i, i + j - 1]$ ▷ *left subplan*
      $R \leftarrow dp[i + s, i + s - 1]$ ▷ *right subplan*
      if L can join with R ▷ *existence of join predicate*
        $P \leftarrow L \bowtie R$ ▷ *current plan*
        if $C(P) < C(dp[i, i + s - 1])$
          $dp[i, i + s - 1] \leftarrow P$
    end for
  end for
end for

return $dp[0, |O| - 1]$ ▷ *sub-optimal bushy tree*

[10] Neumann and Radke, "Adaptive Optimization of Very Large Join Queries".

Conclusion

IKKBZ can be used on it's own to find an optimal-left deep plan in polynomial time which is good enough for most practical use-case.
Also,
Act as a stepping stone for a DP-based enumeration algorithm that results in an bushy plan in a reasonable amount of time.

Bibliography I

Bast, Hannah and Björn Buchhold. "QLever: A Query Engine for Efficient SPARQL+Text Search". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. CIKM '17: ACM Conference on Information and Knowledge Management. Singapore Singapore: ACM, June 11, 2017, pp. 647–656. ISBN: 978-1-4503-4918-5. DOI: 10.1145/3132847.3132921. URL: https://dl.acm.org/doi/10.1145/3132847.3132921.

Galkin, Mikhail et al. "SMJoin: A Multi-way Join Operator for SPARQL Queries". In: *Proceedings of the 13th International Conference on Semantic Systems*. Semantics2017. New York, NY, USA: Association for Computing Machinery, Sept. 11, 2017, pp. 104–111. ISBN: 978-1-4503-5296-3. DOI: 10.1145/3132218.3132220. URL: https://doi.org/10.1145/3132218.3132220.

Bibliography II

- Leis, Viktor et al. "How Good Are Query Optimizers, Really?" In: 9.3 (Nov. 1, 2015), pp. 204–215. DOI: 10.14778/2850583.2850594.
- Moerkotte, Guido. *Building Query Compilers*. Oct. 31, 2024. 720 pp. URL: https://pi3.informatik.uni-mannheim.de/~moer/querycompiler.pdf.
- Neumann, Thomas. "Query Optimization". Query Optimization (Technische Universität München). 2023. URL: https://db.in.tum.de/teaching/ws2324/queryopt/.

Bibliography III

Neumann, Thomas and Bernhard Radke. "Adaptive Optimization of Very Large Join Queries". In: *Proceedings of the 2018 International Conference on Management of Data*. SIGMOD/PODS '18: International Conference on Management of Data. Houston TX USA: ACM, May 27, 2018, pp. 677–692. ISBN: 978-1-4503-4703-7. DOI: 10.1145/3183713.3183733. URL: https://dl.acm.org/doi/10.1145/3183713.3183733.