Carnegie Mellon University

OPTIMIZE!

Database Query Optimization

Join Ordering: Bottom-Up

SPRING 2025 » SPECIAL TOPICS IN DATABASES » PROF. ANDY PAVLO

LAST CLASS

Transformation rules to generate and improve query plans.

- → Access Path
- → Inner Joins
- → Outer Joins
- → Group-By
- → Star / Snowflake Queries

OBSERVATION

Most queries with a join only target two tables.

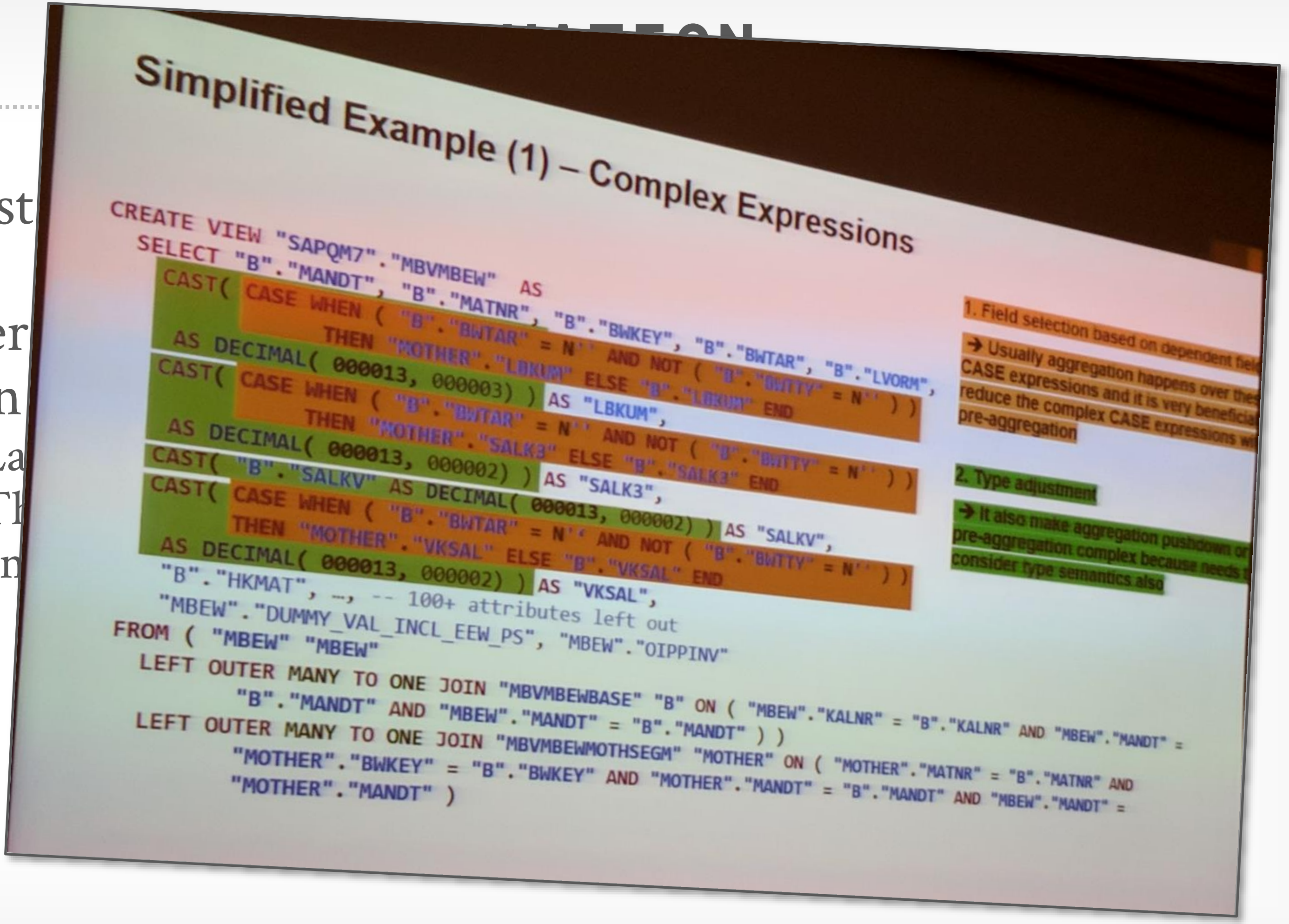
There are ridiculous outlier queries with 100s or even 1000s of tables.

→ Largest known query joins 5000 tables (SAP).

→ The most complex queries are generated from computers and not written by humans.

Most

Ther
even
→ La
→ Th
an



Simplified Example (1) – Complex Expressions

```
CREATE VIEW "SAPQM7"."MBVMBEW"  AS
  SELECT "B"."MANDT", "B"."MATNR", "B"."BWKEY", "B"."BWTAR", "B"."LVORM",
   CAST( CASE WHEN ( "B"."BWTAR" = N'' AND NOT ( "B"."BWTTY" = N'' ) )
            THEN "MOTHER"."LBKUM" ELSE "B"."LBKUM" END
   AS DECIMAL( 000013, 000003) ) AS "LBKUM",
   CAST( CASE WHEN ( "B"."BWTAR" = N'' AND NOT ( "B"."BWTTY" = N'' ) )
            THEN "MOTHER"."SALK3" ELSE "B"."SALK3" END
   AS DECIMAL( 000013, 000002) ) AS "SALK3",
   CAST( "B"."SALKV" AS DECIMAL( 000013, 000002) ) AS "SALKV",
   CAST( CASE WHEN ( "B"."BWTAR" = N'' AND NOT ( "B"."BWTTY" = N'' ) )
          THEN "MOTHER"."VKSAL" ELSE "B"."VKSAL" END
   AS DECIMAL( 000013, 000002) ) AS "VKSAL",
    "B"."HKMAT", …, -- 100+ attributes left out
    "MBEW"."DUMMY_VAL_INCL_EEW_PS", "MBEW"."OIPPINV"
FROM ( "MBEW" "MBEW"
  LEFT OUTER MANY TO ONE JOIN "MBVMBEWBASE" "B" ON ( "MBEW"."KALNR" = "B"."KALNR" AND "MBEW"."MANDT" =
       "B"."MANDT" AND "MBEW"."MANDT" = "B"."MANDT" ) )
  LEFT OUTER MANY TO ONE JOIN "MBVMBEWMOTHSEGM" "MOTHER" ON ( "MOTHER"."MATNR" = "B"."MATNR" AND
       "MOTHER"."BWKEY" = "B"."BWKEY" AND "MOTHER"."MANDT" = "B"."MANDT" AND "MBEW"."MANDT" =
       "MOTHER"."MANDT" )
```

1. Field selection based on dependent fie
→ Usually aggregation happens over thes
CASE expressions and it is very beneficia
reduce the complex CASE expressions w
pre-aggregation

2. Type adjustment
→ It also make aggregation pushdown or
pre-aggregation complex because needs t
consider type semantics also

OBSERVATION

Most queries with a join only target two tables.

There are ridiculous outlier queries with 100s or even 1000s of tables.

→ Largest known query joins 5000 tables (SAP).
→ The most complex queries are generated from computers and not written by humans.

An optimizer must be able to handle the common-case "easy" queries but still support the occasional freak queries.

TODAY'S AGENDA

Adaptive Join Optimization

Randomized Algorithms

ADAPTIVE JOIN OPTIMIZATION

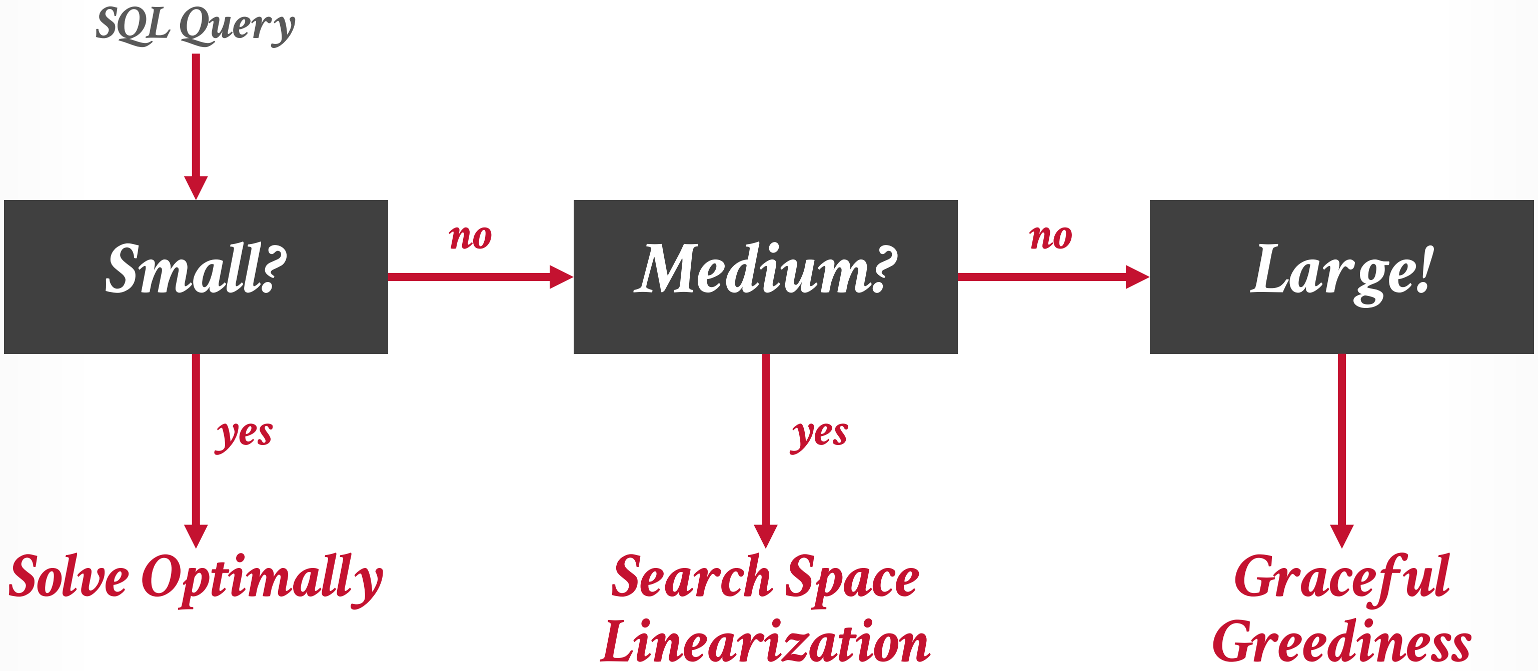
Instead of using a single search strategy for all queries, the optimizer can select a suitable algorithm per query based on the logical complexity.

Combine dynamic programming with search space linearization for small to medium queries.

For larger queries, degrade plan quality gracefully.

ADAPTIVE OPTIMIZATION OF VERY LARGE JOIN QUERIES
SIGMOD 2018

ADAPTIVE OPTIMIZATION DECISION TREE



Source: Thomas Neumann

OBSERVATION

The logical complexity of a query plan is not just the number of relations it references.

Instead, a query's complexity depends on the **structure** of its graph.
→ How the different relations join with each other.

QUERY GRAPH STRUCTURES

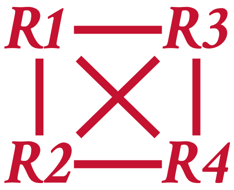
Chain Graph

- → Each relation is connected to at most two other relations.
- → Linear ordering of join precedence.
- → Best Case Scenario

```
SELECT * FROM R1
  JOIN R2 ON R2.r1_id = R1.id
  JOIN R3 ON R3.r2_id = R2.id
  JOIN R4 ON R4.r3_id = R3.id;
```

Clique Graph

- → Every relation is connected to all other relations.
- → These queries are nasty but rare.
- → Worst Case Scenario



```
SELECT * FROM R1, R2, R3, R4
 WHERE R1.id = R2.id
   AND R1.id = R3.id
   AND R1.id = R4.id
   AND R2.id = R3.id
   AND R2.id = R4.id
   AND R3.id = R4.id;
```

QUERY SIMPLIFICATION: GRACEFUL DEGRADATION FOR JOIN-ORDER OPTIMIZATION
SIGMOD 2009

SMALL QUERIES

Queries where the DP table is up 10,000 entries.
- → Chains: Up to 1000 relations
- → Cliques: Less than 14 relations

Run the **DPhyp** algorithm to generate the optimal join ordering.
- → Adapts to the query's graph structure
- → Completely and minimally enumerates all possible join orders without cross products.

Source: Thomas Neumann

DPHYP

Dynamic Programming Hypergraph (DPHyp)

Model the query as a hypergraph and then incrementally expand to enumerate new plans.

Algorithm Overview:

→ Iterate connected sub-graphs and incrementally add new edges to other nodes to complete query plan.

→ Use rules to determine which nodes the traversal is allowed to visit and expand.

Used in HyPer, Umbra, DuckDB, and GlareDB.

DYNAMIC PROGRAMMING STRIKES BACK
SIGMOD 2008

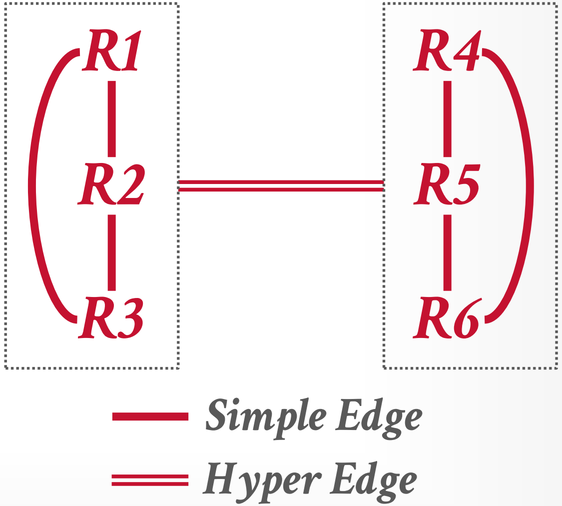
DPHYP: HYPERGRAPHS

A hypergraph is a pair $H=(V,E)$ such that:

- → V is a non-empty set of nodes.
- → E is a set of hyperedges, where a hyperedge is an unordered pair (u,v) of non-empty subsets of V ($u \subset V$, $v \subset V$) with the additional condition that $u \cap v = \emptyset$.

Allows search algorithm to consider node groupings instead of each individual node.

```
SELECT *
  FROM R1, R2, R3, R4, R5, R6
 WHERE R1.a + R2.b + R3.c
       =
       R4.d + R5.e + R6.f;
```



DYNAMIC PROGRAMMING STRIKES BACK
SIGMOD 2008

DPHYP: ALGORITHM

Traverse the graph in a fixed order and recursively produce larger connected subgraphs.

- → Incrementally expand connected subgraphs.
- → Identify reachable nodes from a subgraph, excluding certain nodes based on constraints.
- → Treat hypernodes as single instances when choosing subsets.

DPHyp handles complex join predicates and non-inner joins.

MEDIUM QUERIES

For a query with more than 100 relations, the search strategy depends on its graph structure.

The goal is to convert every query into a chain query to simplify the problem.

→ ***Search Space Linearization***

→ Only need to consider associativity and <u>not</u> commutativity of relations when enumerating join orderings.

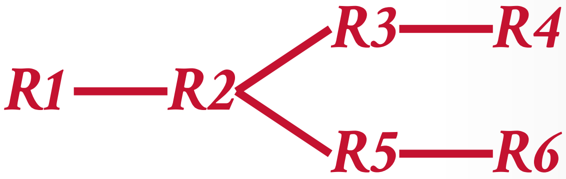
Source: Thomas Neumann

SEARCH SPACE LINEARIZATION

Assume the order of relations in the optimal plan is known.

Use a polynomial DP algorithm to generate optimal plan from this linearization.

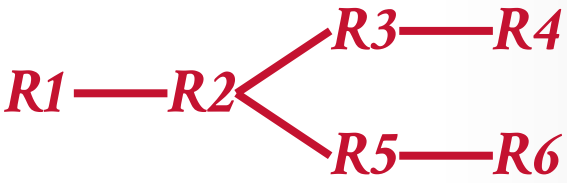
Optimally combine optimal solutions for sub-chains of increasing size.



Source: Thomas Neumann

SEARCH SPACE LINEARIZATION

Assume the order of relations in the optimal plan is known.



Use a polynomial DP algorithm to generate optimal plan from this linearization.

Optimally combine optimal solutions for sub-chains of increasing size.

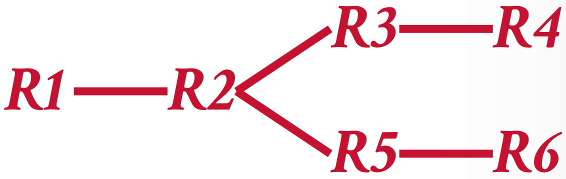
Source: Thomas Neumann

SEARCH SPACE LINEARIZATION

Assume the order of relations in the optimal plan is known.

Use a polynomial DP algorithm to generate optimal plan from this linearization.

Optimally combine optimal solutions for sub-chains of increasing size.



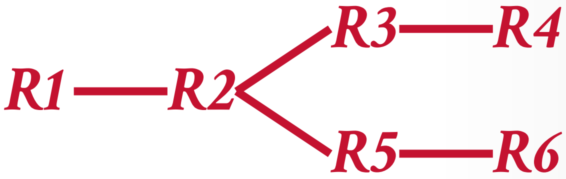
Source: Thomas Neumann

SEARCH SPACE LINEARIZATION

Assume the order of relations in the optimal plan is known.

Use a polynomial DP algorithm to generate optimal plan from this linearization.

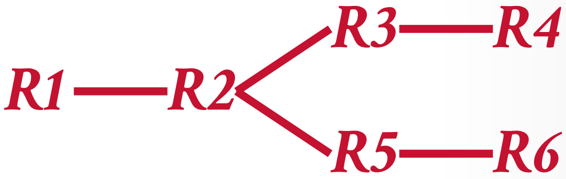
Optimally combine optimal solutions for sub-chains of increasing size.



Source: Thomas Neumann

SEARCH SPACE LINEARIZATION

Assume the order of relations in the optimal plan is known.



Use a polynomial DP algorithm to generate optimal plan from this linearization.

Optimally combine optimal solutions for sub-chains of increasing size.

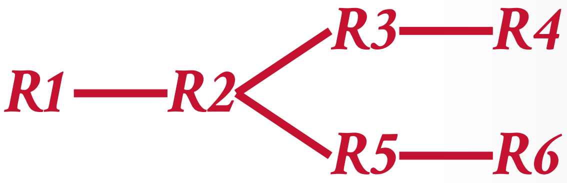
Source: Thomas Neumann

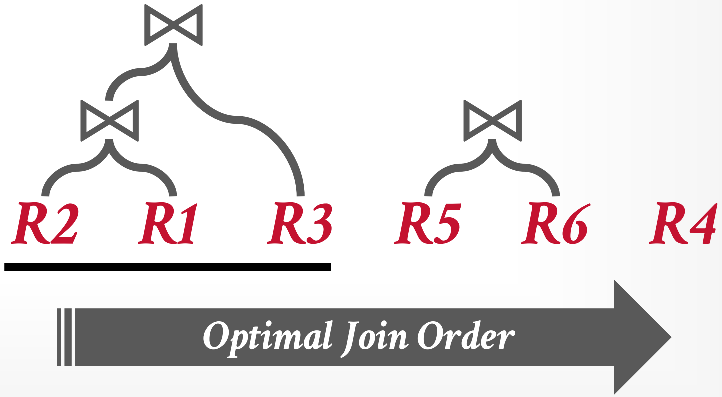
SEARCH SPACE LINEARIZATION

Assume the order of relations in the optimal plan is known.

Use a polynomial DP algorithm to generate optimal plan from this linearization.

Optimally combine optimal solutions for sub-chains of increasing size.





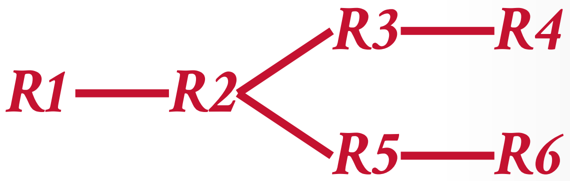
Source: Thomas Neumann

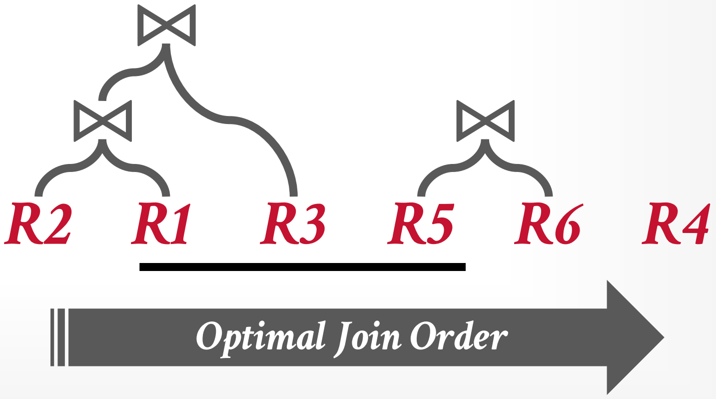
SEARCH SPACE LINEARIZATION

Assume the order of relations in the optimal plan is known.

Use a polynomial DP algorithm to generate optimal plan from this linearization.

Optimally combine optimal solutions for sub-chains of increasing size.

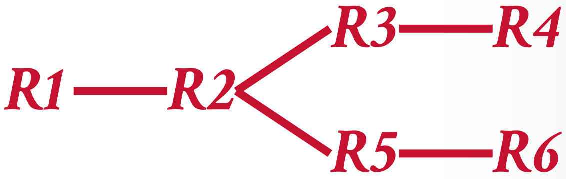




Source: Thomas Neumann

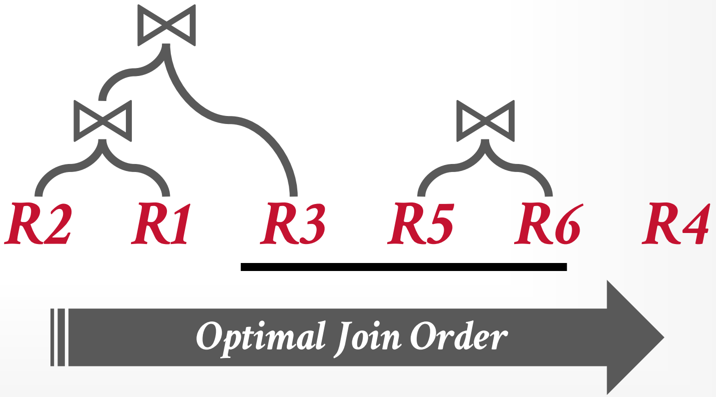
SEARCH SPACE LINEARIZATION

Assume the order of relations in the optimal plan is known.



Use a polynomial DP algorithm to generate optimal plan from this linearization.

Optimally combine optimal solutions for sub-chains of increasing size.



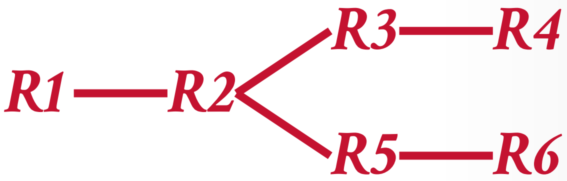
Source: Thomas Neumann

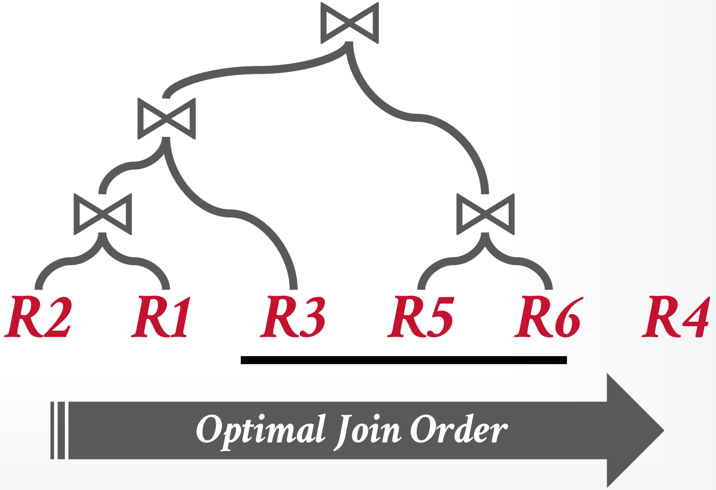
SEARCH SPACE LINEARIZATION

Assume the order of relations in the optimal plan is known.

Use a polynomial DP algorithm to generate optimal plan from this linearization.

Optimally combine optimal solutions for sub-chains of increasing size.





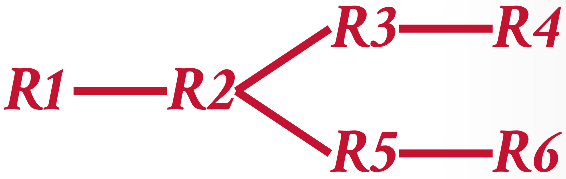
Source: Thomas Neumann

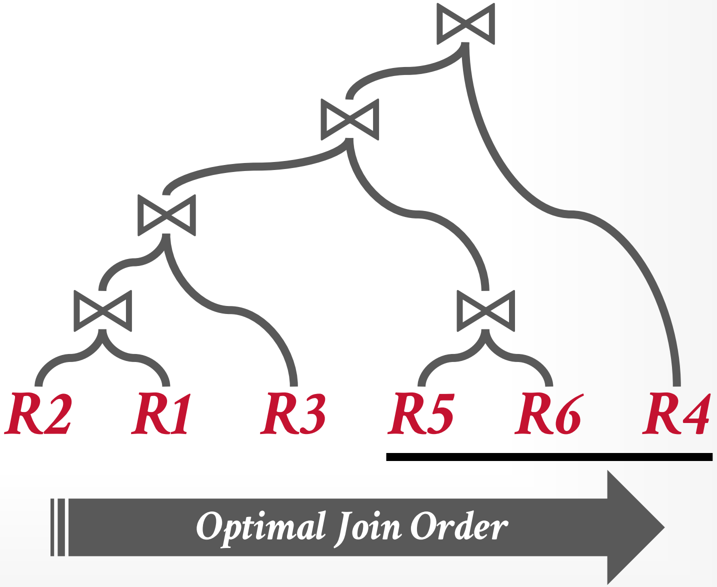
SEARCH SPACE LINEARIZATION

Assume the order of relations in the optimal plan is known.

Use a polynomial DP algorithm to generate optimal plan from this linearization.

Optimally combine optimal solutions for sub-chains of increasing size.





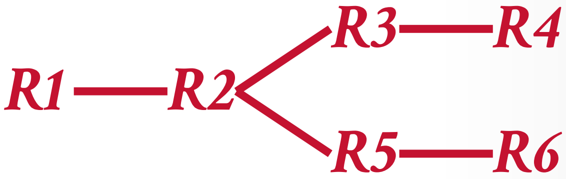
Source: Thomas Neumann

SEARCH SPACE LINEARIZATION

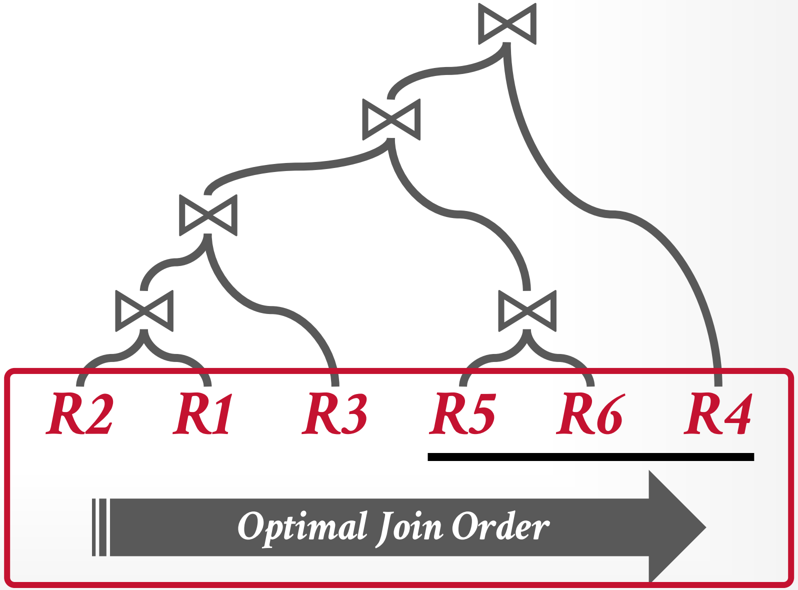
Assume the order of relations in the optimal plan is known.

Use a polynomial DP algorithm to generate optimal plan from this linearization.

Optimally combine optimal solutions for sub-chains of increasing size.



???



Source: Thomas Neumann

OPTIMAL ORDER

IKKBZ algorithm from 1984/1986 to generate an optimal left-deep plan in $O(n^2)$.

Algorithm Overview:

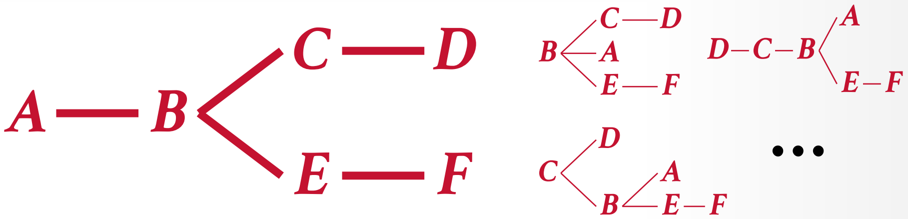
- → Transform precedence graph into a linear order.
- → If query graph has cycles, generate min-spanning tree.
- → Assign a rank to nodes (cost/benefit ratio).
- → Successively merge child chains increasing in ranks.
- → Resolve contradictory sequences in child chains by merging them into a single node.

ON THE OPTIMAL NESTING ORDER FOR COMPUTING *N*-RELATIONAL JOINS
TODS 1984

OPTIMIZATION OF NONRECURSIVE QUERIES
VLDB 1986

IKKBZ ALGORITHM

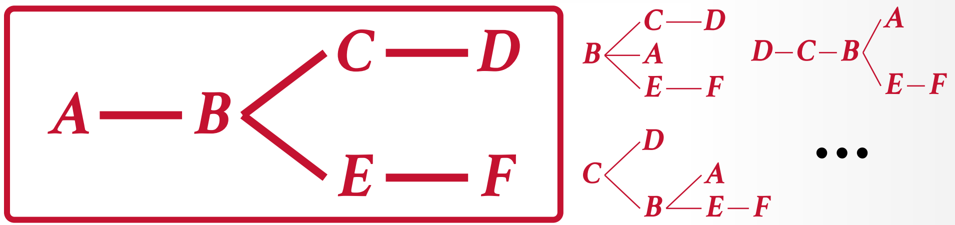
Build a precedence graph for each individual relation.

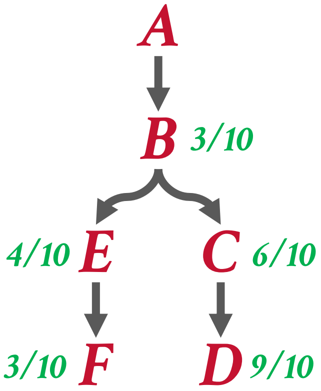


Source: Thomas Neumann

IKKBZ ALGORITHM

Build a precedence graph for each individual relation.





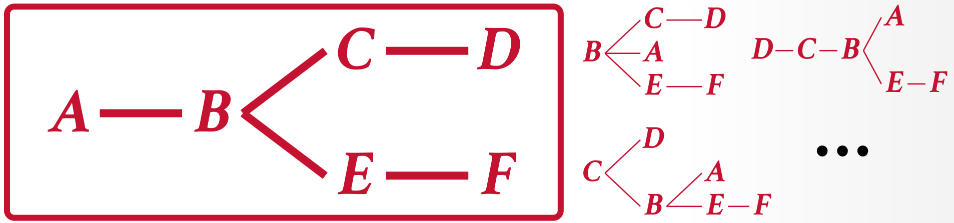
Source: Thomas Neumann

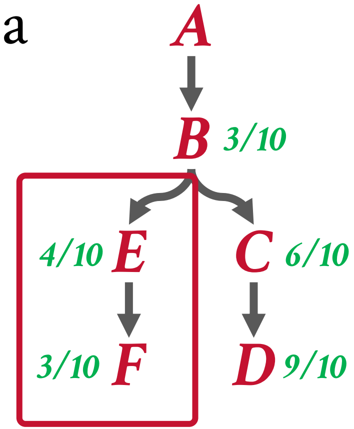
IKKBZ ALGORITHM

Build a precedence graph for each individual relation.

Resolve contradictory sequences in child chains by merging into a single node.

→ Ex: ***rank(E)* > *rank(F)***, but ***E*** precedes ***F***





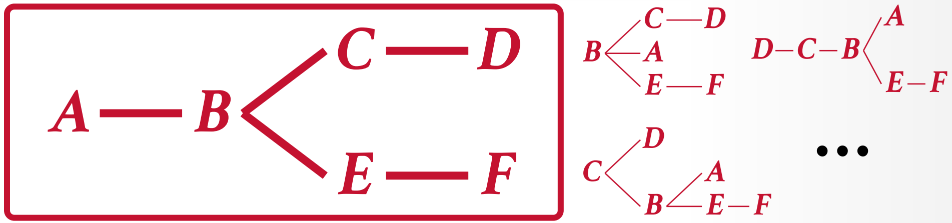
Source: Thomas Neumann

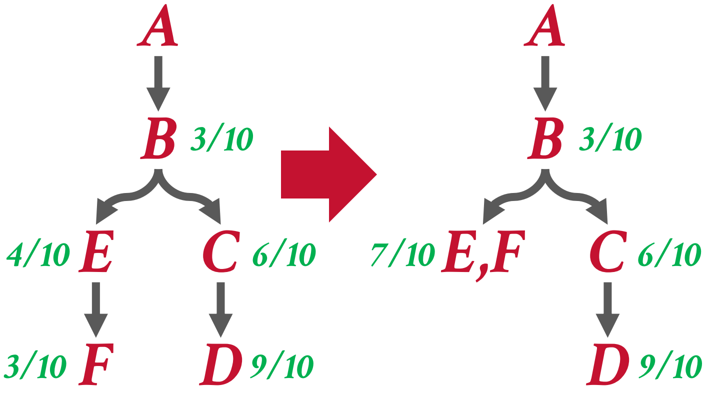
IKKBZ ALGORITHM

Build a precedence graph for each individual relation.

Resolve contradictory sequences in child chains by merging into a single node.

→ Ex: $rank(E) > rank(F)$, but E precedes F





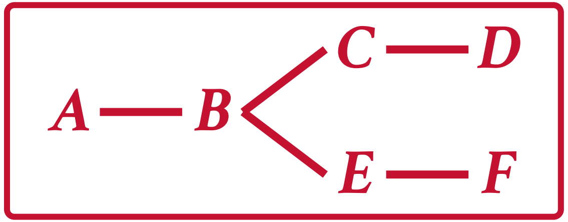
Source: Thomas Neumann

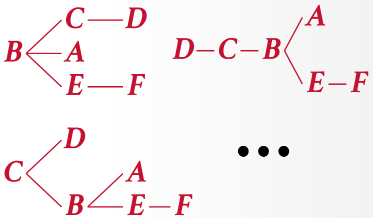
IKKBZ ALGORITHM

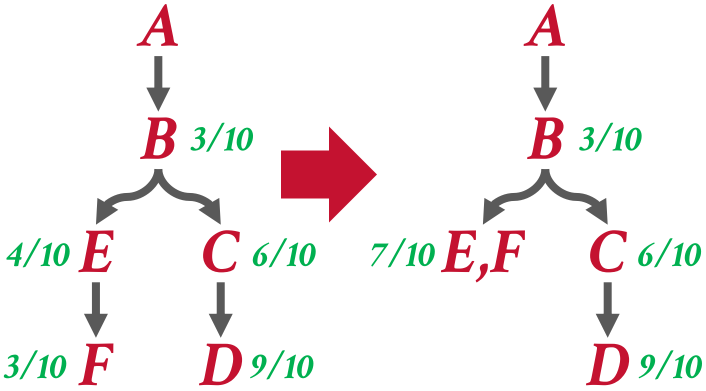
Build a precedence graph for each individual relation.

Resolve contradictory sequences in child chains by merging into a single node.

→ Ex: ***rank(E) > rank(F)***, but ***E*** precedes ***F***







M

→ Ex: ***rank(C) < rank(E,F) < rank(D)***

→ ***e nodes rank***

Source: Thomas Neumann

IKKBZ ALGORITHM

Build a precedence graph for each individual relation.

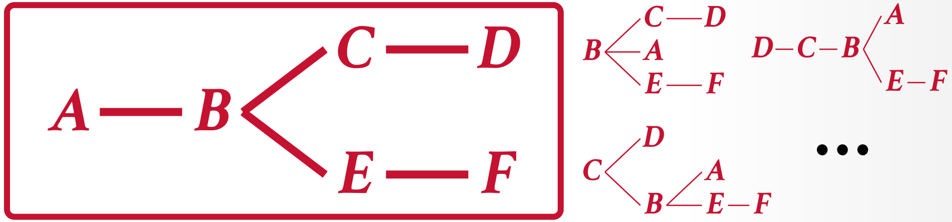
Resolve contradictory sequences in child chains by merging into a single node.

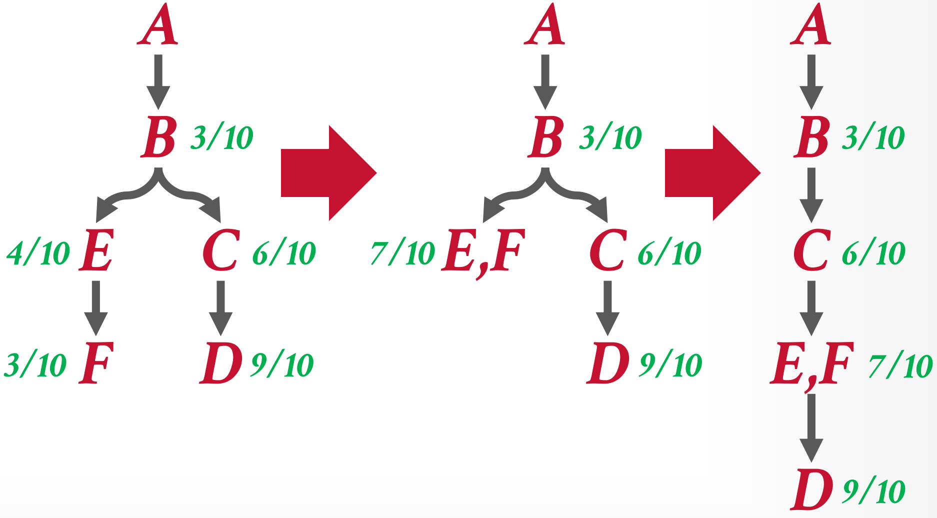
→ Ex: ***rank(E) > rank(F)***, but ***E*** precedes ***F***

M

→ Ex: ***rank(C) < rank(E,F) < rank(D)***

→ ***e nodes rank***





Source: Thomas Neumann

MEDIUM QUERIES

Procedure:

→ Linearize query graph using **IKKBZ**.
→ Build best bushy plan for linearization.

Properties:

→ Algorithm runs in $O(n^3)$
→ Result is at least as good as the optimal left-deep plan.
→ With proper linearization, discovers globally optimal bushy plan.

Source: Thomas Neumann

LARGE QUERIES

Use an iterative dynamic programming approach to handle the most complex queries.

→ First greedily build an initial query plan.

→ Then incrementally refine the plan by optimizing the most expensive sub-trees of size $\boxed{k}$ using DP.

Use the same linearization trick to improve k=7 to k=100!

Greedy Algorithms:

→ Minimum Selectivity *(Min-sel)*

→ Greedy Operator Ordering *(GOO)*

Source: Thomas Neumann

LARGE QUERIES

Use an iterative dynamic programming approach to handle the most complex queries.
→ First greedily build an initial query plan.
→ Then incrementally refine the plan by optimizing the most expensive sub-trees of size ***k*** using DP.

Use the same linearization trick to improve k=7 to k=100!

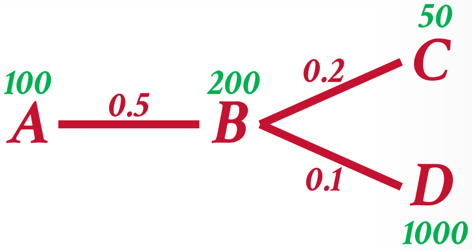
Greedy Algorithms:
→ Minimum Selectivity *(Min-sel)*
→ Greedy Operator Ordering *(GOO)*

Source: Thomas Neumann

GREEDY OPERATOR ORDERING (GOO)

Find pair of connected nodes ***i***, ***j*** with min value:

→ ***size(i) × size(j) × selectivity(i,j)***

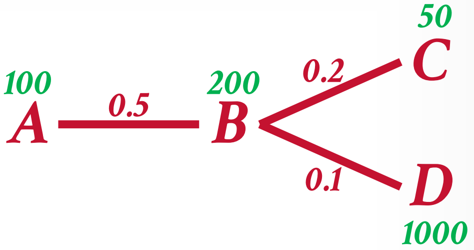


A NEW HEURISTIC FOR OPTIMIZING LARGE QUERIES
DEXA 1998

GREEDY OPERATOR ORDERING (GOO)

Find pair of connected nodes ***i***, ***j*** with min value:

→ ***size(i) × size(j) × selectivity(i,j)***



A,B = 100 × 200 × 0.5 = 10000

B,C = 200 × 50 × 0.2 = 2000

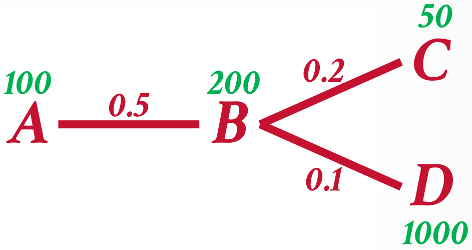
B,D = 200 × 1000 × 0.1 = 20000

A NEW HEURISTIC FOR OPTIMIZING LARGE QUERIES
DEXA 1998

GREEDY OPERATOR ORDERING (GOO)

Find pair of connected nodes ***i***, ***j*** with min value:

→ ***size(i) × size(j) × selectivity(i,j)***



$A,B = 100 \times 200 \times 0.5 = 10000$

$\boxed{B,C = 200 \times 50 \times 0.2 = 2000}$

$B,D = 200 \times 1000 \times 0.1 = 20000$

A NEW HEURISTIC FOR OPTIMIZING LARGE QUERIES
DEXA 1998

GREEDY OPERATOR ORDERING (GOO)

Find pair of connected nodes ***i***, ***j*** with min value:

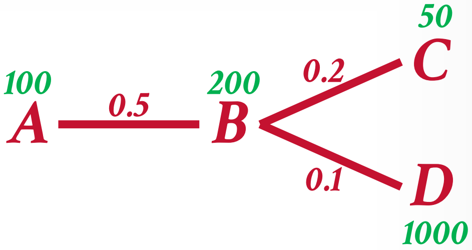
→ ***size(i) × size(j) × selectivity(i,j)***

Merge nodes ***i*** and ***j*** into new node and update graph:

→ S

→ Recompute selectivities of edges to other nodes.

→ vity(i,j)



$A,B = 100 \times 200 \times 0.5 = 10000$

$\boxed{B,C = 200 \times 50 \times 0.2 = 2000}$

$B,D = 200 \times 1000 \times 0.1 = 20000$

A NEW HEURISTIC FOR OPTIMIZING LARGE QUERIES
DEXA 1998

GREEDY OPERATOR ORDERING (GOO)

Find pair of connected nodes ***i***, ***j*** with min value:

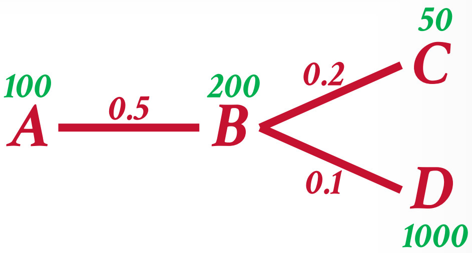
→ ***size(i) × size(j) × selectivity(i,j)***

Merge nodes ***i*** and ***j*** into new node and update graph:

→ S

→ Recompute selectivities of edges to other nodes.

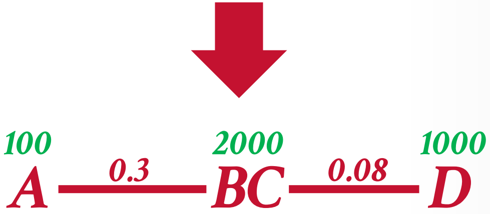
→ vity(i,j)



$A,B = 100 \times 200 \times 0.5 = 10000$

$\boxed{B,C = 200 \times 50 \times 0.2 = 2000}$

$B,D = 200 \times 1000 \times 0.1 = 20000$



A NEW HEURISTIC FOR OPTIMIZING LARGE QUERIES
DEXA 1998

GREEDY OPERATOR ORDERING (GOO)

Find pair of connected nodes ***i***, ***j*** with min value:

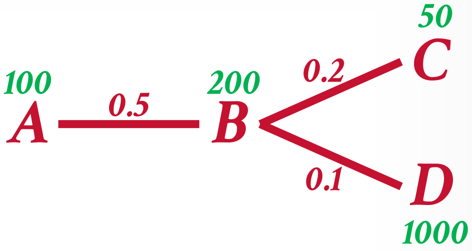
→ ***size(i) × size(j) × selectivity(i,j)***

Merge nodes ***i*** and ***j*** into new node and update graph:

→ S

→ Recompute selectivities of edges to other nodes.

Repeat until only one node remains.



$A,B = 100 \times 200 \times 0.5 = 10000$

$\boxed{B,C = 200 \times 50 \times 0.2 = 2000}$

$B,D = 200 \times 1000 \times 0.1 = 20000$

A NEW HEURISTIC FOR OPTIMIZING LARGE QUERIES
DEXA 1998

GREEDY OPERATOR ORDERING (GOO)

Find pair of connected nodes ***i***, ***j*** with min value:

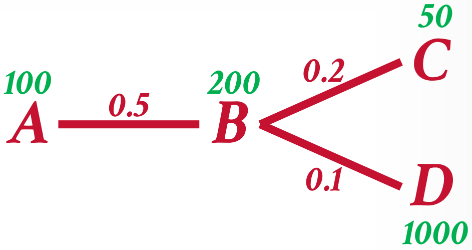
→ ***size(i) × size(j) × selectivity(i,j)***

Merge nodes ***i*** and ***j*** into new node and update graph:

→ S

→ Recompute selectivities of edges to other nodes.

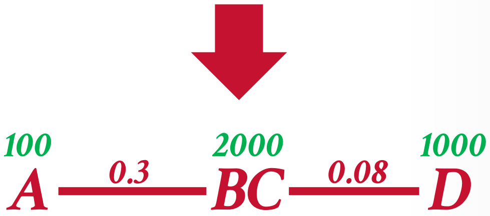
Repeat until only one node remains.



$A,B = 100 \times 200 \times 0.5 = 10000$

$\boxed{B,C = 200 \times 50 \times 0.2 = 2000}$

$B,D = 200 \times 1000 \times 0.1 = 20000$



$\boxed{A,BC = 100 \times 2000 \times 0.3 = 60000}$

$BC,D = 2000 \times 1000 \times 0.08 = 160000$

A NEW HEURISTIC FOR OPTIMIZING LARGE QUERIES
DEXA 1998

GREEDY OPERATOR ORDERING (GOO)

Find pair of connected nodes ***i***, ***j*** with min value:

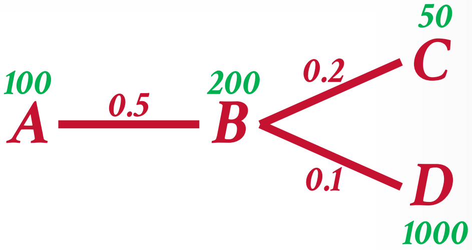
→ ***size(i) × size(j) × selectivity(i,j)***

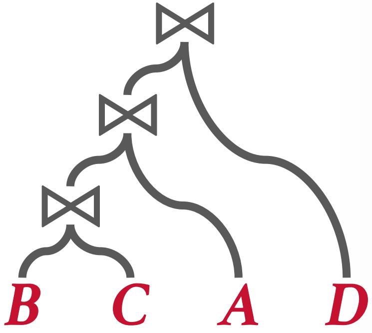
Merge nodes ***i*** and ***j*** into new node and update graph:

→ S

→ Recompute selectivities of edges to other nodes.

Repeat until only one node remains.





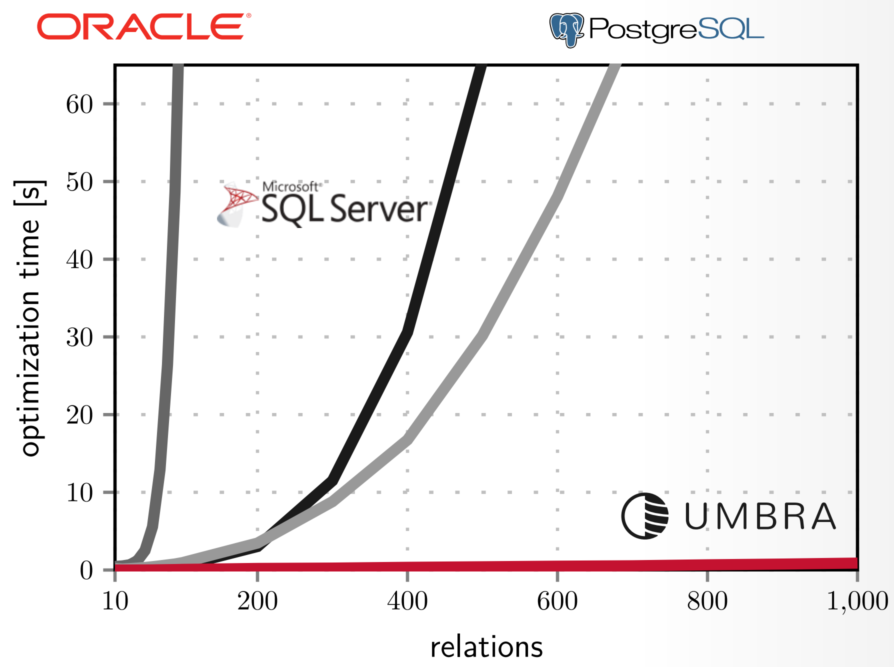
Works with cyclic and acyclic query graphs.
Generates bushy (not just deep-left) join trees.

A NEW HEURISTIC FOR OPTIMIZING LARGE QUERIES
DEXA 1998

EXPERIMENTAL RESULTS

Comparison of different DBMSs on random queries with an increasing number of relations.

Neither the algorithms or data structures used in other optimizer implementations can handle large queries.



Source: Thomas Neumann

OBSERVATION

All the methods we've discuss today assume queries only contain inner joins and no cross products.

We will further examine how to consider outer joins and cross products next class.

RANDOMIZED ALGORITHMS

Alternative for handling large queries by randomly exploring solution space of (valid) plans for a query.

- → Keep searching until a cost threshold is reached or the optimizer runs for a length of time.
- → No guarantees about optimality of plans.

Only one DBMS does this and the quality of their plans are known to be bad…

QUICKPICK

Incrementally build random join trees and then pick the one that has the lowest cost.

- → Randomly select and remove an edge in the query graph.
- → Add a join or predicate to the new plan.
- → If new query plan has lower cost than best plan seen, keep going.
- → Otherwise, discard plan, reset query graph, and start over.

Bias the sampling function when choosing an edge towards edges based with lower selectivities.

JOIN ORDER SELECTION — GOOD ENOUGH IS EASY
ADVANCES IN DATABASES 2000

SIMULATED ANNEALING

Start with a query plan that is generated using the heuristic-only approach.

Compute random permutations of operators (e.g., swap the join order of two tables):

- → Always accept a change that reduces cost.
- → Only accept a change that increases cost with some probability.
- → Reject any change that violates correctness (e.g., sort ordering).

QUERY OPTIMIZATION BY SIMULATED ANNEALING
SIGMOD 1987

POSTGRES GENETIC OPTIMIZER

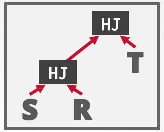
More complicated queries use a **genetic algorithm** that selects join orderings (GEQO).

At the beginning of each round, generate different variants of the query plan.

Select the plans that have the lowest cost and permute them with other plans. Repeat.
→ The mutator function only generates valid plans.

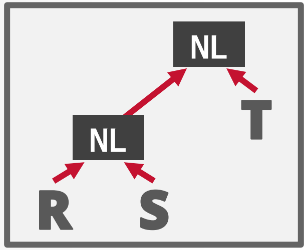
Source: Postgres Documentation

POSTGRES GENETIC OPTIMIZER

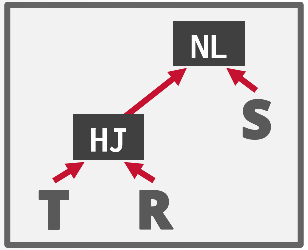


Best:100

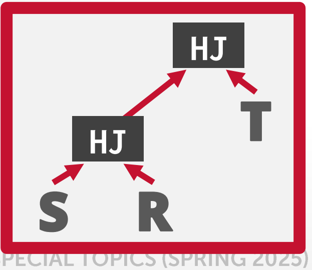
1st Generation



Cost: 300

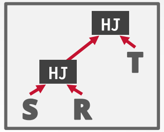


Cost: 200



Cost: 100

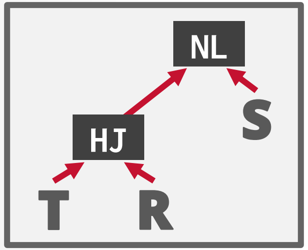
POSTGRES GENETIC OPTIMIZER



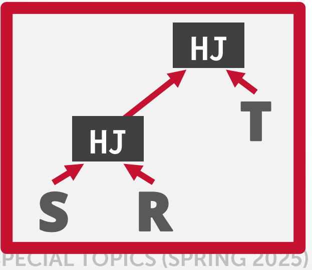
Best:100

1st Generation

Cost: 300

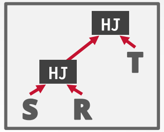


Cost: 200



Cost: 100

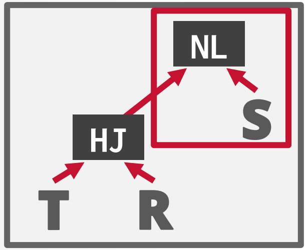
POSTGRES GENETIC OPTIMIZER



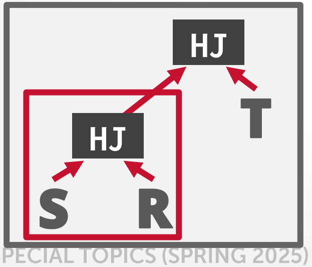
Best:100

1st Generation

Cost: 300

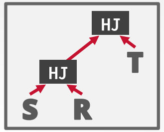


Cost: 200



Cost: 100

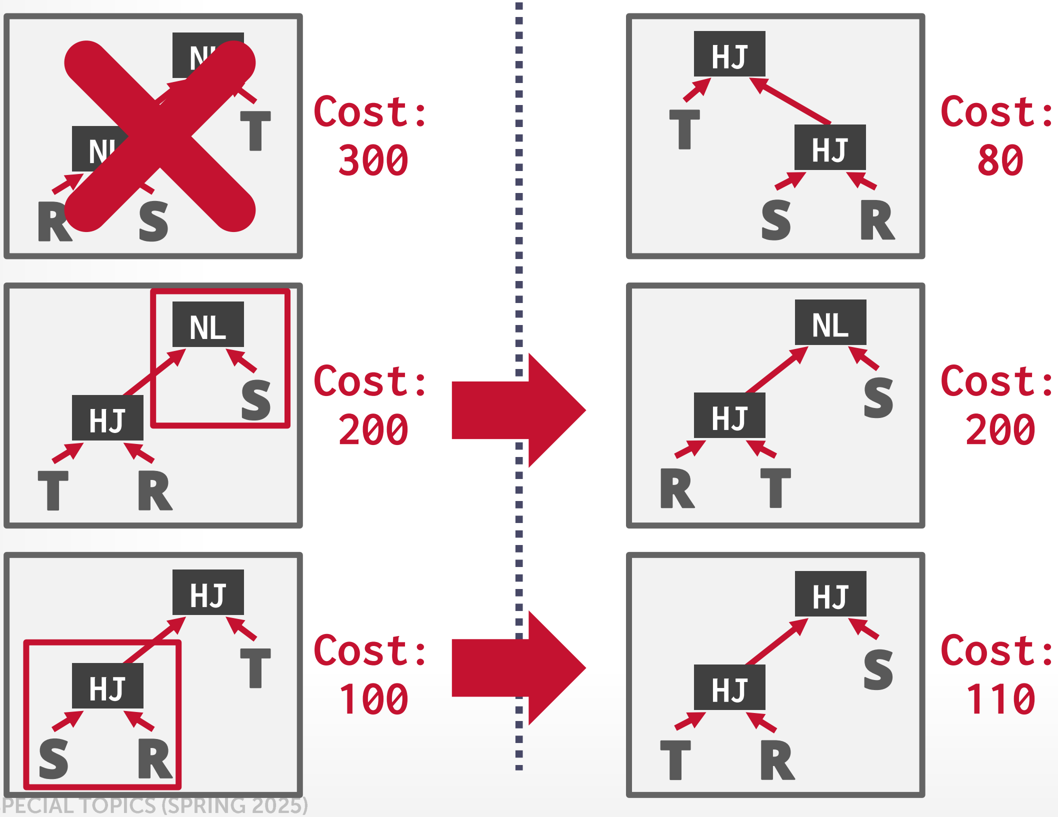
POSTGRES GENETIC OPTIMIZER



Best:100

1st Generation

2nd Generation



Cost: 300

Cost: 200

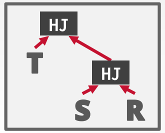
Cost: 100

Cost: 80

Cost: 200

Cost: 110

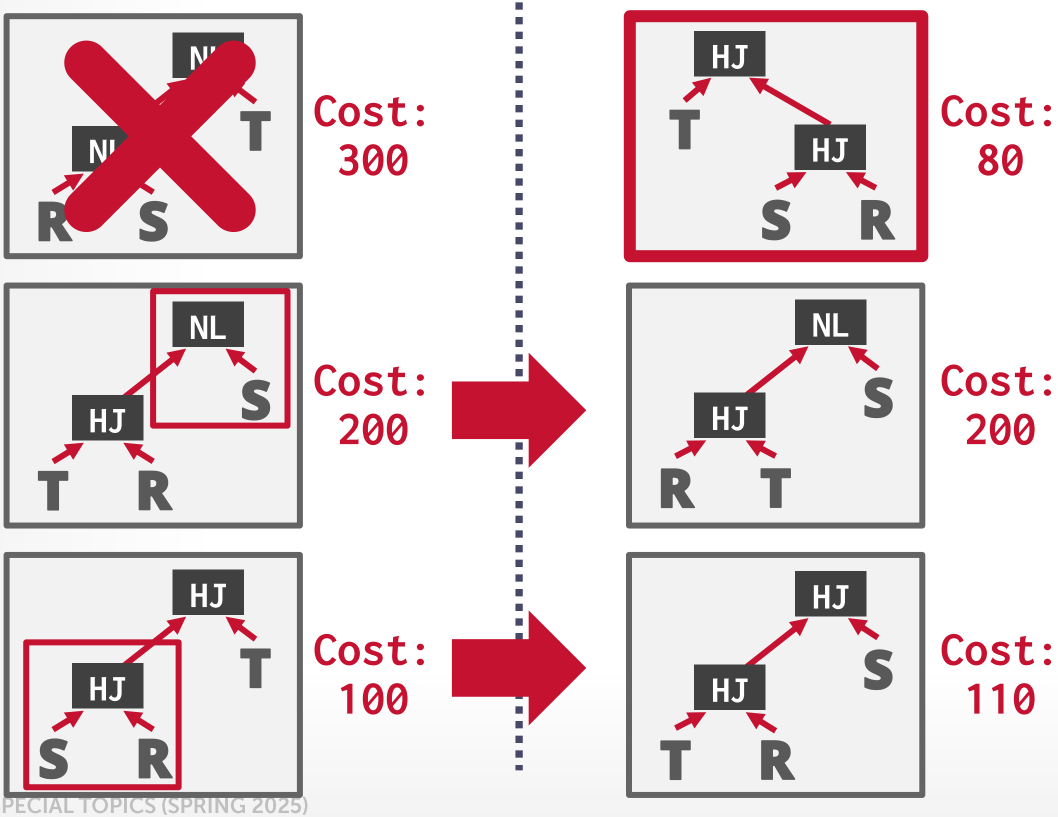
POSTGRES GENETIC OPTIMIZER



Best:80

1st Generation

2nd Generation



Cost: 300

Cost: 200

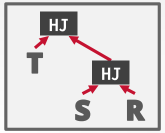
Cost: 100

Cost: 80

Cost: 200

Cost: 110

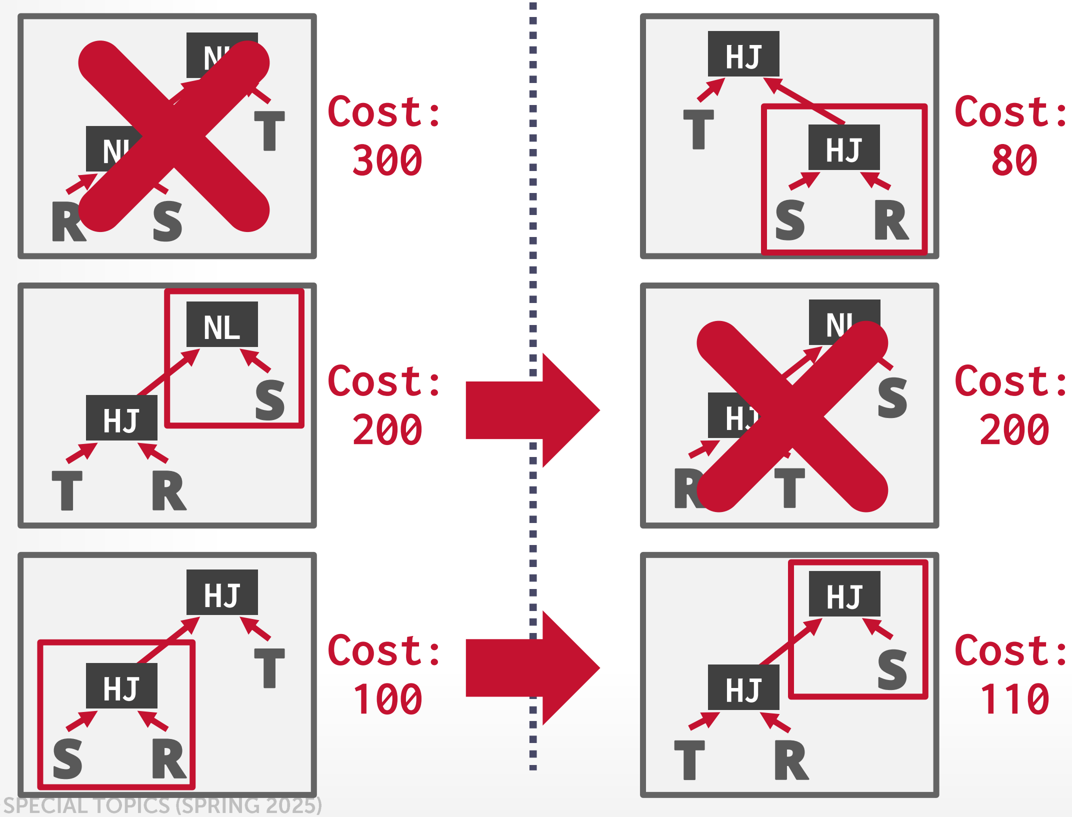
POSTGRES GENETIC OPTIMIZER



Best:80

1st Generation

2nd Generation



Cost: 300

Cost: 200

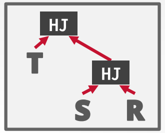
Cost: 100

Cost: 80

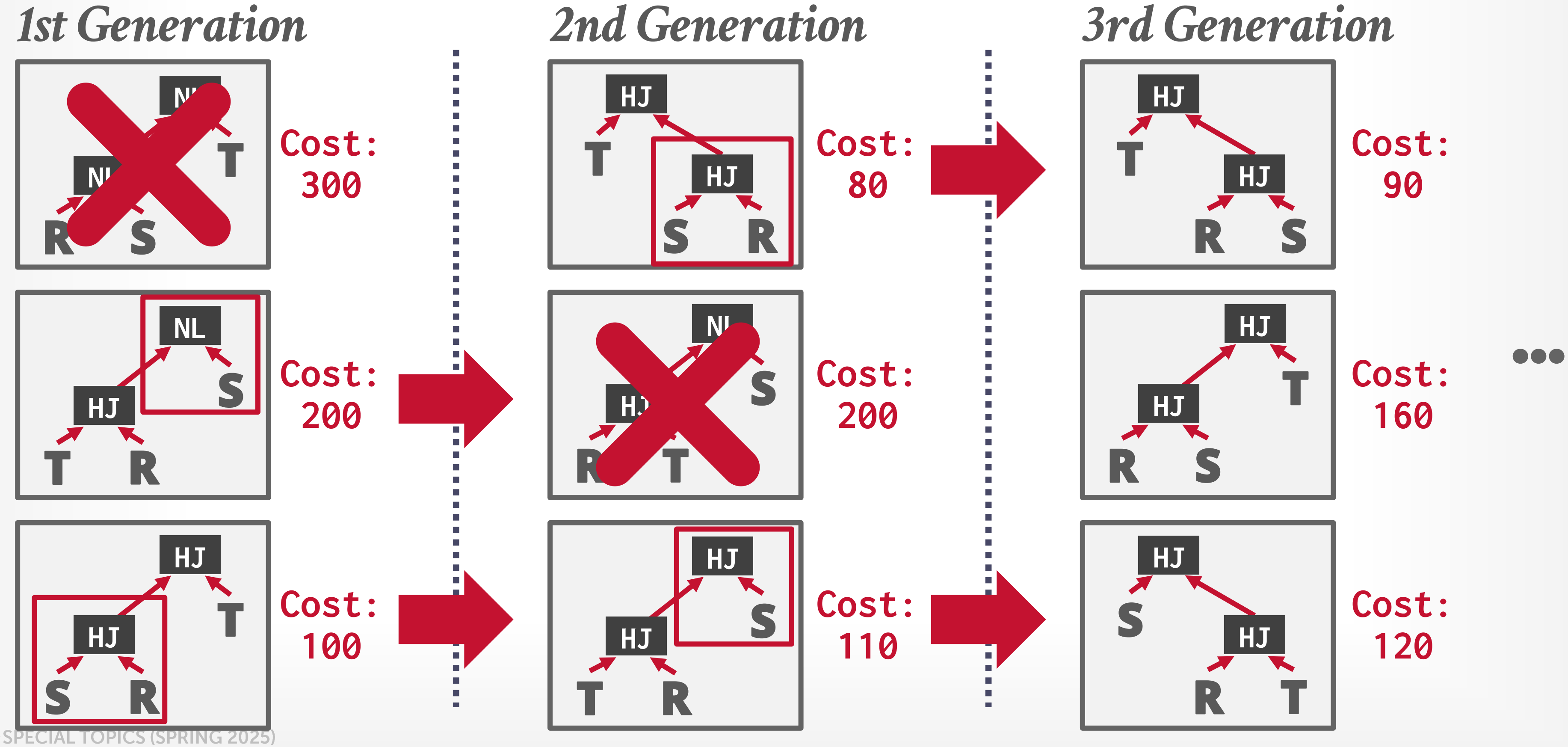
Cost: 200

Cost: 110

POSTGRES GENETIC OPTIMIZER



Best:80



1st Generation

Cost: 300

Cost: 200

Cost: 100

2nd Generation

Cost: 80

Cost: 200

Cost: 110

3rd Generation

Cost: 90

Cost: 160

Cost: 120

RANDOMIZED ALGORITHMS

Advantages:

- → Jumping around the search space randomly allows the optimizer to get out of local minimums.
- → Low memory overhead (if no history is kept).

Disadvantages:

- → Difficult to determine why the optimizer may have chosen a plan.
- → Must do extra work to ensure that query plans are deterministic.

RANDOMIZED ALGORITHMS

Advantages:

- → Jumping around the search space randomly allows the optimizer to get out of local minimums.
- → Low memory overhead (if no history is kept).

Disadvantages:

- → Difficult to determine why the optimizer may have chosen a plan.
- → Must do extra work to ensure that query plans are deterministic.

PARTING THOUGHTS

Using different strategies based on the complexity of a query is a good idea.

Use an "inferior" algorithm that is fast to get a quick answer, then spend remaining time to refine it with more robust methods.

NEXT CLASS

Top-down join enumeration
→ Assigned reading will not consider adaptivity.