Join Ordering Problem

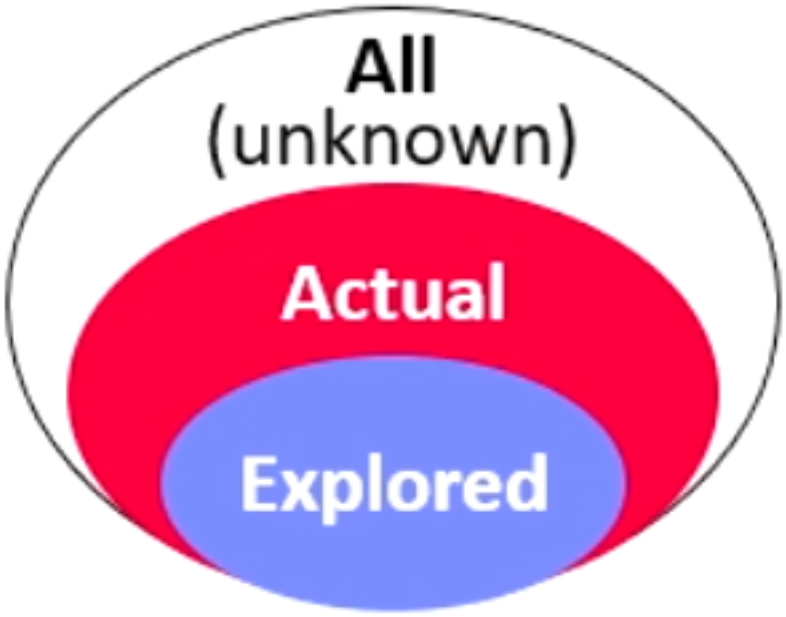
- given a join query graph, find the optimal join ordering
- usually NP-hard
  - polynomical algorithms exist for special cases
- search space sizes

	Chain (no CP)			Star (no CP)		Clique / CP (cross product)		
	left-deep	zig-zag	bushy	left-deep	zig-zag/ bushy	left-deep	zig-zag	bushy
n	2^{n-1}	2^{2n-3}	$2^{n-1}C(n-1)$	2(n-1)!	$2^{n-1}(n-1)!$	n!	$2^{n-2}n!$	n! C(n-1)
5	16	128	224	48	384	120	960	1,680
10	512	~131K	~2.4M	~726K	~186M	~3.6M	~929M	~17.6G

  - C(n) ... Catalan Numbers

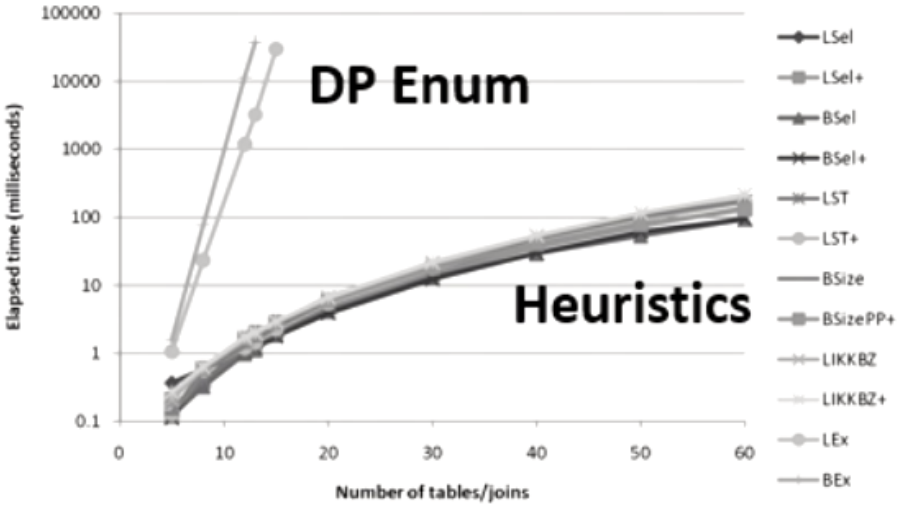
Join Order Seach Strategies

- tradeoff between optimal plan and compile time



-
- naive full enumeration
  - infeasible for large queries
- exact dynamic programming
  - guarantees optimal plan
  - often too expensive (beyond 20 relations)
  - bottom-up/top-down approach
- greedy/heuristic algorithms
  - tests out some
  - ignore worst

  - further optimization on best
    - e.g. random mutations (as with genetics)
- approximate algorithms



-

Greedy Join Ordering

- does not always return optimal join ordering
- algorithm:
  - calculate cost of each two table join combination
  - calculate costs of previous join with next table
  - repeat until every table is used
- example
  - Part ⋈ Lineorder ⋈ Supplier ⋈ σ(Customer) ⋈ σ(Date), **left-deep plans**

#	Plan	Costs
1	Lineorder ⋈ Part	30M
	Lineorder ⋈ Supplier	20M
	Lineorder ⋈ σ(Customer)	90K
	Lineorder ⋈ σ(Date)	**40K**
	~~Part ⋈ Customer~~	N/A
	...	...
2	(Lineorder ⋈ σ(Date)) ⋈ Part	150K
	(Lineorder ⋈ σ(Date)) ⋈ Supplier	100K
	(Lineorder ⋈ σ(Date)) ⋈ σ(Customer)	**75K**
3	((Lineorder ⋈ σ(Date)) ⋈ σ(Customer)) ⋈ Part	120M
	((Lineorder ⋈ σ(Date)) ⋈ σ(Customer)) ⋈ Supplier	**105M**
4	**(((Lineorder ⋈ σ(Date)) ⋈ σ(Customer)) ⋈ Supplier) ⋈ Part**	**135M**

Note: Simple $O(n^2)$ algorithm for left-deep trees; $O(n^3)$ algorithms for bushy trees existing (e.g., GOO)

-

Dynamic Programming Join Ordering

- exact enumeration via dynamic programming
  - tries to find optimal substructures first
  - overlapping subproblems allow for memoization

        * reuse already retrieved data
- e.g. DPSize
  - bottom-up
  - split into independent subproblems
  - solve subproblems
  - combine solutions
  - ▪ **Example**

Q1	Plan
{C}	Tbl, IX
{D}	Tbl, IX
{L}	...
{P}	...
{S}	...

Q1+Q1

Q2	Plan
{C,L}	L⋈C, ~~C⋈L~~
{D,L}	L⋈D, ~~D⋈L~~
{L,P}	~~L⋈P~~, P⋈L
{L,S}	~~L⋈S~~, S⋈L
~~{C,D}~~	~~N/A~~
...	...

Q1+Q2, Q2+Q1

Q3	Plan
{C,D,L}	(L⋈C)⋈D, ~~D⋈(L⋈C)~~, ~~(L⋈D)⋈C~~, ~~C⋈(L⋈D)~~
{C,L,P}	~~(L⋈C)⋈P~~, P⋈(L⋈C), ~~(P⋈L)⋈C~~, ~~C⋈(P⋈L)~~
{C,L,S}	...
{D,L,P}	...
{D,L,S}	...
{L,P,S}	...

Q1+Q3, Q2+Q2, Q3+Q1

Q4	Plan
{C,D,L,P}	~~((L⋈C)⋈D)⋈P~~, P⋈((L⋈C)⋈D)
{C,D,L,S}	...
{C,L,P,S}	...
{D,L,P,S}	...

Q1+Q4, Q2+Q3, Q3+Q2, Q4+Q1

Q5	Plan
{C,D,L,P,S}	...

  -

[[Plan Optimization]]