Classification of Join Ordering Problems

We distinguish four different dimensions:

1. query graph class: *chain*, *cycle*, *star*, and *clique*
2. join tree structure: *left-deep*, *zig-zag*, or *bushy* trees
3. join construction: *with* or *without* cross products
4. cost function: *with* or *without* ASI property

In total, 48 different join ordering problems.

Reminder: Catalan Numbers

The number of binary trees with n leave nodes is given by $\mathcal{C}(n-1)$, where $\mathcal{C}(n)$ is defined as

$$\mathcal{C}(n) = \begin{cases} 1 & \text{if } n = 0 \\ \sum_{k=0}^{n-1} \mathcal{C}(k)\mathcal{C}(n-k-1) & \text{if } n > 0 \end{cases}$$

It can be written in a closed form as

$$\mathcal{C}(n) = \frac{1}{n+1}\binom{2n}{n}$$

The Catalan Numbers grown in the order of $\Theta(4^n/n^{\frac{3}{2}})$

Number Of Join Trees with Cross Products

left deep	$n!$
right deep	$n!$
zig-zag	$n!2^{n-2}$
bushy	$n!\mathcal{C}(n-1)$
	$= \frac{(2n-2)!}{(n-1)!}$

- rational: number of leaf combinations ($n!$) $\times$ number of unlabeled trees (varies)
- grows exponentially
- increases even more with a flexible tree structure

Chain Queries, no Cross Products

Let us denote the number of left-deep join trees for a chain query $R_1 - \ldots - R_n$ as $f(n)$

- obviously $f(0) = 1, f(1) = 1$
- for $n > 1$, consider adding R_n to all join trees for $R_1 - \ldots - R_{n-1}$
- R_n can be added at any position following R_{n-1}
- lets denote the position of R_{n-1} from the bottom with k ($[1, n-1]$)
- there are $n - k$ join trees for adding R_n after R_{n-1}
- one additional tree if $k = 1$, R_n can also be added before R_{n-1}
- for R_{n-1} to be at k, $R_{n-k} - \ldots R_{n-2}$ must be below it. $f(k-1)$ trees

for $n > 1$:

$$f(n) = 1 + \sum_{k=1}^{n-1} f(k-1) * (n-k)$$

Chain Queries, no Cross Products (2)

The number of left-deep join trees for chain queries of size n is

$$f(n) = \begin{cases} 1 & \text{if } n < 2 \\ 1 + \sum_{k=1}^{n-1} f(k-1) * (n-k) & \text{if } n \geq 2 \end{cases}$$

solving the recurrence gives the closed form

$$f(n) = 2^{n-1}$$

- generalization to zig-zag as before

Chain Queries, no Cross Products (3)

The generalization to bushy trees is not as obvious

- each subtree must contain a subchain to avoid cross products
- thus do not add single relations but subchains
- whole chain must be $R_1 - \ldots - R_n$, cut anywhere
- consider commutativity (two possibilities)

This leads to the formula

$$f(n) = \begin{cases} 1 & \text{if } n < 2 \\ \sum_{k=1}^{n-1} 2f(k)f(n-k) & \text{if } n \geq 2 \end{cases}$$

solving the recurrence gives the closed form

$$f(n) = 2^{n-1}\mathcal{C}(n-1)$$

Star Queries, no Cross Products

Consider a star query with R_1 at the center and $R_2, \ldots, R_n$ as satellites.

- the first join must involve R_1
- afterwards all other relations can be added arbitrarily

This leads to the following formulas:

- left-deep: $2 * (n-1)!$
- zig-zag: $2 * (n-1)! * 2^{n-2} = (n-1)! * 2^{n-1}$
- bushy: no bushy trees possible (R_1 required), same as zig-zag

Clique Queries, no Cross Products

- in a clique query, every relation is connected to each other
- thus no join tree contains cross products
- all join trees are valid join trees, the number is the same as with cross products

Sample Numbers, without Cross Products

	Chain Queries			Star Queries	
	Left-Deep	Zig-Zag	Bushy	Left-Deep	Zig-Zag/Bushy
n	2^{n-1}	2^{2n-3}	$2^{n-1}\mathcal{C}(n-1)$	$2(n-1)!$	$2^{n-1}(n-1)!$
1	1	1	1	1	1
2	2	2	2	2	2
3	4	8	8	4	8
4	8	32	40	12	48
5	16	128	224	48	384
6	32	512	1344	240	3840
7	64	2048	8448	1440	46080
8	128	8192	54912	10080	645120
9	256	32768	366080	80640	10321920
10	512	131072	2489344	725760	18579450

Sample Numbers, with Cross Products

n	Left-Deep $n!$	Zig-Zag $n!2^{n-2}$	Bushy $n!\mathcal{C}(n-1)$
1	1	1	1
2	2	2	2
3	6	12	12
4	24	96	120
5	120	960	1680
6	720	11520	30240
7	5040	161280	665280
8	40320	2580480	17297280
9	362880	46448640	518918400
10	3628800	968972800	17643225600

Problem Complexity

query graph	join tree	cross products	cost function	complexity
general	left-deep	no	ASI	NP-hard
tree/star/chain	left-deep	no	ASI, 1 joint.	P
star	left-deep	no	NLJ+SMJ	NP-hard
general/tree/star	left-deep	yes	ASI	NP-hard
chain	left-deep	yes	-	open
general	bushy	no	ASI	NP-hard
tree	bushy	no	-	open
star	bushy	no	ASI	P
chain	bushy	no	any	P
general	bushy	yes	ASI	NP-hard
tree/star/chain	bushy	yes	ASI	NP-hard

Greedy Heuristics - First Algorithm

- search space of joins trees is very large
- greedy heuristics produce suitable join trees very fast
- suitable for large queries

For the first algorithm we consider:

- left-deep trees
- no cross products
- relations ordered to some weight function (e.g. cardinality)

Note: the algorithms produces a sequence of relations; it uniquely identifies the left-deep join tree.

Greedy Heuristics - First Algorithm (2)

GreedyJoinOrdering-1($R = \{R_1, \ldots, R_n\}$,$w : R \rightarrow \mathbb{R}$)
Input: a set of relations to be joined and weight function
Output: a join order
$S = \epsilon$
while ($|R| > 0$) {
  $m = \arg\min_{R_i \in R} w(R_i)$
  $R = R \setminus \{m\}$
  $S = S \circ < m >$
}
return S

- disadvantage: fixed weight functions
- already chosen relations do not affect the weight
- e.g. does not support minimizing the intermediate result

Greedy Heuristics - Second Algorithm

GreedyJoinOrdering-2($R = \{R_1, \ldots, R_n\}$,$w : R, R^* \to \mathbb{R}$)
Input: a set of relations to be joined and weight function
Output: a join order
$S = \epsilon$
while $(|R| > 0)$ {
  $m = \arg\min_{R_i \in R} w(R_i, S)$
  $R = R \setminus \{m\}$
  $S = S \circ < m >$
}
return S

- can compute relative weights
- but first relation has a huge effect
- and the fewest information available

Greedy Heuristics - Third Algorithm

GreedyJoinOrdering-3($R = \{R_1, \ldots, R_n\}, w : R, R^* \rightarrow \mathbb{R}$)
Input: a set of relations to be joined and weight function
Output: a join order

$S = \emptyset$
for $\forall R_i \in R$ {
  $R' = R \setminus \{R_i\}$
  $S' = < R_i >$
  while $(|R'| > 0)$ {
    $m = \arg\min_{R_j \in R'} w(R_j, S')$
    $R' = R' \setminus \{m\}$
    $S' = S' \circ < m >$
  }
  $S = S \cup \{S'\}$
}
return $\arg\min_{S' \in S} w(S'[n], S'[1 : n-1])$

- commonly used: minimize selectivities (*MinSel*)

Greedy Operator Ordering

- the previous greedy algorithms only construct left-deep trees
- Greedy Operator Ordering (GOO) [1] constructs bushy trees

Idea:

- all relations have to be joined somewhere
- but joins can also happen between whole join trees
- we therefore greedily combine join trees (which can be relations)
- combine join trees such that the intermediate result is minimal

Greedy Operator Ordering (2)

GOO($R = \{R_1, \ldots, R_n\}$)
Input: a set of relations to be joined
Output: a join tree
T = R
while $|T| > 1$ {
  $(T_i, T_j) = \arg\min_{(T_i \in T, T_j \in T), T_i \neq T_j} |T_i \bowtie T_j|$
  $T = (T \setminus \{T_i\}) \setminus \{T_j\}$
  $T = T \cup \{T_i \bowtie T_j\}$
}
return $T_0 \in T$

- constructs the result bottom up
- join trees are combined into larger join trees
- chooses the pair with the minimal intermediate result in each pass