3. Join Ordering

- Basics
- Search Space
- Greedy Heuristics
- IKKBZ
- MVP
- Dynamic Programming
- Generating Permutations
- Transformative Approaches
- Randomized Approaches
- Metaheuristics
- Iterative Dynamic Programming
- Order Preserving Joins

Queries Considered

Concentrate on join ordering, that is:

- conjunctive queries
- simple predicates
- predicates have the form $a_1 = a_2$ where a_1 is an attribute and a_2 is either an attribute or a constant
- even ignore constants in some algorithms

We join relations $R_1, \ldots, R_n$, where R_i can be

- a base relation
- a base relation including selections
- a more complex building block or access path

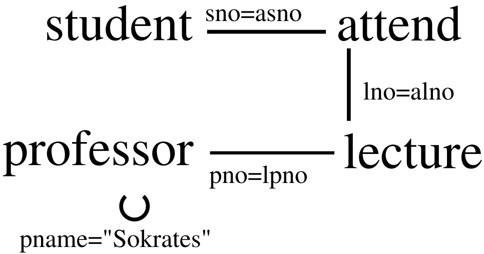
Pretending to have a base relation is ok for now.

Query Graph

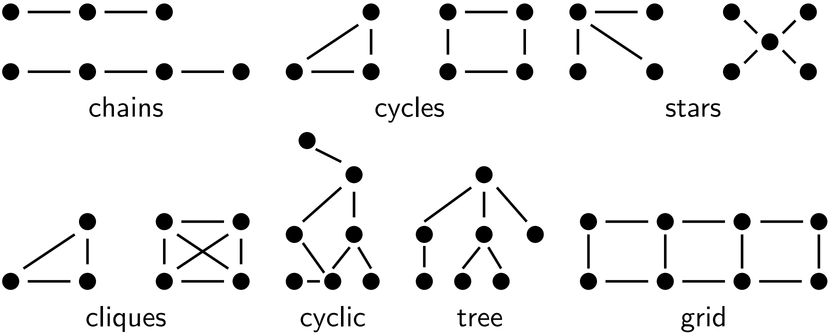
Queries of this type can be characterized by their query graph:

- the query graph is an undirected graph with $R_1, \ldots, R_n$ as nodes
- a predicate of the form $a_1 = a_2$, where $a_1 \in R_i$ and $a_2 \in R_j$ forms an edge between R_i and R_j labeled with the predicate
- a predicate of the form $a_1 = a_2$, where $a_1 \in R_i$ and a_2 is a constant forms a self-edge on R_i labeled with the predicate
- most algorithms will not handle self-edges, they have to be pushed down

Sample Query Graph



Shapes of Query Graphs



- real world queries are somewhere in-between
- chain, cycle, star and clique are interesting to study
- they represent certain kind of problems and queries

Join Trees

A join tree is a binary tree with

- join operators as inner nodes
- relations as leaf nodes

Algorithms will produce different kinds of join trees

- ordered or unordered
- with cross products or without

The most common case is ordered, without cross products

Shape of Join Trees

Commonly used classes of join trees:

- left-deep tree
- right-deep tree
- zigzag tree
- bushy tree

The first three are summarized as *linear trees*.

Join Selectivity

Input:

- cardinalities $|R_i|$
- selectivities $f_{i,j}$: if $p_{i,j}$ is the join predicate between R_i and R_j, define

$$f_{i,j} = \frac{|R_i \bowtie_{p_{i,j}} R_j|}{|R_i \times R_j|}$$

Calculate:

- result cardinality:

$$|R_i \bowtie_{p_{i,j}} R_j| = f_{i,j}|R_i||R_j|$$

Rational: The selectivity can be computed/estimated easily (ideally).

Cardinality of Join Trees

Given a join tree T, the result cardinality $|T|$ can be computed recursively as

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

- allows for easy calculation of join cardinality
- requires only base cardinalities and selectivities
- assumes independence of the predicates

Sample Statistics

As running example, we use the following statistics:

$$
\begin{aligned}
|R_1| &= 10 \\
|R_2| &= 100 \\
|R_3| &= 1000 \\
f_{1,2} &= 0.1 \\
f_{2,3} &= 0.2
\end{aligned}
$$

- implies query graph $R_1 - R_2 - R_3$
- assume $f_{i,j} = 1$ for all other combinations

A Basic Cost Function

Given a join tree T, the cost function C_{out} is defined as

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

- sums up the sizes of the (intermediate) results
- rational: larger intermediate results cause more work
- we ignore the costs of single relations as they have to be read anyway

Basic Join Specific Cost Functions

For single joins:

$$
\begin{aligned}
C_{nlj}(e_1 \bowtie e_2) &= |e_1||e2| \\
C_{hj}(e_1 \bowtie e_2) &= 1.2|e_1| \\
C_{smj}(e_1 \bowtie e_2) &= |e_1|\log(|e_1|) + |e_2|\log(|e_2|)
\end{aligned}
$$

For sequences of join operators $s = s_1 \bowtie \ldots \bowtie s_n$:

$$
\begin{aligned}
C_{nlj}(s) &= \sum_{i=2}^{n} |s_1 \bowtie \ldots \bowtie s_{i-1}||s_i| \\
C_{hj}(s) &= \sum_{i=2}^{n} 1.2|s_1 \bowtie \ldots \bowtie s_{i-1}| \\
C_{smj}(s) &= \sum_{i=2}^{n} |s_1 \bowtie \ldots \bowtie s_{i-1}|\log(|s_1 \bowtie \ldots \bowtie s_{i-1}|) + \sum_{i=2}^{n} |s_i|\log(|s_i|)
\end{aligned}
$$

Remarks on the Basic Cost Functions

- cost functions are simplistic
- algorithms are modelled very simplified (e.g. 1.2, no n-way sort etc.)
- designed for left-deep trees
- C_{hj} and C_{smj} do not work for cross products (fix: take output cardinality then, which is C_{nl})
- in reality: other parameters than cardinality play a role
- cost functions assume the same join algorithm for the whole join tree

Sample Cost Calculations

	C_{out}	C_{nl}	C_{hj}	C_{smj}
$R_1 \bowtie R_2$	100	1000	12	697.61
$R_2 \bowtie R_3$	20000	100000	120	10630.26
$R_1 \times R_3$	10000	10000	10000	10000.00
$(R_1 \bowtie R_2) \bowtie R_3$	20100	101000	132	11327.86
$(R_2 \bowtie R_3) \bowtie R_1$	40000	300000	24120	32595.00
$(R_1 \times R_3) \bowtie R_2$	30000	1010000	22000	143542.00

- costs differ vastly between join trees
- different cost functions result in different costs
- the cheapest plan is always the same here, but relative order varies
- join trees with cross products are expensive
- join order is essential under all cost functions

More Examples

For the query $|R_1| = 1000, |R_2| = 2, |R_3| = 2, f_{1,2} = 0.1, f_{1,3} = 0.1$ we have costs:

	C_{out}
$R_1 \bowtie R_2$	200
$R_2 \times R_3$	4
$R_1 \bowtie R_3$	200
$(R_1 \bowtie R_2) \bowtie R_3$	240
$(R_2 \times R_3) \bowtie R_1$	44
$(R_1 \bowtie R_3) \bowtie R_2$	240

- here cross product is best
- but relies on the small sizes of $|R_2|$ and $|R_3|$
- attractive if the cardinality of one relation is small

More Examples (2)

For the query $|R_1| = 10, |R_2| = 20, |R_3| = 20, |R_4| = 10, f_{1,2} = 0.01, f_{2,3} = 0.5, f_{3,4} = 0.01$
we have costs:

	C_{out}
$R_1 \bowtie R_2$	2
$R_2 \bowtie R_3$	200
$R_3 \bowtie R_4$	2
$((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$	24
$((R_2 \times R_3) \bowtie R_1) \bowtie R_4$	222
$(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)$	6

- covers all join trees due to the symmetry of the query
- the bushy tree is better than all join trees

Symmetry and ASI

- cost function C_{impl} is called *symmetric* if $C_{impl}(e_1 \bowtie^{impl} e_2) = C_{impl}(e_2 \bowtie^{impl} e_1)$
- for symmetric cost functions commutativity can be ignored
- ASI: *adjacent sequence interchange* (see IKKBZ algorithm for a definition)

Our basic cost functions can be classified as:

	ASI	$\neg$ASI
symmetric	C_{out}	C_{smj}
$\neg$symmetric	C_{hj}	-

- more complex cost functions are usually $\neg$ASI, often also $\neg$symmetric
- symmetry and especially ASI can be exploited during optimization

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