Solutions for Exercise Sheet 4

Richter, Yannick
MTK 03741982
ge78tup@mytum.de

Rodrigues, Diogo
MTK 03770446
diogo.rodrigues@tum.de

TUM – Query Optimization 2022/23
18th November 2022

Our solutions for Exercise Sheet 4.

Exercise 1

Give an example query instance (query graph with selectivities and cardinalities) where the optimal join tree (using C_{out}) is truly bushy and includes a cross product. A join tree is truly bushy if it is not a zig-zag tree. Note: the query graph should be connected and should not contain explicit cross products!

Assume a query with tables R_1, R_2, R_3, R_4 with $|R_1| = |R_2| = |R_3| = 10$, $|R_4| = 2$, $f_{1,2} = f_{2,3} = f_{3,4} = 0.5$.

$$|R_1 \bowtie R_2| = |R_1| \cdot |R_2| \cdot f_{1,2} = 10 \cdot 10 \cdot 0.5 = 50$$
$$C_{out}(R_1 \bowtie R_2) = |R_1 \bowtie R_2| = 50$$
$$|R_3 \bowtie R_4| = |R_3| \cdot |R_4| \cdot f_{3,4} = 10 \cdot 2 \cdot 0.5 = 10$$
$$C_{out}(R_3 \bowtie R_4) = |R_3 \bowtie R_4| = 10$$

$$|(R_1 \bowtie R_2) \bowtie R_3| = |R_1 \bowtie R_2| \cdot |R_3| \cdot f_{2,3} = 50 \cdot 10 \cdot 0.5 = 250$$
$$C_{out}((R_1 \bowtie R_2) \bowtie R_3) = C_{out}(R_1 \bowtie R_2) + |(R_1 \bowtie R_2) \bowtie R_3| = 50 + 250 = 300$$

$$|((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4| = |(R_1 \bowtie R_2) \bowtie R_3| \cdot |R_4| \cdot f_{3,4} = 250 \cdot 2 \cdot 0.5 = 250$$
$$C_{out}(((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4) = C_{out}((R_1 \bowtie R_2) \bowtie R_3) + |((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4| = 300 + 250 = 550$$
$$|(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)| = |R_1 \bowtie R_2| \cdot |R_3 \bowtie R_4| \cdot f_{2,3} = 50 \cdot 10 \cdot 0.5 = 250$$
$$C_{out}((R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)) = C_{out}(R_1 \bowtie R_2) + C_{out}(R_3 \bowtie R_4) + |(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)| = 50 + 10 + 250 = 310$$

	C_{out}
$((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$	550
$(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)$	310

We have created an optimal query tree using the C_{out} cost function, which is a bushy tree. Now we have to show that the translation of logical to physical operators yields at least one nested loop join, which is essentially equivalent to a cross product followed by a selection.

Let us assume the join condition between R_3 and R_4 is a range condition, so we cannot use hash join for this join.

The costs of different operators are:

- $C_{nlj} = |R_3| \cdot |R_4| = 10 \cdot 2 = 20$
- $C_{smj} = |R_1| \log_2 |R_1| + |R_2| \log_2 |R_2| = 10 \log_2 10 + 2 \log_2 2 = 10 \cdot 3.322 + 2 \cdot 1 = 35.22$

So the cheapest way to calculate $R_3 \bowtie R_4$ is to do a cross product.

Exercise 2

Give an algorithm that computes a bijective function $f_n : [0, 2^{n-1}) \mapsto \mathcal{T}$ where $\mathcal{T}$ is the set of left-deep join trees without cross products for a chain query of n relations. Note that every integer in the range $[0, 2^{n-1})$ maps to a distinct tree in $\mathcal{T}$ and every tree in $\mathcal{T}$ corresponds to a distinct integer in the range $[0, 2^{n-1})$. Explain why these properties hold for your algorithm (no need for a formal proof, a proof sketch is enough). Hints:

- Imagine you have a start node R_0 in an infinitely long chain query that extends to both sides ... $-$ $R_{-1} - R_0 - R_1 -$ .... There are two possible joins with R_0. We either grow left and join with R_{-1} or grow right and join with R_1.
- Given an infinite query graph, a start node R_0, and a sequence of grow left/grow right operations $(l, r, l, l, r, ...) \in \{l, r\}^{n-1}$, how would you build a join tree without cross products containing n relations (including R_0)?
- Note that the query graph you are given is not infinitely large and you do not have a fixed start node. How do you deal with this?
- Build up the solution step by step. Define the functions $g_n : [0, 2^{n-1}) \mapsto \{l, r\}^{n-1}$ and $h_n : \{l, r\}^{n-1} \mapsto \mathcal{T}$ where $f_n(i) = h_n(g_n(i))$.

```
function F(n, x)
    v ← G(n, x)
    return H(n, v)
function G(n, x)                                  ▷ Obtains binary representation of x
    for i ∈ [0, n) do
        v(i) ← (x%2 ? 'r' : 'l')
        x ← x/2                                   ▷ Integer division
    return v
function H(n, v, relation)
    l ← #_l(v)                                    ▷ #_l(v) is the number of times 'l' appears in v
    r ← l + 1
    T ← relation[l]
    for i ∈ [0, n) do
        if v(i) = 'l' then
            m ← l − 1
            l ← l − 1
        else
            m ← r
            r ← r + 1
        T = JOIN(T, relation[m])
    return T
```

We want to prove that function f_n is bijective. This is equivalent to proving that g_n and h_n are bijective.

- The proof that g_n is bijective is trivial. It is simply the conversion of an input number x into its binary representation $v \in \{l, r\}^{n-1}$, where we can declare either $'r'$ or $'l'$ to be 0 and the other to be 1.
- The proof that h_n is bijective can be made in an inductive way. h_1 is trivially bijective, because $x = ()$ maps to the tree that only contains node R_0. The invariant is that h_n is bijective, so we want to prove that, if h_n is bijective, then h_{n+1} is also bijective. h_{n+1} can be produced from h_n by appending either a '0' or a '1' at the end of the input. If we only had one operation available (say, we can only append a '0' to generate h_{n+1} from h_n), it is trivial that it is still a bijective function (same applies if the only available operation is to append '1'). If we can use both operations (append '0' or append '1'), we want to prove that, for any numbers $x, y \in 2^{n-2}$, that $x \circ$'0' and $y \circ$'1' cannot represent the same tree. This

step can be proven if we consider that the top-most node of $x \circ$ '0' will have an index < 0 (because it is picked from the left side) and the top-most node of $y \circ$ '1' will have an index > 0 (because it is picked from the right side), so obviously $x \circ$ '0' and $y \circ$ '1' do not originate the same tree.