Draft

# 1 Join Tree

Throughout these sections, for the sake of simplicity, we are dealing only with Boolean Conjunctive Query (BCQ) and we call them query. These result can be extended to other queries. Let fix a given query as $Q$. Suppose for $Q$ we have its join tree $JT_Q < V >$ which $V, |V| = n$ represents the set of its nodes (i.e. Figure 1). This join tree has several nice properties. The most important property of join trees is that we can compute their results in linear time ???. The other important property of join trees is *connectedness property*. By the connectedness we mean that if we select a variable which occurs in the given query, ???.

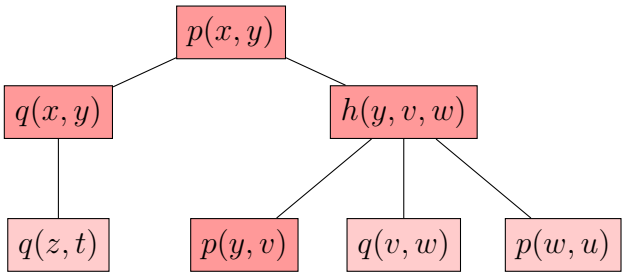


Figure 1: A sample join tree of a query

## 1.1 Number of different nodes is polynomial

Let fix $k$ as the tree width of given $JT_Q$. Here we consider the case when $k$ is constant (we have an upper bound for that). When we apply the $\Delta$-operator to a join tree its nodes change also the structure. Here we prove

that the number of all nodes in all different trees that emerge from all different sequence of applying the $\Delta$-operator is polynomial in term of $n$ for a fixed $k$.

The nodes of $JT_Q$ are all in form of $R_{i_1} \bowtie \cdots \bowtie R_{i_m}$ which $m \leq k$, $m$ relations join together. Lets fix one particular node as $l$ with $m$ relations and the set of its indices is $I$. If we apply $\Delta_{\pm R_{i_p}}$ when $i_p \notin I$ this node is $\emptyset$. For the other case we will have node $R_{i_1} \bowtie \cdots R_{i_{p-1}} \bowtie R_{i_{p+1}} \cdots R_{i_m}$, in the other words relation $R_{i_p}$ was deleted from the join operation. Thus, for counting the number of all nodes in all $JT_Q$ for all sequences of applying $\Delta$-operator, we should count all nodes which contain $1, 2, \cdots, k$ different relations. This number is $\sum_{i=1}^{k} \binom{n}{i} < (n+1)^k$ which is polynomial for a fixed $k$.

## 1.2 Cost of Evaluation

According to the last section the number of nodes in all delta trees is polynomial for a fixed $k$. Now we want to prove that the structures of all these trees are polynomial for a fixed $k$.

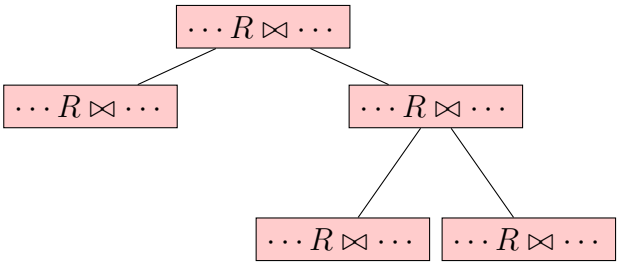


Figure 2: ??

When $\Delta_R$-operator applies to a join tree it doesn't change the nodes that do not contain $R$ relation. Thus, without loss of generality we can assume that $R$ has appeared in all nodes, since we have connectedness property.

When we apply $\Delta_R$-operator the such a tree, each node splits into 3 different nodes($\Delta(R \bowtie S) = \Delta(R) \bowtie S + R \bowtie \Delta(S) + \Delta(R) \bowtie \Delta(S)$). If we consider each $+, \bowtie$ as one operation, evaluation of $\Delta(R \bowtie S)$ needs

at most 5 operations. Thus, the cost of evaluation of all $\Delta$ trees is $O(n) *$ (Cost of join operator).

### 1.3 Hypertree Decomposition

Gottlob et al. have shown that for each Hypertree there exists a join tree with the same tree structure(Lemma 4.6). Thus, we have these results for hypertrees too.

## 2 Algorithms

In this section we give some algorithms for converting the $JT_Q$ into the proper data structures which are used in DBToaster. Suppose we want to compute $\Delta_{\pm R} JT_Q$. As shown in 1.2 we can assume that every node contains relation $R$ (i.e. 3).

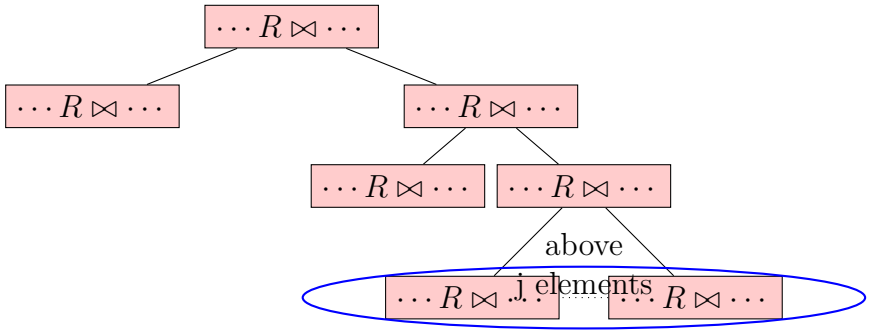


Figure 3: ??

**Algorithm 1** Computing $\Delta_{\pm R}$

**Input:** node $v$ as the root of $JT_Q$ and relation $R$ as the $\Delta$ variable.
**Output:** Calculus expression for $\Delta_R$ for node $v$

1: $J \leftarrow \emptyset$
2: **for all** Child $i$ of node $v$ **do**
3:  **if** the subtree rooted at $i$ has any node labeled by $R$ **then**
4:  Computes the $\Delta_R$ of $i$ recursively and store it in $\Delta[i]$ if it is not already computed.
5:  add $i$ to $J$
6:  **end if**
7: **end for**
8: $value \leftarrow ComputeChain(J)$
9: **return** $value.expr$

**Algorithm 2** ComputeChain

**Input:** A set of sibling nodes of $JT_Q$ as $J$ and relation $R$ as the $\Delta$ variable.
**Output:** Calculus expression for $\Delta_R$ for input nodes

1: **if** $|J| = 1$ **then**
2:  $expr \leftarrow \Delta_R(J_1)$ $\{J_1$ is the first element of $J\}$
3:  $join \leftarrow J_1$
4: **else**
5:  $rest \leftarrow ComputeChain(J \setminus J_1)$
6:  $join \leftarrow J_1 \bowtie rest.join$
7:  $expr \leftarrow rest.expr \bowtie \Delta(J_1) + \Delta(J_1) \bowtie rest.join + J_1 \bowtie rest.expr$
8: **end if**
9: **return** $(exp, join)$

Obviously this algorithm needs to compute the delta of each element just one time. If we want to compute the $\Delta$ of a node with branching factor $j$ it needs $2^j - 1$ different joint operation($\bowtie$) which is exponential in the worst case. But we can use a simple trick here. For the node we can first consider $\Delta$ of $j-1$ children and store it somewhere then with this value we can merge the $j$th one. With Algorithm. 2 we can compute the $\Delta$ in $O(n)$.