CS 784: Foundations of Data Management *Instructor: Paris Koutris*

Lecture 7: Worst-Case Optimal Joins

In this lecture, we ask the following question: *can we compute a full Conjunctive Query in running time bounded by the AGM bound?* We will answer this positively, and describe two algorithmic techniques that can achieve this optimal result. But first we will sketch the main technical ideas using as an example the triangle query.

7.1 Computing the Triangle Query

Consider the triangle query

$$\Delta(x, y, z) \text{ :- } R(x, y), S(y, z), T(z, x)$$

where the three relations have sizes N_R, N_S, N_T respectively. Recall that the AGM bound gives a bound of $(N_R N_S N_T)^{1/2}$ for the output size, which becomes $N^{3/2}$ when all relation sizes are equal to N. With the next example we show that it is *not* possible to achieve a running time of $O(N^{3/2})$ using only join-project plans.

Example 7.1. *Let $\{a_0, \ldots, a_n\}$, $\{b_0, \ldots, b_n\}$, $\{c_0, \ldots, c_n\}$ be the domains for variables x, y, z respectively. We define an instance I as follows:*

$$R^I = (\{a_0\} \times \{b_0, \ldots, b_n\}) \cup (\{a_0, \ldots, a_n\} \times \{b_0\})$$
$$S^I = (\{b_0\} \times \{c_0, \ldots, c_n\}) \cup (\{b_0, \ldots, b_n\} \times \{c_0\})$$
$$T^I = (\{c_0\} \times \{a_0, \ldots, a_n\}) \cup (\{c_0, \ldots, c_n\} \times \{a_0\})$$

Each relation has size $2n + 1$, and the output has size $3n + 1$. However, observe that every pairwise join ($R \bowtie S$, $S \bowtie T$, $R \bowtie T$) has size $n^2 + n = \Omega(n^2)$. Hence, no join plan can achieve the $O(n^{3/2})$ bound.

The reason that the join plan from the above example fails is that the values a_0, b_0, c_0 are all *skewed*. To overcome this obstacle, we will deal with such values differently. We say that a value a is *heavy* if $|\sigma_{x=a}(R)| \cdot |\sigma_{x=a}(T)| \geq |S|$; otherwise, it is *light*.

P2C algorithm. The Power of 2 Choices algorithm works as follows:

1. Let $L \leftarrow \pi_x(R) \cap \pi_x(T)$.
2. For each $a \in L$ do:
   (a) If a is *light*: compute $\sigma_{x=a}(R) \bowtie \sigma_{x=a}(T)$ and semijoin the result with S.
   (b) If a is *heavy*: for each tuple $(b, c) \in S$, check whether $(a, b) \in R$ and $(c, a) \in T$.

Analysis. Assume for now that we have computed the appropriate indices for each relation (which we can do in time linear to the input size). It is easy to see that we can compute the set L in linear time. For the main loop, the key observation is that the running time of case (a) is bounded by $|\sigma_{x=a}R| \cdot |\sigma_{x=a}T|$, while the running time of case (b) is bounded by N_S. Hence, the total time T can be bounded as follows:

$$\begin{aligned}
T &= \sum_{a \in L} \min\{|\sigma_{x=a}R| \cdot |\sigma_{x=a}T|, N_S\} \\
&\leq \sum_{a \in L} (|\sigma_{x=a}R| \cdot |\sigma_{x=a}T| \cdot N_S)^{1/2} \\
&= N_S^{1/2} \sum_{a \in L} (|\sigma_{x=a}R| \cdot |\sigma_{x=a}T|)^{1/2} \\
&\leq N_S^{1/2} \left(\sum_{a \in L} |\sigma_{x=a}R| \right)^{1/2} \cdot \left(\sum_{a \in L} |\sigma_{x=a}T| \right)^{1/2} \\
&\leq N_S^{1/2} N_R^{1/2} N_T^{1/2}
\end{aligned}$$

Here, the first inequality used the fact that $\min\{x, y\} \leq \sqrt{xy}$, while the second inequality is an application of the Cauchy-Schwarz inequality.

7.2 The GenericJoin algorithm

We now discuss a general algorithmic framework that works for all CQs called GenericJoin [NRR13]. Let $H(q) = (\mathcal{V}, \mathcal{E})$ be the hypergraph for the query q. For any $I \subseteq \mathcal{V}$, define:

$$\mathcal{E}_I = \{F \in \mathcal{E} \mid F \cap I \neq \emptyset\}$$

We use R_F to denote the relation that corresponds to hyperedge F. GenericJoin works by recursively splitting the variables into two disjoint sets I and $J = \mathcal{V} \setminus I$.

Algorithm 1: GenericJoin

Input: hypergraph $(\mathcal{V}, \mathcal{E})$

$OUT \leftarrow \emptyset$;
if $|\mathcal{V}| = 1$ **then**
  return $\cap_{F \in \mathcal{E}} R_F$;
end
pick $I : \emptyset \subsetneq I \subsetneq \mathcal{V}$;
$L \leftarrow GenericJoin(\bowtie_{F \in \mathcal{E}_I} \pi_I(R_F))$;
for $t_I \in L$ **do**
  $J \leftarrow \mathcal{V} \setminus I$;
  $q[t_I] \leftarrow GenericJoin(\bowtie_{F \in \mathcal{E}_J} \pi_J(R_F \ltimes t_I))$;
  $OUT \leftarrow OUT \cup q[t_I]$;
end
return OUT

To achieve the desired runtime, an implementation of GenericJoin must satisfy the following:

- The intersection $\cap_{F \in \mathcal{E}} R_F$ must be computable in time $\min_F |R_F|$, i.e., depending only on the smallest relation.
- The subrelations $R_F \ltimes t_I$ must be able to be computed in constant time. This means that it is necessary to construct a data structure that allows us to access and iterate over that subpart of the relation as fast as possible.

Implementation. We next discuss a concrete implementation of the GenericJoin algorithm, called Leapfrog Triejoin [V14]. Leapfrog Triejoin is implemented as part of the LogicBlox Datalog engine. It has the following characteristics:

- It picks I at every step to be of size 1 (i.e., picks a single variable). Any order of picking the variables will work (in the worst-case analysis), but in practice certain variable orders can be orders of magnitude faster. Finding the best variable sequence is a difficult optimization problem!
- It merges unary relations (of arity 1) using Leapfrog join. If the relations are already sorted (according to the same total order), then Leapfrog join needs time linear with respect to the *smallest* relation.
- The relations are indexed using *tries*. A trie is a tree with depth equal to the arity of the relation (plus 1). Each level has values of a particular variable, and each tuple is represented by a path from the root of the trie to a leaf. Moreover, the children of each node are distinct from one another and sorted, with the leftmost child being the smallest. The variable ordering in a trie must agree with the ordering chosen by the Leapfrog Triejoin algorithm.

Analysis. Finally, we analyze the running time of GenericJoin. Let $\{u_F \mid F \in \mathcal{E}\}$ be a fractional edge cover for H. We analyze the running time for GenericJoin using induction on the size of $\mathcal{V}$.

For the base case where $|\mathcal{V}| = 1$, we perform merging between unary relations. If these are sorted, we can compute their intersection in time $\tilde{O}(\min_F |R_F|) = \tilde{O}(\prod_F |R_F|^{u_F})$.

For the inductive step, the running time is (asymptotically):

$$\prod_{F \in \mathcal{E}_I} |R_F|^{u_F} + \sum_{t_I \in L} \prod_{F \in \mathcal{E}_J} |R_F \ltimes t_I|^{u_F} \leq \prod_{F \in \mathcal{E}_I} |R_F|^{u_F} + \prod_{F \in \mathcal{E}} |R_F|^{u_F} \leq 2 \prod_{F \in \mathcal{E}} |R_F|^{u_F}$$

Here, the second inequality comes from the so-called query decomposition lemma [NRR13].

7.3 Fractional Hypertree Width

We can now combine the worst-case optimal join algorithm with tree decompositions to obtain faster evaluation algorithms for any CQ. Recall that to define the ghw we used the minimum

cover as the width of each bag in the decomposition. By replacing this with the *minimum fractional edge cover*, we obtain instead the *fractional hypertree width (fhw)* of a CQ. We always have that $fhw \leq ghw$, but in fact fhw can be much smaller than ghw.

Example 7.2. *The ghw of the triangle query is $ghw(\Delta) = 2$, since the size of the minimum cover for the bag that contains all three variables is 2 (it is necessary to pick at least two relations to cover each variable). However, $fhw(\Delta) = 3/2$.*

Hence, we can evaluate any full CQ in time $O(N^{fhw} + OUT)$, where N is the input size.

References

[Alice] S. ABITEBOUL, R. HULL and V. VIANU, "Foundations of Databases."

[AGM08] A. ATSERIAS, M. GROHE and D. MARX, "Size bounds and query plans for relational joins," *FOCS 2008*.

[NRR13] H. NGO, C. RE and A. RUDRA, "Skew Strikes Back: New Developments in the Theory of Join Algorithms," *SIGMOD Record, 2013*.

[V14] T. VELDHUIZEN, "Leapfrog Triejoin: a worst-case optimal join algorithm," *ICDT, 2014*.