CS 838: Foundations of Data Management **Spring 2016**

Lecture 4: Acyclic Joins

Instructor: Paris Koutris 2016

Recall that we showed that the combined complexity of evaluating CQs is NP-complete. In this lecture, we show how to obtain polynomial time algorithms for some classes of CQs. Let us start with an example.

Example 4.1. *Consider the boolean path query, which asks whether there is path of length k in the directed graph represented by the edge relation R.*

$$P^k() : -R(x_1, x_2), R(x_2, x_3), \ldots, R(x_k, x_{k+1})$$

Assume that $|R| = n$. Consider the following algorithm: we first project R on the attribute x_2, then compute the semi-join with R_2, project the result on attribute x_3, and so on. Observe that at every point the size of the intermediate result is at most n. Hence, we can implement the algorithm with running time time $O(kn \log(n))$. The running time in this case is polynomial (in combined complexity)!

Let's see now how we can generalize this intuition to compute a larger class of queries. Notice that the output of a query may be in the worst case exponential in the input size; in this case, designing an algorithm that runs in polynomial time in the size of the input is not possible. Thus, we want to design an algorithm that is polynomial with respect to the *input and the output size*. We will show that such an algorithm is possible for the class of conjunctive queries called *acyclic CQs*.

4.1 Acyclic CQs: Definition

Definition 4.2. *The hypergraph $H(q) = (V, E)$ associated to a conjunctive query q is defined as follows. The set of vertices V consists of all variables in the body of q, $\mathbf{var}(q)$, while the set E of hyperedges contains, for each atom in the query, the set of variables that appear in this atom.*

Exercise 4.3. *Construct the hypergraphs for the query $q_0(x, y, z) : -R(x, y), S(y, z), T(z, x)$ and $q_1() : -R(x, y), S(y, z), T(z, w)$.*

Intuitively, a CQ is acyclic if the hypergraph contains no cycles. If the hypergraph is actually a graph (which happens if every relation is binary), then query acyclicity coincides with the graph being a tree. But the situation is not as clear when the hyperedges have arity more than 2. To formally define acyclicity, we start with the following algorithm, called the GYO algorithm (from Graham-Yu-Ozsoyoglu).

An *ear* in the hypergraph H is a hyperedge e for which we can divide its nodes into two groups: (a) those that appear exclusively in e, and (b) those that are contained in another hyperedge f of

H. For example, if a hyperedge is isolated, or if it is contained within another hyperedge, it is by definition an ear.

Example 4.4. *For the query q_1, both R and T are ears, but S is not. The query q_0 contains no ears.*

The GYO algorithm works as follows:

- Until H has no ears
  - Pick an ear e of H.
  - Remove the ear from the set E and all the vertices that belong exclusively to e (and not any other hyperedge) from the set V.

Example 4.5. *If we apply the GYO algorithm to $H(q_0)$, since no hyperedge is an ear, the output will be just $H(q_0)$. For q_1, suppose we start with T. Then, the remaining hypergraph is $R(x,y), S(y,z)$. Now both R, S are ears; if we remove S we get only $R(x,y)$. Since any isolated hyperedge is an ear, the resulting hypergraph is the empty one.*

Let $GYO(H)$ be the resulting hypergraph if we apply the GYO algorithm to H. It turns out that we produce always the same hypergraph, independent of the order that we choose to visit the ears of the hypergraph. We say that H is *acyclic* if $GYO(H)$ is the empty hypergraph.

Definition 4.6 (CQ Acyclicity)**.** *A conjunctive query q is* acyclic *if $GYO(H(q))$ is the empty hypergraph.*

This is a type of acyclicity that is called α-acyclicity. There are other definitions of acyclicity for hypergraphs (Berge-acyclicity, β-acyclicity, γ-acyclicity), but they are more restrictive. Next, we will see some equivalent definitions of acyclic CQs.

Definition 4.7 (Join Forest)**.** *A* join forest *for a conjunctive query q is a forest*[1] *$F = (V, E)$ whose vertices are the atoms in q and such that for each pair of atoms R, S having variables in common the following conditions hold:*

1. *R, S are in the same connected component of F, and*
2. *all variables common to R and S occur on the unique path from R to S.*

If the forest is actually a tree, it is called a *join tree* of the query q.

Lemma 4.8. *A conjunctive query q is acyclic if and only if it admits a join forest.*

We next show a third equivalent characterization of acyclic CQs. Say that, given a query q, we want to remove from a relation R the tuples that we are certain that they do not participate in the final result of q, called *dangling tuples*. We can attempt to do this using the *semijoin* operator. A semijoin between relations R, S is defined as:

$$R \ltimes S = \pi_{\mathbf{var}(R)}(R \bowtie S)$$

[1] A forest is a graph that is a collection of disjoint trees.

In other words, the result is the set of tuples from R that we are certain that they join with a tuple from S. Semijoins are often used in distributed databases to reduce the communication cost of joining two relations. An instance without any dangling tuples is called *globally consistent*. Can we remove all dangling tuples by using only a (finite) sequence semijoin operations? Such a sequence is called a *full reducer*.

Example 4.9. *Here is the full reducer for the query P:*

$$S := S \ltimes R, R := R \ltimes S, S := S \ltimes R, T := T \ltimes S.$$

The triangle query T does not admit any full reducer. For example, consider the following instance: $\{R(a,a), R(b,b), S(a,a), S(b,b), T(a,b), T(b,a)\}$.

Lemma 4.10. *A conjunctive query q is acyclic if and only it admits a full reducer.*

[Alice] contains a detailed proof of this equivalence result. The idea is to compute semi-joins in the order in which we remove the ears in the GYO algorithm (and in the reverse order as well).

4.2 Acyclic CQs: PTIME Algorithm

We will now use join forests and full reducers to compute acyclic CQs. Without loss of generality, we can assume that the query q is connected (since we can compute each connected component separately and then take the cartesian product). The algorithm works as follows:

1. Apply a full reducer to obtain a globally consistent instance.
2. Construct a join tree T for q and choose a root R.
3. Choose a join order in which each node is joined with its parent in any bottom-up order.
4. After each join into a relation R, project the result onto the set of attributes that are either in $\mathbf{var}(R) \cup \mathbf{head}(q)$.

Theorem 4.11. *We can evaluate an acyclic conjunctive query q in time polynomial in the size of the query, the input and the output.*

Proof. Why is this algorithm correct? The key property is that if we have a variable x that appears in a node R and not its parent in the join tree, the variable will not appear at any ancestor (so it will not be needed further in the evaluation).

To show that we obtain the desired complexity, we will prove that no intermediate result can grow more than a polynomial of the input and output. Consider some node with atom R and children $S_1, \ldots, S_n$. Then the algorithm will replace this node with a new node $T = \pi_{Y \cup Z}(T)$, where $T = R \bowtie S_1 \bowtie \ldots S_n$, Y are the variables in R, and Z are the variables in $\mathbf{head}(q) \setminus Y$. Since Y, Z are disjoint, $\pi_{Y \cup Z}(T) \subseteq \pi_Y(T) \times \pi_Z(T)$. Now notice that $\pi_Y(T)$ is at most the size of T. Also, $\pi_Z(T)$ is bounded by the size of the output (but only because we took care of dangling tuples by applying the full reducer). □

Example 4.12. *How does the acyclic algorithm work for the following query?*

$$q(x,t) : -R(x,y,z), S(y,v), T(y,z,u), U(z,u,w), V(u,w,t).$$

Run the GYO algorithm, find the join tree and explain what is the query plan produced by the polynomial-time algorithm.

References

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

[Y81] M. YANNAKAKIS, "Algorithms for acyclic database schemes," *VLDB 1981*.