CS 784: Foundations of Data Management **Spring 2019**

Lecture 5: Query Decompositions

Instructor: Paris Koutris

In the last lecture we showed that for the class of acyclic Conjunctive Queries, evaluation is in polynomial time (combined complexity). It is logical to ask whether acyclic queries is the only class of queries that can be evaluated in polynomial time. The answer to this question is yes, and we will see here how we can extend the concept of acyclic queries to other queries.

Let's start with an example!

Example 5.1. *Consider the boolean cycle query, which asks whether there is cycle of length k in the directed graph represented by the edge relation R.*

$$C^k() :- R_1(x_1, x_2), R_2(x_2, x_3), \dots, R_k(x_k, x_1)$$

Assume that for every i, $|R_i| = N$. As we showed in the last lecture, this query is not acyclic and does not admit a join tree. However, we can apply a similar idea to evaluate this query in polynomial time. We first perform the natural join $R(x_1, x_2) \bowtie R(x_2, x_3)$, then project on x_1, x_3, then join with $R(x_3, x_4)$, project on x_1, x_4, and so on. The key difference from the algorithm that evaluates the path query P^k is that we also keep x_1 around. Observe that at every point the size of the intermediate result is at most N^2. Hence, we can implement the algorithm with running time time $O(kN^2)$, and the running time is again polynomial in the size of the query and the input!

The characterization of CQs that we can compute in polynomial time is based on the idea of "decomposing" the hypergraph of the query. We start by presenting one such decomposition, called the *query decomposition*. We can think of the query decomposition as a generalization of the construct of a join forest for acyclic queries.

5.1 Generalized Hypertree Decompositions

Definition 5.2. *A* generalized hypertree decomposition (GHD) *of a CQ q is a pair (T, λ), where $T = (V, E)$ is a tree and χ is a labeling function which associates to each vertex $v \in V$ a subset of variables of q, $\chi(v)$, such that the following conditions are satisfied:*

1. *For each atom A in q, there exists a node $v \in V$ such that $\chi(v)$ contains all variables of A.*
2. *For each variable $x \in \mathbf{vars}(q)$, the set of nodes $\{v \mid x \in \chi(v)\}$ forms a connected subtree.*

The set $\chi(v)$ is also called a *bag*. The second condition of the decomposition generalizes the "connectedness" condition for join trees and is equivalent to requiring that for every two vertices v_1, v_2

of the tree, the variables in $\chi(v_1) \cap \chi(v_2)$ will appear in all the vertices along the unique path that connects v_1 to v_2. When each bag of a GHD consists of the attributes of a single relation, it is easy to see that the GHD is exactly a join tree!

Given a node v in the query decomposition, we define its *width* $w(v)$ as the minimum number of atoms necessary to cover all variables in $\chi(v)$.

Definition 5.3 (Generalized Hypertree Width)**.** *Let* (T, λ) *be a GHD of* q*. The* generalized hypertree width (ghw) *of the decomposition is defined as* $\max_{v \in V} w(v)$*. The* ghw *of a query* q *is the minimum ghw over all possible GHDs.*

Example 5.4. *Consider the cycle query* C_k *of the initial example. We will construct three different decompositions for this query.*

The first decomposition is a tree with a single node v*, and* $\chi(v) = \{x_1, x_2, \dots, x_k\}$*. It is trivial to see that this decomposition satisfies both conditions, and has ghw* $k - 1$*.*

The second decomposition is a tree with two nodes v_1, v_2*, and an edge* $\{v_1, v_2\}$*. The labeling function is* $\chi(v_1) = \{x_1, x_2, \dots, x_{k-1}\}, \chi(v_2) = \{x_{k-1}, x_k, x_1\}$*. This is a correct decomposition because* v_1, v_2 *share 2 variables and indeed there are connected. The ghw in this case is* $\max\{k - 2, 2\} = k - 2$*.*

The third decomposition has k vertices $v_1, \dots, v_k$ *that form a path, where* $\chi(v_i) = \{x_1, x_i, x_{i+1}\}$*. It is easy to see that the first 2 conditions are satisfied. For the third condition, notice that every variable apart from* x_1 *appears in consecutive vertices in the path and that satisfies the connectedness property. Variable* x_1 *appears in every vertex, so it trivially satisfies the property. The ghw here is* 2*.*

Can we construct a GHD for C_k with ghw less than 2? The lemma below tells us that it is not possible to do, which shows that the *ghw* of C_k is 2.

Lemma 5.5. *A CQ* q *is acyclic if and only if* $ghw(q) = 1$*.*

GHDs can be interpreted as query plans for joins. Given a GHD, we can first compute one intermediate relation per bag/node, by computing the join restricted to the variables of the bag. The intermediate relations can then be joined using Yannakakis algorithm.

Algorithm 1: Evaluation of CQ with ghw k

```
Input: Conjunctive Query $q$ of $ghw = k$, instance $I$
Output: $q(I)$
construct a GHD $(T, \lambda)$ for $q$ ;
for vertex $v$ in $T$ do
    $q_v \leftarrow$ obtained from $q$ by keeping only the variables in $\lambda(v)$ ;
    compute $q_v(I)$ ;
end
Run Yannakakis using $T$ as the join tree and $q_v(I)$ as the input relations of each node ;
```

In the above algorithm, the key observation is that we need $O(|I|^k)$ time to compute the intermediate result for each node in the decomposition. Moreover, the size of each intermediate result is bounded by $O(|I|^k)$. Hence, applying Yannakakis algorithm leads to the following results:

Theorem 5.6 ([CR00])**.** *Let q be a boolean CQ with $ghw(q) = k$. Given a GHD of q of width k, we can compute q with running time $O(|I|^k)$, where I is the input database.*

Theorem 5.7 ([CR00])**.** *Let q be a full CQ with $ghw(q) = k$. Given a GHD of q of width k, we can compute q with running time $O(|I|^k + OUT)$.*

This implies that we can efficiently evaluate a conjunctive query with bounded ghw, where *bounded* means that the width is bounded by some constant. Unlike the case of acyclic CQs, no efficient method for checking whether the ghw is bounded is known. In fact, deciding whether a CQ has a GHD with bounded width is NP-complete.

5.2 Other Notions of Width

Apart from GHDs, several other notions of decompositions have been presented in the literature.

- **Tree-width**: This decomposition is a *tree decomposition* performed on the Gaifman graph of the query q. The Gaifman graph of a query q has the variables as vertices, and an edge occurs between x, y if and only if x, y appear in the same atom. If a query has bounded tree-width, it can be evaluated in polynomial time (combined complexity) [CR00]. Checking whether a graph has a bounded tree decomposition can be done in polynomial time!
- **Hypertree-Width**: This is a decomposition of the hypergraph of q. Again, bounded hypertree-width implies evaluation in polynomial time [GSL02]. The hypertree-width is always larger than the generalized hypertree width, but we can always find a bounded hypertree decomposition, if one exists, in polynomial time.
- **Fractional Hypertree-Width**: This is a generalization of the hypertree-width that encompasses an even bigger class of CQs that can be evaluated in polynomial time. We will talk more about these in subsequent lectures!

All three notions are essentially equivalent when the schema of the database is constant (so the arity of each atom is bounded by a constant). It also turns out that bounded query-width (or tree-width, or hypertree-width) is necessary to achieve polynomial time evaluation [GTS01]. When the schema is not fixed, then the fractional hypertree width leads to polynomial time evaluation of more general classes of queries.

We should note here that decompositions and the evaluation of CQs is very tightly connected with the area of Constraint Satisfaction Problems (CSPs) in artificial intelligence; see the survey [GGS] for more details on this connection and the applications of hypertree decompositions.

References

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

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

[CR00] C. CHEKURI and A. RAJARAMAN, "Conjunctive query containment revisited," *Theoretical Computer Science 239 (2000)* .

[GLS02] G. GOTTLOB, N. LEONE and F. SCARCELLO, "Hypertree Decompositions and Tractable Queries," *Journal of Computer and System Sciences 64 (2002)* .

[GTS01] M. GROHE, T. SCHWENTICK and L. SEGOUFIN, "When is the evaluation of conjunctive queries tractable ?," *STOC 2001* .

[GGS] G. GOTTLOB, G. GRECO and F. SCARCELLO, "Treewidth and Hypertree Width".