In the last lecture we showed that for the class of acyclic conjunctive queries, query evaluation is in polynomial time. It is logical to ask if 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 $R$.

\[ C^k() : \overline{R(x_1, x_2)}, R(x_2, x_3), \ldots, R(x_k, x_1) \]

Assume that $|R| = n$. 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 join $R(x_1, x_2), R(x_2, x_3)$, 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 $P^k$ 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 $O(kn^2)$. Thus, the running time is again polynomial!

The characterization of CQs that we can compute in polynomial time is based on the idea of "decomposing" 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 Bounded Query-Width

**Definition 5.2 (Query Decomposition).** A query decomposition of a CQ $q$ is a pair $(T, \lambda)$, where $T = \langle V, E \rangle$ is a tree and $\lambda$ is a labeling function which associates to each vertex a subset of atoms and variables of $q$, such that the following conditions are satisfied:

1. For each atom $A$ in $q$, there exists a node $v \in V$ such that $A \in \lambda(v)$.
2. For each atom $A$ in $q$, the set of nodes that contain $A$ induces a connected subtree of $T$.
3. For each variable $x$ in $q$, the set of nodes that contain $x$ (either as a variable, or in an atom in the label), induces a connected subtree of $T$. 

5-1
The width of the query decomposition is \( \max_{v \in V} |\lambda(v)| \). The query width of a query \( q \) is the minimum width over all possible query decompositions.

**Example 5.3.** Consider the cycle query \( C_k \) of the initial example. Construct a query decomposition with width equal to 2.

**Lemma 5.4.** A CQ \( q \) is acyclic if and only if its query width is equal to 1.

Using the same logic as the Yannakakis algorithm [Y81] for acyclic queries, we can now show that a boolean CQ that admits a query decomposition of width \( k \) can be evaluated in time \( O(n^k) \), where \( n \) is the input size.

**Theorem 5.5** ([CR00]). Let \( q \) be a boolean CQ with query-width \( k \). Given the query decomposition of \( q \), we can compute \( q \) with running time \( O(|I|^k) \), where \( I \) is the input database.

This implies that we can efficiently evaluate a conjunctive query with bounded query-width, where bounded means that the width is bounded by some constant. Unfortunately, unlike for acyclic CQs, no efficient method for checking bounded query-width is known. In fact, deciding whether a CQ has a bounded-width query decomposition is \( \text{NP} \)-complete [GSL02].

### 5.2 Discussion

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

- **Bounded 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 \( \mathbb{P} \) [CR00]. Checking whether a graph has a bounded tree decomposition can be done in polynomial time!

- **Bounded 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 smaller than the query width, but we can always find a bounded hypertree decomposition, if one exists, in polynomial time.

- **Bounded 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.

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 hypertree-based approach 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


