We have learned the concepts of data complexity and combined complexity. We have seen that CQs can be evaluated in polynomial time under data complexity:
- And that the degree of the polynomial “necessarily” depends on the query (W[1]-hardness)
- We have seen that, under combined complexity:
  - Boolean CQ evaluation is NP-complete
  - CQs cannot be evaluated in polynomial total time, unless P = NP

In this lecture, we focus on combined complexity.
- We have seen an example of a fragment of CQs that can be evaluated in polynomial total time:
  - Namely, n-length paths
- We will learn more a general fragment of tractable CQs:
  - Acyclic CQs
  - More generally, CQs of a bounded hypertree width
- In addition, we will learn size bounds on (projection-free) joins, and a matching (worst-case optimal) algorithm
Recalling Conjunctive Queries

- Recall that a Conjunctive Query (CQ) has the form
  \[ Q(x) := \psi_1(x, y), \ldots, \psi_m(x, y) \]
  where each \( \psi_j \) is an atomic formula, \( x \) and \( y \) are disjoint sequences of unique variables.

  - An atomic formula has the form \( R(\tau_1, \ldots, \tau_k) \) where \( R \) is a \( k \)-ary relation symbol and each \( \tau_i \) is either a variable (in \( x \) or \( y \)) or a constant term.
  - \( Q(x) \) is the head, \( \psi_1(x, y), \ldots, \psi_m(x, y) \) is the body, and each \( \psi_j(x, y) \) is a body atom.
  - We require every variable in the head to occur at least once in the body.

Reductions

- To understand the difficulty of joins, we will recall the proof of NP-hardness, and see a new one.

  - In the first reduction (that we have seen already), we generated a CQ with a single binary relation, repeating many times.
  - In the second reduction, we generate a CQ with many ternary relation symbols, but none of them appears more than once in \( Q \); in addition, each relation has precisely seven tuples.

    - A CQ without repeated relation symbols is called non-repeating or self-join free.

Problem Def. (Clique)

Given a graph \( G = (V, E) \) and a number \( k \), determine whether \( G \) contains a clique of size \( k \), that is, a subset \( U \subseteq V \) such that \( |U| = k \) and every two nodes in \( U \) are neighbours.

Problem Def. (3-SAT)

Given a propositional formula \( \psi = \varphi_1 \land \cdots \land \varphi_m \) over the variables \( x_1, \ldots, x_n \), where each \( \varphi_i \) is a disjunction of three atomic formulas (each has the form \( x_i \text{ or } \neg x_i \)), determine whether \( \psi \) is satisfiable.
From CQs to Joins

Reduction

- Given $\psi = \varphi_1 \land \cdots \land \varphi_m$, we construct:
  - A relation symbol $R_i$ for each $\varphi_i$.
  - An atomic formula $\phi_i = R_i(x, y, z)$ where $x$, $y$, and $z$ are the variables that appear in $\varphi_i$.
  - $Q(x_1, \ldots, x_n) = \phi_1, \ldots, \phi_m$.
  - The instance $I$ has in the relation $R_i$ all 7 tuples $(b_1, b_2, b_3) \in \{0, 1\}^3$ that satisfy $\varphi_i$.
- That's it!

Example: $\psi = (x \lor y \lor z) \land (\neg x \lor y \lor w) \land (x \lor z \lor \neg w)$

<table>
<thead>
<tr>
<th>$I$</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Problem Def. (3-Coloring)

Given a (directed) graph $G = (V, E)$, determine whether we can assign a color from $\{r, g, b\}$ to each node, so that no two neighbors get the same color.

From CQs to Joins

- It is sometimes more comfortable to work with RA joins (and projection) instead of CQs.
- Given a CQ $Q$ and an instance $I$ over a schema $S$, we can easily construct a schema $T$, an RA expression $\alpha$ over $T$ and an instance $J$ over $T$ such that:
  - $\alpha$ has the form $\pi_{A_1, \ldots, A_m}(T_1 \bowtie \cdots \bowtie T_m)$ where the $T_i$ are distinct relation symbols.
  - $\alpha(J)$ and $Q(I)$ are “the same”.
  - That is, there is a straightforward translation between the two.
- For example, how would you translate the following CQ?
  $$Q(x, y) := R(x, y, Avia), R(y, z, x), S(x, x)$$

Translation

- Let $Q(x) := \varphi_1(x, y), \ldots, \varphi_m(x, y)$ and $I$ be over $S$.
  - Each variable becomes an attribute.
  - Each body atom $\varphi_i$ becomes a unique relation schema $T_i$ with the attributes (variables) that appear in $\varphi_i$ (in any order).
  - Each head variable becomes a projection attribute.
  - In $J$, the relation $T_i$ is obtained by evaluating $\varphi_i$ over $I$ as if $\varphi_i$ is a CQ with all variables in the head.
- Example: $Q(x, y) := R(x, y, Avia), R(y, z, x), S(x, x)$
  $$\Rightarrow \pi_{x,y}(T_1(x, y) \bowtie T_2(x, y, z) \bowtie T_3(x))$$
In the remainder of this lecture, a **CQ expression** is an RA expression of the form

\[ \pi_A(R_1 \bowtie \cdots \bowtie R_k) \]

- Every \( R_i \) is a distinct relation symbol (of any arity)
- \( A \) is a sequence of attributes among the \( A_i \)s
- If projection \( \pi \) is redundant, it may be omitted

A **hypergraph** is a pair \((V, H)\), where \( V \) is a finite set of nodes, and \( H \) is a set of subsets of \( V \), called hyperedges (and sometimes just edges)

- If \( H \) is a hypergraph, then we denote by
  - nodes\((H)\) the set of nodes of \( H \)
  - edges\((H)\) the set of hyperedges of \( H \)
- Let \( \alpha = \pi_A(R_1 \bowtie \cdots \bowtie R_k) \) be a CQ expression
- The **hypergraph of \( \alpha \)**, denoted \( H_\alpha \), has:
  - The attributes in \( \alpha \) as the set of nodes
  - A hyperedge \( e_i \) for each \( R_i \), containing the attributes of \( R_i \)

A **join tree** of a hypergraph \( H \) is a tree \( T \) with the following properties

- The nodes of \( T \) are the hyperedges of \( H \)
  - In notation, nodes\((T) = \text{edges}(H)\)
- For every \( v \in \text{nodes}(H) \), the nodes of \( T \) that contain \( v \) form a connected subtree of \( T \)

Example:

- An ear of a hypergraph \( H \) is a hyperedge \( e \) of \( H \) such that
  - \( e \) is disjoint from all other hyperedges or
  - there exists another hyperedge \( e' \) where \( e \setminus e' \) is disjoint from all other hyperedges
- An **ear removal** on \( H \) is the operation of obtaining a new hypergraph \( H' \) by removing an ear \( e \) of \( H \)
  - nodes\((H') = \text{nodes}(H) \) and edges\((H') = \text{edges}(H) \setminus \{e\}\)

Example:
Acyclic Hypergraphs

**Proposition**
Let \( H \) be a hypergraph. The following are equivalent:
- \( H \) has a join tree.
- By repeatedly applying ear removal (in any order), one can eliminate all the hyperedges of \( H \).
If \( H \) satisfies the above conditions, then \( H \) is said to be **acyclic**.

**Comments**
- You will prove the proposition in a home assignment.
- In particular, you will show how to build a join tree for a given \( H \) via ear removal.
  - Efficiently!
  - (This will be used later in this lecture)
- When \( H \) is a graph (i.e., every hyperedge has exactly two nodes), acyclicity is the usual notion of graph acyclicity (forest).
  - In other words, **graph acyclicity** and **hypergraph acyclicity** are the same on graphs.

**Acyclic CQs**
- A CQ expression \( \alpha \) is acyclic if its associated hypergraph \( H_\alpha \) is acyclic.
- **Which of the following is acyclic?**
  - \[ \bigvee_{1 \leq i < j \leq n} R_{ij}(x_i, x_j) \]
  - \[ \bigvee_{1 \leq i < j \leq n} R_{ij}(x_i, x_j) \land S(x_1, \ldots, x_n) \]
- **Which of the above can be solved in polynomial total time?**

**Algorithms for Acyclic Database Schemes [Yan81]**
- In this part we describe the algorithm of Mihalis Yannakakis [Yan81] for computing acyclic CQs.
- The algorithm terminates in polynomial total time.
  - Recall: polynomial time in the combined size of the input and the output.

**Input:** CQ expression \( \alpha = \pi_A(R_1 \times \cdots \times R_k) \), instance \( I \)
- Compute a join tree \( T \) for \( H_\alpha \)
- Apply a full reduction to \( I \) according to \( T \)
  - More specifically, replace source relations with **semijoins**
- Compute \( \alpha(I) \) in **leaf-to-root** order according to \( T \), projecting on only relevant variables
  - And eliminating every redundant/irrelevant variable.
Computing a Join Tree

- This can be done (in polynomial time) by the ear-removal procedure.
- We will view the join tree as directed and ordered by:
  - Selecting an arbitrary root that all nodes are reachable from.
  - This action determines all directions.
  - Selecting an arbitrary order among every set of siblings.
- In the next slides, denote this (directed & ordered) tree by $T$.

Notation

- For each node $v$ of $T$, let:
  - $R_v$ be the relation symbol that corresponds to $v$.
  - $r_v$ be the relation of $I$ over $R_v$.
- Example: $\pi_{x,y}(R(x,y,z) \times S(x,u) \times T(y,z,w))$.

$$
\begin{array}{c}
  v \\
  y \\
  z \\
  w \\
  u \\
  x \\
  \end{array}

R_v = R \quad R_v' = T \quad R_v'' = S
$$

Intuition on Full Reduction (1)

- The left semijoin of two relations $r$ and $s$, denoted $r \bowtie s$, is the relation that is obtained from $r$ by selecting only the tuples that have a matching tuple in (i.e., are joinable with) $s$.
- In RA:
  $$
  r \bowtie s \overset{def}{=} \pi_A(r \bowtie s)
  $$
  where $A$ is the attribute sequence of $r$.
- For example, what is $r \bowtie s$ if:
  - $r$ and $s$ have the same set of attributes?
  - $r$ and $s$ have disjoint sets of attributes?
Applying a Full Reduction

- Procedure called \textit{Inside-Out}, using two passes
  - \textbf{Leaf-to-root (inside)}:
    
    1. \textbf{for} all nodes \( v \) of \( T \) in leaf-to-root order \textbf{do}
    2. \textbf{if} \( v \) is not the root of \( T \) \textbf{then}
    3. \( r_p \coloneqq r_p \bowtie r_v \), where \( p \) is the parent of \( v \)
  - \textbf{Root-to-leaf (out)}:
    
    1. \textbf{for} all nodes \( v \) of \( T \) in root-to-leaf order \textbf{do}
    2. \textbf{for} all children \( c \) of \( v \) \textbf{do}
    3. \( r_c \coloneqq r_c \bowtie r_v \)

Leaf-to-Root Join

- For each node \( v \) of \( T \), let:
  - \( T_v \) be the subtree of \( T \) rooted at \( v \)
  - \( O_v \) be the set of projected attributes that appear in \( T_v \)
  - \( P_v \) be the set of attributes shared by \( v \) and its parent (empty for the root)

We apply the join as follows:

1. \textbf{for} all nodes \( v \) of \( T \) in leaf-to-root order \textbf{do}
2. \textbf{if} \( v \) is a leaf \textbf{then}
3. \( \text{result}(v) \coloneqq r_v \)
4. \textbf{else}
5. \textbf{let} \( c_1, \ldots, c_k \) be the children of \( v \)
6. \( \text{result}(v) \coloneqq \pi_{P_v} \big( r_v \bowtie \text{result}(c_1) \bowtie \cdots \bowtie \text{result}(c_k) \big) \)
7. The result is \( \text{result}(\text{root}(T)) \)

Intuition (1)

![Diagram of a tree structure with tuples](image1.png)

Intuition (2)

![Diagram of a hypertree decomposition with tuples](image2.png)

Correctness and Efficiency (Sketch)

- Proof idea:
  - Every tuple that is deleted during the full reduction does not contribute to the overall result of the join; \textit{why so?}
  - On the other hand, after the full reduction, there are no "hanging tuples" in \( r_v \) (every tuple participates in the join)
  - Similarly, in the evaluation, there are no hanging tuples in \( \text{result}(v) \) (every tuple can be extended to a join tuple)
  - Consequently:
    - We compute the correct result
    - The size of each \( \text{result}(v) \) is polynomial in the size of the final output

Table of Contents

1. \textbf{Introduction}
2. \textbf{Preliminaries}
3. \textbf{Acyclic Joins}
4. \textbf{Algorithm for Acyclic Joins (Yannakakis)}
5. \textbf{Joins with Hypertree Decompositions}
6. \textbf{Size Bounds and Worst-Case Optimality}

References
Let $G$ be a graph

A **Tree Decomposition** $(TD)$ of $G$ is a pair $(T, \chi)$ with the following properties:

- $T$ is a tree
- $\chi$ is a function that maps every node $t$ of $T$ to a subset (called **bag**) of $\text{nodes}(G)$, so that:
  - For every edge $e \in \text{edges}(G)$ there is a node $t$ of $T$ such that $e \subseteq \chi(t)$
  - Every node $v$ of $G$ occurs in a connected subtree of $T$; that is, the set $\{ t \in \text{nodes}(T) \mid v \in \chi(t) \}$ induces a connected subtree of $T$
- The **width** of $T$ is $\max \{|\chi(v)| \mid v \in \text{nodes}(T)\} - 1$
- The **treewidth** of $G$ is the minimal width over all TDs of $G$

Definitions of this part taken from Gottlob et al. [GGM'05]

Let $H$ be a hypergraph

A **Tree Decomposition** $(TD)$ of $H$ is a pair $(T, \chi)$ with the following properties:

- $T$ is a tree
- $\chi$ is a function that maps every node $t$ of $T$ to a subset (called **bag**) of $\text{nodes}(H)$, so that:
  - For every hyperedge $e \in \text{edges}(H)$ there is a node $t$ of $T$ such that $e \subseteq \chi(t)$
  - Every node $v$ of $H$ occurs in a connected subtree of $T$; that is, the set $\{ t \in \text{nodes}(T) \mid v \in \chi(t) \}$ induces a connected subtree of $T$
- Note: if $(T, \chi)$ is a TD of $H$, then $T$ is a **join tree** over the bags

### Intuition (3)

Tree Decomposition of a Graph

- Let $G$ be a graph
- A **Tree Decomposition** $(TD)$ of $G$ is a pair $(T, \chi)$ with the following properties:
  - $T$ is a tree
  - $\chi$ is a function that maps every node $t$ of $T$ to a subset (called **bag**) of $\text{nodes}(G)$, so that:
    - For every edge $e \in \text{edges}(G)$ there is a node $t$ of $T$ such that $e \subseteq \chi(t)$
    - Every node $v$ of $G$ occurs in a connected subtree of $T$; that is, the set $\{ t \in \text{nodes}(T) \mid v \in \chi(t) \}$ induces a connected subtree of $T$
  - The width of $T$ is $\max \{|\chi(v)| \mid v \in \text{nodes}(T)\} - 1$
  - The treewidth of $G$ is the minimal width over all TDs of $G$

### Example (1)

**Tree Decomposition of a Hypergraph**

- Definitions of this part taken from Gottlob et al. [GGM'05]
- Let $H$ be a hypergraph
- A **Tree Decomposition** $(TD)$ of $H$ is a pair $(T, \chi)$ with the following properties:
  - $T$ is a tree
  - $\chi$ is a function that maps every node $t$ of $T$ to a subset (called **bag**) of $\text{nodes}(H)$, so that:
    - For every hyperedge $e \in \text{edges}(H)$ there is a node $t$ of $T$ such that $e \subseteq \chi(t)$
    - Every node $v$ of $H$ occurs in a connected subtree of $T$; that is, the set $\{ t \in \text{nodes}(T) \mid v \in \chi(t) \}$ induces a connected subtree of $T$
  - Note: if $(T, \chi)$ is a TD of $H$, then $T$ is a **join tree** over the bags

### Example (2)

**Examples**

- Let $G$ be a graph
- A **Tree Decomposition** $(TD)$ of $G$ is a pair $(T, \chi)$ with the following properties:
  - $T$ is a tree
  - $\chi$ is a function that maps every node $t$ of $T$ to a subset (called **bag**) of $\text{nodes}(G)$, so that:
    - For every edge $e \in \text{edges}(G)$ there is a node $t$ of $T$ such that $e \subseteq \chi(t)$
    - Every node $v$ of $G$ occurs in a connected subtree of $T$; that is, the set $\{ t \in \text{nodes}(T) \mid v \in \chi(t) \}$ induces a connected subtree of $T$
  - The width of $T$ is $\max \{|\chi(v)| \mid v \in \text{nodes}(T)\} - 1$
  - The treewidth of $G$ is the minimal width over all TDs of $G$
Every hypergraph has a TD!

In what sense is a TD “good”?

Depends on the context!

In our case, we would like to be able to efficiently compute the part of the join that corresponds to each bag.

This could be achieved if each bag could be covered by a small number of relations.

Just intuition... Later we show how exactly that helps to get complexity bounds.

Let $\mathcal{H}$ be a hypergraph

A Generalized Hypertree Decomposition (GHD) of $\mathcal{H}$ is a triple $(T, \chi, \lambda)$ such that:

- $(T, \chi)$ is a tree decomposition of $\mathcal{H}$
- $\lambda$ is a function that maps every node $t$ of $T$ to a subset of edges $e$ that covers $\chi(t)$; that is, $\chi(t) \subseteq \cup \lambda(t)$

- $\cup \lambda(t)$ means $\bigcup_{e \in \lambda(t)} e$

The width of a GHD $(T, \chi, \lambda)$ is the maximal number of hyperedges needed for covering a node; that is, $\max \{|\lambda(t)| | t \in \text{nodes}(T)|}$

- The generalized hypertree width (gwh) of a hypergraph $\mathcal{H}$ is the minimum of the widths of all GHDs of $\mathcal{H}$

- The gwh of a CQ expression $\alpha$ is the gwh of $\mathcal{H}_\alpha$

- Claim (easy to prove): $\alpha$ (or $\mathcal{H}$) is acyclic if and only if its gwh is 1

We now show how a small (bounded) gwh can be used for efficiently computing a join.

For each hyperedge $e$ of $\mathcal{H}_\alpha$, let:

- $R_e$ be the relation symbol that corresponds to $e$
- $r_e$ be the relation of $I$ over $R_e$

Next, for each relation $r_i$ find a node $t$ such that $\chi(t)$ contains all the attributes of $R_i$ and set:

$r(t) := r(t) \Box r_i$

That is, delete from $r(t)$ every tuple that cannot be joined with any tuple from $r_i$.

Let $\alpha$ be a CQ expression, and let $(T, \chi, \lambda)$ be a GHD of $\mathcal{H}_\alpha$.

Given an instance $I$, we can compute $\alpha(I)$ as follows:

For each node $t$ of $T$ compute the relation:

$r(t) := \pi_{\chi(t)}(\Box_{e \in \lambda(t)} r_e)$

CQ Evaluation with a GHD (1)
CQ Evaluation with a GHD (2)

- Now we have the following:
  - \( M_{r_1}^{m_1} r_1 = M_{\text{hypertree}(T)} r(f) \)
  - \( \pi_A(M_{r_1}^{m_1} r_1) = \pi_A(M_{\text{hypertree}(T)} r(f)) \)
  - \( \pi_A(M_{\text{hypertree}(T)} r(f)) \) is an acyclic CQ expression
- Apply Yannakakis's to compute \( \pi_A(M_{\text{hypertree}(T)} r(f)) \)
- That's it!

Finding a GHD

- It is NP-complete to decide whether a given hypergraph \( H \) has a ghw at most \( k \) for any constant \( k \geq 3 \) [GMS09]
- Nevertheless, there is a restricted variant of a GHD, called hypertree decomposition, which can be found in polynomial time for every fixed \( k \)
  - Basically, it is a GHD with an additional requirement
- We do not discuss hypertree decompositions here, but still:
- We define the Hypertree Width of a hypergraph \( H \) as the minimal width over all hypertree decompositions of \( H \)
- Fact: A hypergraph is acyclic if and only if its hypertree width (and ghw) is 1

Resulting Theorem

**THEOREM**

For every constant \( k \), CQ expressions with hypertree width at most \( k \) can be evaluated in polynomial total time.

*In fact, polynomial delay [KS06]*

What about Bounded ghw?

- We know that it is intractable to construct, for a given CQ expression, a GHD of width at most \( k \) for all constants \( k \geq 3 \) [GMS09]
- So, the strategy discussed so far (materializing bags) will not work for showing the tractability of CQs with a bounded GHD
- Quite remarkably, Chen and Dalmau [CD05] showed that bounded ghw allows to evaluate Boolean CQs in polynomial time
  - Even if we cannot construct a corresponding CQ
- Again, this gives polynomial delay [KS06]

Stronger Theorem [CD05]

**THEOREM**

For every constant \( k \), CQ expressions with a generalized hypertree width at most \( k \) can be evaluated with polynomial delay.
In this part, we focus on a projection-free join query:

\[ Q \overset{\text{def}}{=} R_1 \times \cdots \times R_k \]

- For \( i = 1, \ldots, k \), denote by \( \text{Att}(R_i) \) the attribute set of \( R_i \).
- Denote by \( \text{Att}(Q) \) the union \( \bigcup_{i=1}^{k} \text{Att}(R_i) \).
- A database \( D \) consists of the relation \( r_i \) over each \( R_i \).
- We denote by \( |r_i| \) the number of tuples in \( r_i \).

How many answers can be for the following queries, in terms of \( |r_1|, \ldots, |r_k| \)?

- \( R_i(A) \times R_i(B) \times R_i(C) \)
- \( R_i(A) \times R_i(B) \times R_i(C) \)
- \( R_i(A, B) \times R_i(B, C) \times R_i(C, A) \)
- \( R_i(A, B) \times R_i(B, C) \times R_i(C, A) \times R_i(A, B) \)
- \( R_i(A, B) \times R_i(B, C) \times R_i(C, A) \times R_i(A, B, C) \)

Recall that each \( A_n \) is a sequence \( (a_1, \ldots, a_k) \in \{0,1\}^k \) such that each \( A \in \text{Att} \) occurs in at least one \( R_i \) with \( a_i = 1 \)

In the previous slide we established the following:

\[ |Q(D)| \leq \prod_{i=1}^{k} |r_i|^{w_i} \]

This bound, however, is not tight; we get tightness via the fractional edge cover.
Examples

- What is the fractional edge cover of the following join?

  \[ R(A, B) \Join S(B, C) \Join T(C, A) \]

- More generally, the Loomis Whitney join \( \mathcal{Q}_{k}^{\text{LW}} \) is the following:

  \[ \mathcal{Q}_{k}^{\text{LW}} = R_{1}(x_{1}, \ldots, x_{k}) \Join R_{2}(x_{1}, x_{3}, \ldots, x_{k}) \Join \cdots \Join R_{k}(x_{1}, x_{3}, \ldots, x_{k-1}) \]

  What is the fractional edge cover of \( \mathcal{Q}_{k}^{\text{LW}} \)?

Worst-Case Optimality

- An algorithm for computing \( \mathcal{Q} \) is worst-case optimal if its running time is \( O(f(|r_{1}|, \ldots, |r_{n}|)) \) where \( f(r_{1}, \ldots, r_{n}) \) is the maximal \( |Q(D)| \) over all databases \( D \) with \( |r_{i}| = n_{i} \)

- Starting with Ngo et al. [NPRR12], in recent years several worst-case optimal join algorithms have been devised [Vel14, KNRR15, KEK17]

- In particular, the running time of these algorithms does not exceed the AGM bound

  (The algorithms themselves are beyond the scope of the course)

References


References


End of lecture 4
Computing Joins