Principles of Managing Uncertain Data

Lecture 4: Computing Joins
Table of Contents

1 Introduction

2 Preliminaries

3 Acyclic Joins

4 Algorithm for Acyclic Joins (Yannakakis)

5 Joins with Hypertree Decompositions

6 Size Bounds and Worst-Case Optimality
We have learned the concepts of data complexity and combined complexity.
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 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.
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**
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
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 about a general fragment of tractable CQs.
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
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 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.
Table of Contents

1 Introduction
2 Preliminaries
3 Acyclic Joins
4 Algorithm for Acyclic Joins (Yannakakis)
5 Joins with Hypertree Decompositions
6 Size Bounds and Worst-Case Optimality
Recalling Conjunctive Queries

- Recall that a Conjunctive Query (CQ) has the form

\[ Q(x) := \varphi_1(x, y), \ldots, \varphi_m(x, y) \]

where each \( \varphi_i \) is an atomic formula, \( x \) and \( y \) are disjoint sequences of unique variables.
Recalling Conjunctive Queries

- Recall that a Conjunctive Query (CQ) has the form

\[ Q(x) :- \varphi_1(x, y), \ldots, \varphi_m(x, y) \]

where each \( \varphi_i \) is an atomic formula, \( x \) and \( y \) are disjoint sequences of unique variables

  - An \textit{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
Recalling Conjunctive Queries

- Recall that a Conjunctive Query (CQ) has the form

\[ Q(x) \leftarrow \varphi_1(x, y), \ldots, \varphi_m(x, y) \]

where each \( \varphi_i \) 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**, \( \varphi_1(x, y), \ldots, \varphi_m(x, y) \) is the **body**, and each \( \varphi_i(x, y) \) is a **body atom**

- We require every variable in the head to occur at least once in the body
Result of a CQ

- Let \( Q(x) :- \varphi_1(x, y), \ldots, \varphi_m(x, y) \) and \( I \) be a CQ and an instance, respectively (over the same signature)
Result of a CQ

- Let $Q(x) :- \varphi_1(x, y), \ldots, \varphi_m(x, y)$ and $I$ be a CQ and an instance, respectively (over the same signature).
- A *homomorphism* from $Q$ to $I$ is a function $\mu$ that maps every variable of $Q$ to a constant, such that $\mu(\varphi_i(x, y))$ is a fact of $I$ for every $i = 1, \ldots, m$.
  - $\mu(\varphi_i(x, y))$ is the fact that is obtained by replacing every variable $z$ with the constant $\mu(z)$. 
Result of a CQ

- Let $Q(x) := \varphi_1(x,y), \ldots, \varphi_m(x,y)$ and $I$ be a CQ and an instance, respectively (over the same signature).
- A *homomorphism* from $Q$ to $I$ is a function $\mu$ that maps every variable of $Q$ to a constant, such that $\mu(\varphi_i(x,y))$ is a fact of $I$ for every $i = 1, \ldots, m$.
  - $\mu(\varphi_i(x,y))$ is the fact that is obtained by replacing every variable $z$ with the constant $\mu(z)$.
- If $\mu$ is a homomorphism from $Q$ to $I$, then $\mu|_x$ is the restriction of $\mu$ to the variables of $x$. 
Result of a CQ

- Let $Q(x) := \varphi_1(x, y), \ldots, \varphi_m(x, y)$ and $I$ be a CQ and an instance, respectively (over the same signature).
- A homomorphism from $Q$ to $I$ is a function $\mu$ that maps every variable of $Q$ to a constant, such that $\mu(\varphi_i(x, y))$ is a fact of $I$ for every $i = 1, \ldots, m$.
  - $\mu(\varphi_i(x, y))$ is the fact that is obtained by replacing every variable $z$ with the constant $\mu(z)$.
- If $\mu$ is a homomorphism from $Q$ to $I$, then $\mu|_x$ is the restriction of $\mu$ to the variables of $x$.
- The result of evaluating $Q$ over $I$, denoted $Q(I)$, is the set
  \[
  \{ \mu|_x \mid \mu \text{ is a homomorphism from } Q \text{ to } I \}.
  \]
To understand the difficulty of joins, we will recall the proof of NP-hardness, and see a new one.
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 \textit{single binary relation}, repeating many times.
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*.
Reduction 1: from Clique

**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 $U$ of $V$ such that $|U| = k$ and every two nodes in $U$ are neighbours.
**Reduction**

- Given $G = (V, E)$ with $V = \{1, \ldots, n\}$, and $k$, construct:
  - $S = \{R_E/2\}$
  - $I_G = \{R_E(i, j) \mid \{i, j\} \in E \text{ and } i < j\}$
  - $Q_k(x_1, \ldots, x_k) := \bigwedge_{1 \leq i < j \leq k} R_E(x_i, x_j)$

- For example, suppose that $G$ is the following graph:

```
  1 -- 2
  |   |
  |   |
  3 -- 4
```

```
IG =  \[
\begin{array}{c|cc}
   & R_E \\
\hline
 1 & 3 \\
 2 & 3 \\
 2 & 4 \\
 3 & 4 \\
\end{array}
\]

Q_3 := R_E(X_1, X_2), R_E(X_1, X_3), R_E(X_2, X_3)
```
Reduction 2: from 3-SAT

**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 $x_i$ or $\neg x_i$), determine whether $\psi$ is satisfiable.
Reduction

- Given $\psi = \varphi_1 \land \cdots \land \varphi_m$ we construct:
  - A relation symbol $R_i/3$ 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 $\varphi_i$
  - $Q(x_1, \ldots, x_n) :\!\!\!\!\!\!\!\!\!: \varphi_1, \ldots, \varphi_m$
  - 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

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

- $\psi: (x \lor y \lor z) \land (\neg x \lor y \lor w) \land (x \lor \neg z \lor \neg w)$
- $Q(x, y, z, w):= R_1(x, y, z), R_2(x, y, w), R_3(x, z, w)$

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

$I =$
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.
Reduction

Given $G = (V, E)$ with $V = \{1, \ldots, n\}$, construct:
Reduction

- Given $G = (V, E)$ with $V = \{1, \ldots, n\}$, construct:
  - $S = \{R_E/2\}$
Reduction

- Given $G = (V, E)$ with $V = \{1, \ldots, n\}$, construct:
  - $S = \{R_E/2\}$
  - $I = \{R_E(c_1, c_2) \mid \{c_1, c_2\} \subseteq \{r, g, b\} \land c_1 \neq c_2\}$
Reduction

- Given $G = (V, E)$ with $V = \{1, \ldots, n\}$, construct:
  - $S = \{R_E/2\}$
  - $I = \{R_E(c_1, c_2) \mid \{c_1, c_2\} \subseteq \{r, g, b\} \land c_1 \neq c_2\}$
  - $Q := \bigwedge_{(i,j) \in E} R_E(x_i, x_j)$
Reduction

- Given $G = (V, E)$ with $V = \{1, \ldots, n\}$, construct:
  - $S = \{R_E/2\}$
  - $I = \{R_E(c_1, c_2) \mid \{c_1, c_2\} \subseteq \{r, g, b\} \land c_1 \neq c_2\}$
  - $Q := \bigwedge_{(i,j) \in E} R_E(x_i, x_j)$

- That's it!
Given $G = (V, E)$ with $V = \{1, \ldots, n\}$, construct:

- $S = \{R_E/2\}$
- $I = \{R_E(c_1, c_2) \mid \{c_1, c_2\} \subseteq \{r, g, b\} \land c_1 \neq c_2\}$
- $Q :\neg \bigwedge_{(i,j) \in E} R_E(x_i, x_j)$

That's it!

Note: $I$ is fixed (6 x 2 table)!
From CQs to Joins

- It is sometimes more comfortable to work with RA joins (and projection) instead of CQs
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:

\[
\pi_{A_1, \ldots, A_k}(T_1 \cdots T_m) \text{ where the } T_i \text{ are distinct relation symbols.}
\]

$\alpha(J)$ 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?

$Q(x, y) : \neg R(x, y, \text{Avia}), R(y, z, x), S(x, x)$
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 $\pi_{A_1,\ldots,A_k}(T_1 \bowtie \cdots \bowtie T_m)$ where the $T_i$ are distinct relation symbols.
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 $\pi_{A_1,\ldots,A_k}(T_1 \Join \cdots \Join T_m)$ 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
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 $\pi_{A_1,\ldots,A_k}(T_1 \Join \cdots \Join T_m)$ 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?

$$Q(x,y) \leftarrow R(x,y,\text{Avia}), R(y,z,x), S(x,x)$$
Translation

- Let \( Q(x) : \varphi_1(x, y), \ldots, \varphi_m(x, y) \) and \( I \) be over \( S \)
Translation

- Let $Q(x) :- \varphi_1(x, y), \ldots, \varphi_m(x, y)$ and $I$ be over $S$
- Each \textit{variable} becomes an \textit{attribute}
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)
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*
Translation

- Let $Q(x) :− \varphi_1(x, y), \ldots, \varphi_m(x, y)$ and $I$ be over $S$
- Each \textit{variable} becomes an \textit{attribute}
- Each \textit{body atom} $\varphi_i$ becomes a unique \textit{relation schema} $T_i$ with the attributes (variables) that appear in $\varphi_i$ (in any order)
- Each \textit{head variable} becomes a \textit{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
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: $Q(x, y) : \neg R(x, y, \text{Avia}), R(y, z, x), S(x, x)$

$$\Rightarrow \pi_{x, y}(T_1(x, y) \Join T_2(x, y, z) \Join T_3(x))$$
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
In the remainder of this lecture, a \textit{CQ expression} is an RA expression of the form

\[ \pi_{\mathbf{A}}(R_1 \bowtie \cdots \bowtie R_k) \]

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

1 Introduction

2 Preliminaries

3 Acyclic Joins

4 Algorithm for Acyclic Joins (Yannakakis)

5 Joins with Hypertree Decompositions

6 Size Bounds and Worst-Case Optimality
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).
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 \(\mathcal{H}\) is a hypergraph, then we denote by:
- \(\text{nodes}(\mathcal{H})\) the set of nodes of \(\mathcal{H}\),
- \(\text{edges}(\mathcal{H})\) the set of hyperedges of \(\mathcal{H}\).
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 \(\mathcal{H}\) is a hypergraph, then we denote by:
- \(\text{nodes}(\mathcal{H})\) the set of nodes of \(\mathcal{H}\),
- \(\text{edges}(\mathcal{H})\) the set of hyperedges of \(\mathcal{H}\).

Let \(\alpha = \pi_A(R_1 \bowtie \cdots \bowtie R_k)\) be a CQ expression.
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 $\mathcal{H}$ is a hypergraph, then we denote by:
- $\text{nodes}(\mathcal{H})$ the set of nodes of $\mathcal{H}$,
- $\text{edges}(\mathcal{H})$ the set of hyperedges of $\mathcal{H}$

Let $\alpha = \pi_A (R_1 \Join \cdots \Join R_k)$ be a CQ expression.

The hypergraph of $\alpha$, denoted $\mathcal{H}_\alpha$, has:
- The attributes in $\alpha$ as the set of nodes
- A hyperedge $e_i$ for each $R_i$, containing the attributes of $R_i$
Example

\[ \pi_{x,y}(R(x,y,z) \Join S(x,u) \Join T(y,z,w)) \]

\[ \mathcal{H}_\alpha \]
A join tree of a hypergraph $\mathcal{H}$ is a tree $T$ with the following properties.
Join Tree

- A *join tree* of a hypergraph $\mathcal{H}$ is a tree $T$ with the following properties
  - The nodes of $T$ are the hyperedges of $\mathcal{H}$
    - In notation, $\text{nodes}(T) = \text{edges}(\mathcal{H})$
A *join tree* of a hypergraph $\mathcal{H}$ is a tree $T$ with the following properties:

- The nodes of $T$ are the hyperedges of $\mathcal{H}$.
  - In notation, $\text{nodes}(T) = \text{edges}(\mathcal{H})$.
- For every $v \in \text{nodes}(\mathcal{H})$, the nodes of $T$ that contain $v$ form a connected subtree of $T$. 
A *join tree* of a hypergraph $\mathcal{H}$ is a tree $T$ with the following properties:

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

**Example:**

![Join Tree Example](image)
Ear Removal

- An *ear* of a hypergraph $\mathcal{H}$ is a hyperedge $e$ of $\mathcal{H}$ 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
Ear Removal

- An \textit{ear} of a hypergraph $\mathcal{H}$ is a hyperedge $e$ of $\mathcal{H}$ such that
  - $e$ is disjoint from all other hyperedges \textit{or}
  - there exists another hyperedge $e'$ where $e \setminus e'$ is disjoint from all other hyperedges
- An \textit{ear removal} on $\mathcal{H}$ is the operation of obtaining a new hypergraph $\mathcal{H}'$ by removing an ear $e$ of $\mathcal{H}$
Ear Removal

- An *ear* of a hypergraph $\mathcal{H}$ is a hyperedge $e$ of $\mathcal{H}$ 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 $\mathcal{H}$ is the operation of obtaining a new hypergraph $\mathcal{H}'$ by removing an ear $e$ of $\mathcal{H}$
  - $\text{nodes}(\mathcal{H}') = \text{nodes}(\mathcal{H})$ and $\text{edges}(\mathcal{H}') = \text{edges}(\mathcal{H}) \setminus \{e\}$
Ear Removal

- An **ear** of a hypergraph $\mathcal{H}$ is a hyperedge $e$ of $\mathcal{H}$ 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 $\mathcal{H}$ is the operation of obtaining a new hypergraph $\mathcal{H}'$ by removing an ear $e$ of $\mathcal{H}$
  - $\text{nodes}(\mathcal{H}') = \text{nodes}(\mathcal{H})$ and $\text{edges}(\mathcal{H}') = \text{edges}(\mathcal{H}) \setminus \{e\}$

- Example:
**Proposition**

Let $\mathcal{H}$ be a hypergraph. The following are equivalent:
Proposition

Let $\mathcal{H}$ be a hypergraph. The following are equivalent:

1. $\mathcal{H}$ has a join tree.
Acyclic Hypergraphs

**Proposition**

Let $\mathcal{H}$ be a hypergraph. The following are equivalent:

1. $\mathcal{H}$ has a join tree.
2. By repeatedly applying ear removal (in any order), one can eliminate all the hyperedges of $\mathcal{H}$.
Acyclic Hypergraphs

**Proposition**

Let $\mathcal{H}$ be a hypergraph. The following are equivalent:

1. $\mathcal{H}$ has a join tree.
2. By repeatedly applying ear removal (in any order), one can eliminate all the hyperedges of $\mathcal{H}$.

If $\mathcal{H}$ satisfies the above conditions, then $\mathcal{H}$ is said to be *acyclic*. 
You will prove the proposition in a home assignment
Comments

- You will prove the proposition in a home assignment
- In particular, you will show *how to build a join tree for a given \( \mathcal{H} \) via ear removal*
You will prove the proposition in a home assignment.

In particular, you will show how to build a join tree for a given $\mathcal{H}$ via ear removal:

- Efficiently!
- (This will be used later in this lecture)
You will prove the proposition in a home assignment.

In particular, you will show how to build a join tree for a given $\mathcal{H}$ via ear removal:

- Efficiently!
- (This will be used later in this lecture)

When $\mathcal{H}$ 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 $\mathcal{H}_\alpha$ is acyclic.
Acyclic CQs

- A CQ expression $\alpha$ is *acyclic* if its associated hypergraph $\mathcal{H}_\alpha$ is acyclic

- *Which of the following is acyclic?*

\[
\left( \bigotimes_{1 \leq i < j \leq n} R_{i,j}(x_i, x_j) \right) \bigotimes S(x_1, \ldots, x_n)
\]
Acyclic CQs

- A CQ expression $\alpha$ is *acyclic* if its associated hypergraph $H_\alpha$ is acyclic.

- *Which of the following is acyclic?*

$$\left( \bigotimes_{1 \leq i < j \leq n} R_{i,j}(x_i, x_j) \right)$$

$$\left( \bigotimes_{1 \leq i < j \leq n} R_{i,j}(x_i, x_j) \bigotimes S(x_1, \ldots, x_n) \right)$$

- *Which of the above can be solved in polynomial total time?*
Table of Contents

1 Introduction

2 Preliminaries

3 Acyclic Joins

4 Algorithm for Acyclic Joins (Yannakakis)

5 Joins with Hypertree Decompositions

6 Size Bounds and Worst-Case Optimality
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.
Main Steps of The Algorithm

**Input:** CQ expression \( \alpha = \pi_A (R_1 \Join \cdots \Join R_k) \), instance \( I \)

1. Compute a join tree \( T \) for \( \mathcal{H}_\alpha \)
2. Apply a **full reduction** to \( I \) according to \( T \)
   - More specifically, replace source relations with *semijoins*
3. Compute \( \alpha(I) \) in **leaf-to-root** order according to \( T \), projecting on only *relevant variables*
   - And eliminating every redundant/irrelevant variable
This can be done (in polynomial time) by the ear-removal procedure.
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
Computing a Join Tree

- This can be done (in polynomial time) by the ear-removal procedure.
- We will view the join tree as \textit{directed} and \textit{ordered} by:
  - Selecting an arbitrary \textit{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 $R_v$
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: $\pi_{x,y}(R(x,y,z) \Join S(x,u) \Join T(y,z,w))$

$R_v = R$ 
$R_{v'} = T$ 
$R_{v''} = S$
Intuition on Full Reduction (1)
Intuition on Full Reduction (2)
Intuition on Full Reduction (2)
The left semijoin of two relations $r$ and $s$, denoted $r \leftarrow s$, is the relation that is obtained from $r$ by selecting only the tuples that have a matching tuple in (i.e., are joinable with) $s$. 
The left semijoin of two relations \( r \) and \( s \), denoted \( r \Join s \), 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:

\[
r \Join s \overset{\text{def}}{=} \pi_A(r \Join s)
\]

where \( A \) is the attribute sequence of \( r \).
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:

\[
r \bowtie s \overset{\text{def}}{=} \pi_A(r \bowtie s)
\]

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 *Inside-Out*, using two passes
Procedure called *Inside-Out*, using two passes

1. Leaf-to-root (inside):
   1. for all nodes \( v \) of \( T \) in leaf-to-root order do
   2. if \( v \) is not the root of \( T \) then
   3. \( r_p := r_p \times r_v \), where \( p \) is the parent of \( v \)
Procedure called *Inside-Out*, using two passes

1. **Leaf-to-root (inside):**
   
   ```
   for all nodes \( v \) of \( T \) in leaf-to-root order do
   if \( v \) is not the root of \( T \) then
   \( r_p := r_p \Join r_v \), where \( p \) is the parent of \( v \)
   ```

2. **Root-to-leaf (out):**

   ```
   for all nodes \( v \) of \( T \) in root-to-leaf order do
   for all children \( c \) of \( v \) do
   \( r_c := r_c \Join r_v \)
   ```
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)

The result is $\text{result}(\text{root}(T))$
Leaf-to-Root Join

- For each node \( v \) of \( T \), let:
  - \( T_v \) be the subtree of \( T \) rooted at \( v \)
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 \( T_v \)
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)
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. for all nodes $v$ of $T$ in leaf-to-root order do
2.   if $v$ is a leaf then
3.     result($v$) := $r_v$
4.   else
5.     let $c_1, \ldots, c_k$ be the children of $v$;
6.     result($v$) := $\pi_{O_v, P_v} (r_v \Join result(c_1) \Join \cdots \Join result(c_k))$
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. for all nodes $v$ of $T$ in leaf-to-root order do
2.   if $v$ is a leaf then
3.     result($v$) := $r_v$
4.   else
5.     let $c_1, \ldots, c_k$ be the children of $v$;
6.     result($v$) := $\pi_{O_v,P_v}(r_v \Join result(c_1) \Join \cdots \Join result(c_k))$
- The result is result(root($T$))
Correctness and Efficiency (Sketch)

- Proof idea:
Proof idea:

- Every tuple that is deleted during the full reduction *does not* contribute to the overall result of the join; *why so?*
- On the other hand, after the full reduction, there are no “hanging tuples” in $r_v$ (every tuple participates in the join)
Proof idea:

- Every tuple that is deleted during the full reduction does not contribute to the overall result of the join; 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)
Proof idea:

- Every tuple that is deleted during the full reduction *does not* contribute to the overall result of the join; *why so?*
- On the other hand, after the full reduction, there are no “hanging tuples” in $r_\nu$ (every tuple participates in the join)
- Similarly, in the evaluation, there are no hanging tuples in $\text{result}(\nu)$ (every tuple can be extended to a join tuple)
- Consequently:
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; 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
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; *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. Introduction
2. Preliminaries
3. Acyclic Joins
4. Algorithm for Acyclic Joins (Yannakakis)
5. Joins with Hypertree Decompositions
6. Size Bounds and Worst-Case Optimality

References
Intuition (1)
Intuition (1)
Intuition (2)
Intuition (2)

```
\begin{array}{c}
  x_2 & x_1 \\
  x_3 & x_{12} \\
  x_4 & x_{11} \\
  x_5 & x_{10} \\
  x_6 & x_9 \\
  x_7 & x_8 \\
\end{array}
\quad \rightarrow \quad
\begin{array}{c}
  x_2 & x_1 \\
  x_3 & x_{12} \\
  x_4 & x_{11} \\
  x_5 & x_{10} \\
  x_6 & x_9 \\
  x_7 & x_8 \\
\end{array}
```
Intuition (2)
Intuition (3)
Tree Decomposition of a Graph

- Let $G$ be a graph
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:
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
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:
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 $e \subseteq \chi(t)$
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 $T$
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 $\max \{|\chi(v)| \mid v \in \text{nodes}(T)\} - 1$
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)
Example (1)
Example (2)
Example (2)

[Diagram showing two hypergraphs with nodes labeled with variables and edges connecting them, illustrating a transformation or operation.]
Tree Decomposition of a Hypergraph

- Definitions of this part taken from Gottlob et al. [GGM+05]
Tree Decomposition of a Hypergraph

- Definitions of this part taken from Gottlob et al. [GGM+05]
- Let \( \mathcal{H} \) be a hypergraph
Tree Decomposition of a Hypergraph

- Definitions of this part taken from Gottlob et al. [GGM+05]
- Let $\mathcal{H}$ be a hypergraph
- A Tree Decomposition (TD) of $\mathcal{H}$ is a pair $(T, \chi)$ with the following properties:
Tree Decomposition of a *Hypergraph*

- Definitions of this part taken from Gottlob et al. [GGM⁺05]
- Let $\mathcal{H}$ be a hypergraph
- A *Tree Decomposition* ($TD$) of $\mathcal{H}$ is a pair $(T, \chi)$ with the following properties:
  - $T$ is a tree
Tree Decomposition of a *Hypergraph*

- Definitions of this part taken from Gottlob et al. [GGM⁺05]
- Let $\mathcal{H}$ be a hypergraph
- A *Tree Decomposition* ($TD$) of $\mathcal{H}$ 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 $\text{nodes}(\mathcal{H})$, so that:
Tree Decomposition of a Hypergraph

- Definitions of this part taken from Gottlob et al. [GGM+05]
- Let $\mathcal{H}$ be a hypergraph
- A *Tree Decomposition* ($TD$) of $\mathcal{H}$ 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 $\text{nodes}(\mathcal{H})$, so that:
    - For every hyperedge $e \in \text{edges}(\mathcal{H})$ there is a node $t$ of $T$ such that $e \subseteq \chi(t)$
Tree Decomposition of a Hypergraph

- Definitions of this part taken from Gottlob et al. [GGM^+05]
- Let $\mathcal{H}$ be a hypergraph
- A **Tree Decomposition** (TD) of $\mathcal{H}$ 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 $\text{nodes}(\mathcal{H})$, so that:
    - For every hyperedge $e \in \text{edges}(\mathcal{H})$ there is a node $t$ of $T$ such that $e \subseteq \chi(t)$
    - Every node $v$ of $\mathcal{H}$ occurs in a connected subtree of $T$; that is, $\{t \in \text{nodes}(T) \mid v \in \chi(t)\}$ induces a connected subtree of $T$
Tree Decomposition of a Hypergraph

- Definitions of this part taken from Gottlob et al. [GGM⁺05]
- Let $\mathcal{H}$ be a hypergraph
- A Tree Decomposition (TD) of $\mathcal{H}$ 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 $\text{nodes}(\mathcal{H})$, so that:
    - For every hyperedge $e \in \text{edges}(\mathcal{H})$ there is a node $t$ of $T$ such that $e \subseteq \chi(t)$
    - Every node $v$ of $\mathcal{H}$ occurs in a connected subtree of $T$; that is, \{ $t \in \text{nodes}(T)$ | $v \in \chi(t)$ \} induces a connected subtree of $T$
  - Note: if $(T, \chi)$ is a TD of $\mathcal{H}$, then $T$ is a join tree over the bags
Examples
Examples
Quality?

- Every hypergraph has a TD!
Quality?

- Every hypergraph has a TD!
- *In what sense is a TD “good”?*
Quality?

- Every hypergraph has a TD!
- *In what sense is a TD “good”?*
- Depends on the context!
Quality?

- Every hypergraph has a TD!
- *In what sense is a TD “good”?*
- Depends on the context!
- In our case, we would like be able to efficiently compute the part of the join that corresponds to each bag
Quality?

- Every hypergraph has a TD!
- *In what sense is a TD “good”?*
- Depends on the context!
- In our case, we would like 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
Quality?

- 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
Generalized Hypertree Decomposition

- Let $\mathcal{H}$ be a hypergraph
Generalized Hypertree Decomposition

- Let $\mathcal{H}$ be a hypergraph
- A *Generalized Hypertree Decomposition* ($GHD$) of $\mathcal{H}$ is a triple $(T, \chi, \lambda)$ such that:
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 $\mathcal{H}$.
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($\mathcal{H}$) that covers $\chi(t)$; that is, $\chi(t) \subseteq \bigcup \lambda(t)$
  - $\bigcup \lambda(t)$ means $\bigcup_{e \in \lambda(t)} e$
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($\mathcal{H}$) that covers $\chi(t)$; that is, $\chi(t) \subseteq \bigcup \lambda(t)$
  - $\bigcup \lambda(t)$ 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 $\max \{|\lambda(t)| \mid t \in \text{nodes}(T)\}$
The generalized hypertree width (\textit{ghw}) of a hypergraph $\mathcal{H}$ is the minimum of the widths of all GHDs of $\mathcal{H}$.
The generalized hypertree width (ghw) of a hypergraph $\mathcal{H}$ is the minimum of the widths of all GHDs of $\mathcal{H}$.

The ghw of a CQ expression $\alpha$ is the ghw of $\mathcal{H}_\alpha$. 
Generalized Hypertree Width

- The generalized hypertree width \((ghw)\) of a hypergraph \(\mathcal{H}\) is the minimum of the widths of all GHDs of \(\mathcal{H}\).
- The ghw of a CQ expression \(\alpha\) is the ghw of \(\mathcal{H}_\alpha\).
- Claim (easy to prove): \(\alpha\) (or \(\mathcal{H}\)) is acyclic if and only if its ghw is 1.
We now show how a small (bounded) ghw 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 $R_e$
CQ Evaluation with a GHD (1)

- Let $\alpha$ be a CQ expression, and let $(T, \chi, \lambda)$ be a GHD of $H_\alpha$. 

\[ ... \]
CQ Evaluation with a GHD (1)

- 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
Let $\alpha$ be a CQ expression, and let $(T, \chi, \lambda)$ be a GHD of $H_\alpha$

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

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

$$r(t) := \pi_{\chi(t)}\left( \bigotimes_{e \in \lambda(t)} r_e \right)$$
CQ Evaluation with a GHD (1)

- 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
  \[ r(t) := \pi_{\chi(t)}\left( \bigotimes_{e \in \lambda(t)} r_e \right) \]

- Next, for each relation $r_i$ find a node $t$ such that $\chi(t)$ contains all the attributes of $R_i$ and set:
  \[ r(t) := r(t) \bigotimes r_i \]
CQ Evaluation with a GHD (1)

- Let \( \alpha \) be a CQ expression, and let \((T, \chi, \lambda)\) be a GHD of \( H_\alpha \)
- Given an instance \( I \), we can compute \( \alpha(I) \) as follows
- For each node \( t \) of \( T \) compute the relation
  \[
  r(t) := \pi_{\chi(t)} \left( \bigotimes_{e \in \lambda(t)} r_e \right)
  \]
- Next, for each relation \( r_i \) find a node \( t \) such that \( \chi(t) \) contains all the attributes of \( R_i \) and set:
  \[
  r(t) := r(t) \bigotimes r_i
  \]
- That is, delete from \( r(t) \) every tuple that cannot be joined with any tuple from \( r_i \)
Now we have the following:
Now we have the following:

\[ \bigotimes_{i=1}^{m} r_i = \bigotimes_{t \in \text{nodes}(T)} r(t) \]
Now we have the following:

- $\bigotimes_{i=1}^{m} r_i = \bigotimes_{t \in \text{nodes}(T)} r(t)$
- $\pi_A\left(\bigotimes_{i=1}^{m} r_i\right) = \pi_A\left(\bigotimes_{t \in \text{nodes}(T)} r(t)\right)$
Now we have the following:

- $\bigotimes_{i=1}^{m} r_i = \bigotimes_{t \in \text{nodes}(T)} r(t)$
- $\pi_A\left(\bigotimes_{i=1}^{m} r_i\right) = \pi_A\left(\bigotimes_{t \in \text{nodes}(T)} r(t)\right)$
- $\pi_A\left(\bigotimes_{t \in \text{nodes}(T)} r(t)\right)$ is an **acyclic CQ expression**
Now we have the following:

- $\bigotimes_{i=1}^{m} r_i = \bigotimes_{t \in \text{nodes}(T)} r(t)$
- $\pi_A\left(\bigotimes_{i=1}^{m} r_i\right) = \pi_A\left(\bigotimes_{t \in \text{nodes}(T)} r(t)\right)$
- $\pi_A\left(\bigotimes_{t \in \text{nodes}(T)} r(t)\right)$ is an **acyclic CQ expression**
- Apply Yannakakis’s to compute $\pi_A\left(\bigotimes_{t \in \text{nodes}(T)} r(t)\right)$
Now we have the following:

- $\bigotimes_{i=1}^{m} r_i = \bigotimes_{t \in \text{nodes}(T)} r(t)$
- $\pi_A\left(\bigotimes_{i=1}^{m} r_i\right) = \pi_A\left(\bigotimes_{t \in \text{nodes}(T)} r(t)\right)$
- $\pi_A\left(\bigotimes_{t \in \text{nodes}(T)} r(t)\right)$ is an acyclic CQ expression

- Apply Yannakakis’s to compute $\pi_A\left(\bigotimes_{t \in \text{nodes}(T)} r(t)\right)$
- That’s it!
Finding a GHD

- It is NP-complete to decide whether a given hypergraph $\mathcal{H}$ has a ghw at most $k$ for any constant $k \geq 3$ [GMS09]
Finding a GHD

- It is NP-complete to decide whether a given a hypergraph $\mathcal{H}$ 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
Finding a GHD

- It is NP-complete to decide whether a given a hypergraph $\mathcal{H}$ 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:
Finding a GHD

- It is NP-complete to decide whether a given a hypergraph $\mathcal{H}$ 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 $\mathcal{H}$ as the minimal width over all hypertree decompositions of $\mathcal{H}$
Finding a GHD

- It is NP-complete to decide whether a given a hypergraph $\mathcal{H}$ 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 $\mathcal{H}$ as the minimal width over all hypertree decompositions of $\mathcal{H}$
- Fact: A hypergraph is acyclic if and only if its hypertree width (and ghw) is 1
**Theorem**

For every constant $k$, CQ expressions with hypertree width at most $k$ can be evaluated in polynomial total time.$^a$

---

$a$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]
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
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 GHD
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 GHD
- Again, this gives polynomial delay [KS06]
Theorem

For every constant $k$, CQ expressions with a generalized hypertree width at most $k$ can be evaluated with polynomial delay.
Table of Contents

1. Introduction
2. Preliminaries
3. Acyclic Joins
4. Algorithm for Acyclic Joins (Yannakakis)
5. Joins with Hypertree Decompositions
6. Size Bounds and Worst-Case Optimality
In this part, we focus on a projection-free join query:

\[
Q \overset{\text{def}}{=} R_1 \Join \cdots \Join R_k
\]
In this part, we focus on a projection-free join query:

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

For \( i = 1, \ldots, k \), denote by \( \text{Att}(R_i) \) the attribute set of \( R_i \).
In this part, we focus on a projection-free join query:

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

For \( i = 1, \ldots, k \), denote by \( \text{Att}(R_i) \) the attribute set of \( R_i \)

Denote by \( \text{Att}(Q) \) the union \( \bigcup_{i=1}^{k} \text{Att}(R_i) \)
In this part, we focus on a projection-free join query:

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

- 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 \( R_i \)
In this part, we focus on a projection-free join query:

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

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 \( r_i \)
Warm-Up Discussion

- How many answers can be for the following queries, in terms of $|r_1|, \ldots, |r_k|$?
Warm-Up Discussion

- How many answers can be for the following queries, in terms of $|r_1|, \ldots, |r_k|$?
  - $R_1(A) \bowtie R_2(B) \bowtie R_3(C)$
How many answers can be for the following queries, in terms of $|r_1|, \ldots, |r_k|$?

1. $R_1(A) \join R_2(B) \join R_3(C)$
2. $R_1(A, B) \join R_2(B, C)$
How many answers can be for the following queries, in terms of $|r_1|, \ldots, |r_k|$?

1. $R_1(A) \bowtie R_2(B) \bowtie R_3(C)$
2. $R_1(A, B) \bowtie R_2(B, C)$
3. $R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(C, A)$
Warm-Up Discussion

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

1. $R_1(A) \Join R_2(B) \Join R_3(C)$
2. $R_1(A, B) \Join R_2(B, C)$
3. $R_1(A, B) \Join R_2(B, C) \Join R_3(C, A)$
4. $R_1(A, B) \Join R_2(B, C) \Join R_3(C, A) \Join R_4(A, B)$
Warm-Up Discussion

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

1. $R_1(A) \bowtie R_2(B) \bowtie R_3(C)$
2. $R_1(A, B) \bowtie R_2(B, C)$
3. $R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(C, A)$
4. $R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(C, A) \bowtie R_4(A, B)$
5. $R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(C, A) \bowtie R_4(A, B, C)$
Recall that \( Q = R_1 \Join \cdots \Join R_k \)
Rough Bound

- Recall that $Q = R_1 \Join \cdots \Join R_k$
- Suppose that $R_{i_1}, \ldots, R_{i_\ell}$ contain all (i.e., cover the) attributes in $\text{Att}(Q)$.
Rough Bound

- Recall that \( Q = R_1 \Join \cdots \Join R_k \).
- Suppose that \( R_{i_1}, \ldots, R_{i_\ell} \) contain all (i.e., cover the) attributes in \( \text{Att}(Q) \).
- Then, each tuple \( t \in Q(D) \) is the combination of tuples from \( r_{i_1}, \ldots, r_{i_\ell} \) that agree on the common attributes.
  - And some combinations may not be tuples, due to the other relations.
Recall that $Q = R_1 \Join \cdots \Join R_k$.

Suppose that $R_{i_1}, \ldots, R_{i_\ell}$ contain all (i.e., cover the) attributes in $\text{Att}(Q)$.

Then, each tuple $t \in Q(D)$ is the combination of tuples from $r_{i_1}, \ldots, r_{i_\ell}$ that agree on the common attributes.

And some combinations may not be tuples, due to the other relations.

Q: How many such combinations can there be?
Rough Bound

- Recall that $Q = R_1 \Join \cdots \Join R_k$
- Suppose that $R_{i_1}, \ldots, R_{i_\ell}$ contain all (i.e., cover the) attributes in $\text{Att}(Q)$.
- Then, each tuple $t \in Q(D)$ is the combination of tuples from $r_{i_1}, \ldots, r_{i_\ell}$ that agree on the common attributes
  - And some combinations may not be tuples, due to the other relations
- Q: How many such combinations can there be?
- A: At most $|r_{i_1}| \times \cdots \times |r_{i_\ell}|$
Recall that $Q = R_1 \Join \cdots \Join R_k$

Suppose that $R_{i_1}, \ldots, R_{i_\ell}$ contain all (i.e., cover the) attributes in $\text{Att}(Q)$.

Then, each tuple $t \in Q(D)$ is the combination of tuples from $r_{i_1}, \ldots, r_{i_\ell}$ that agree on the common attributes.

And some combinations may not be tuples, due to the other relations.

Q: How many such combinations can there be?

A: At most $|r_{i_1}| \times \cdots \times |r_{i_\ell}|$

Hence, $|Q(D)| \leq |r_{i_1}| \times \cdots \times |r_{i_\ell}|$
An *edge cover* of $Q$ 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 $a_i = 1$.
An *edge cover* of $Q$ 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:

If $(a_1, \ldots, a_k)$ is an edge cover of $Q$, then:

$$Q(D) \leq \prod_{i=1}^{k} |r_i|^{a_i}$$
Rephrase via Edge Cover

- An edge cover of $Q$ 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:

  If $(a_1, \ldots, a_k)$ is an edge cover of $Q$, then:

  $$Q(D) \leq \prod_{i=1}^{k} |r_i|^{a_i}$$

- This bound, however, is not tight; we get tightness via the fractional edge cover.
Fractional Edge Cover

- An edge cover of $Q$ 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 $a_i = 1$. 
Fractional Edge Cover

- An edge cover of $Q$ 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 $a_i = 1$
- A fractional edge cover of $Q$ is a sequence $(w_1, \ldots, w_k)$ in $[0, 1]^k$ such that for every $A \in \text{Att}$ we have

$$\sum_{i \mid A \in \text{Att}(R_i)} w_i \geq 1$$
Fractional Edge Cover

- An *edge cover* of $Q$ 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 $a_i = 1$.

- A *fractional edge cover* of $Q$ is a sequence $(w_1, \ldots, w_k)$ in $[0, 1]^k$ such that for every $A \in \text{Att}$ we have

  $$\sum_{i | A \in \text{Att}(R_i)} w_i \geq 1$$

- A fractional edge cover $(w_1, \ldots, w_k)$ of $Q$ is *optimal* if $\sum_{i=1}^k w_i$ is minimal.
Fractional Edge Cover

- An *edge cover* of $Q$ 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 $a_i = 1$.
- A *fractional edge cover* of $Q$ is a sequence $(w_1, \ldots, w_k)$ in $[0, 1]^k$ such that for every $A \in \text{Att}$ we have
  \[ \sum_{i|A \in \text{Att}(R_i)} w_i \geq 1 \]
- A fractional edge cover $(w_1, \ldots, w_k)$ of $Q$ is *optimal* if $\sum_{i=1}^k w_i$ is minimal.
- Denote by $(w_1^*, \ldots, w_k^*)$ an optimal edge cover of $Q$. 

The AGM Bound

AGM Bound [GM14, AGM13]

**Theorem**

- For every fractional edge cover \((w_1, \ldots, w_k)\) of \(Q\) we have

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

- For every \(N_0 \in \mathbb{N}\) there is a database \(D\) with \(N > N_0\) tuples such that

  \[
  Q(D) \geq \prod_{i=1}^{k} |r_i|^{w^*_i}
  \]

  and \(|r_i| = |r_j|\) whenever \(w^*_i, w^*_j > 0\).
Examples

What is the fractional edge cover of the following join?

\[ R(A, B) \bowtie S(B, C) \bowtie T(C, A) \]
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 \( Q_{k}^{LW} \) is the following:

\[ Q_{k}^{LW} \overset{\text{def}}{=} R_1(x_2, \ldots, x_k) \Join R_2(x_1, x_3, \ldots, x_k) \Join \]
\[ \ldots \Join R_k(x_1, x_3, \ldots, x_{k-1}) \]

What is the fractional edge cover of \( Q_{k}^{LW} \)?
LP for Finding the Upper Bound

Minimize: \( \sum_{i=1}^{k} \log(|r_i|) \cdot x_i \) \hspace{1cm} subject to:

\[ \forall A \in \text{Att}(Q): \sum_{i|A \in \text{Att}(R_i)} x_i \geq 1 \]

\[ \forall R_i : x_i \geq 0 \]
Worst-Case Optimality

- An algorithm for computing $Q$ is **worst-case optimal** if its running time is $O(f(|r_1|, \ldots, |r_k|))$ where $f(n_1, \ldots, n_k)$ is the maximal $|Q(D)|$ over all databases $D$ with $|r_i| = n_i$.
Worst-Case Optimality

- An algorithm for computing $Q$ is **worst-case optimal** if its running time is $O(\max_{D} |Q(D)|)$ where $f(n_1, \ldots, n_k)$ 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]
Worst-Case Optimality

- An algorithm for computing $Q$ is **worst-case optimal** if its running time is $O(f(|r_1|, \ldots, |r_k|))$ where $f(n_1, \ldots, n_k)$ 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
An algorithm for computing $Q$ is **worst-case optimal** if its running time is $O(f(|r_1|, \ldots, |r_k|))$ where $f(n_1, \ldots, n_k)$ 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 II


End of lecture 4

Computing Joins