Principles of Managing Uncertain Data

Lecture 4: Computing Joins
Table of Contents

1 Introduction

2 Acyclic Joins

3 Algorithm for Acyclic Joins (Yannakakis)

4 Joins with Hypertree Decompositions
Table of Contents

1 Introduction

2 Acyclic Joins

3 Algorithm for Acyclic Joins (Yannakakis)

4 Joins with Hypertree Decompositions
We have learned the concepts of data complexity and combined complexity.
Previous Lecture

- 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 \( P = NP \)
Today we consider only *combined complexity*
Today we consider only \textit{combined complexity}

We have seen an example of a \textit{fragment} of CQs that can be evaluated in polynomial total time

\hspace{1em} Namely, $n$-length paths
Today we consider only *combined complexity*

- We have seen an example of a *fragment* of CQs that can be evaluated in polynomial total time
  - Namely, $n$-length paths
- Today we learn more a general fragment of tractable CQs
Today we consider only *combined complexity*

- We have seen an example of a *fragment* of CQs that can be evaluated in polynomial total time
  - Namely, $n$-length paths
- Today we learn more a general fragment of tractable CQs
  - Acyclic CQs
Today we consider only combined complexity.

We have seen an example of a fragment of CQs that can be evaluated in polynomial total time.

- Namely, \( n \)-length paths.

Today we learn more a general fragment of tractable CQs:

- Acyclic CQs.
- More generally, CQs of a bounded hypertree width.
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

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

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
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 **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 \( Q(\mathbf{x}) :- \varphi_1(\mathbf{x}, \mathbf{y}), \ldots, \varphi_m(\mathbf{x}, \mathbf{y}) \) 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 \( \mu(\varphi_i(\mathbf{x}, \mathbf{y})) \) is a fact of \( I \) for every \( i = 1, \ldots, m \).

  - \( \mu(\varphi_i(\mathbf{x}, \mathbf{y})) \) 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 $Q(\mathbf{x}) := \varphi_1(\mathbf{x}, \mathbf{y}), \ldots, \varphi_m(\mathbf{x}, \mathbf{y})$ 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 $\mu(\varphi_i(\mathbf{x}, \mathbf{y}))$ is a fact of $I$ for every $i = 1, \ldots, m$.
  - $\mu(\varphi_i(\mathbf{x}, \mathbf{y}))$ 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 $\mathbf{x}$.
- 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.
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.
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 $U$ of $V$ such that $|U| = k$ and every two nodes in $U$ are neighbours.
Given $G = (V, E)$ with $V = \{1, \ldots, n\}$, and $k$, construct:

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

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

```
   1   2
  / \ /  \
 3---4
```

$I_G = \begin{array}{c}
R_E \\
1 & 3 \\
2 & 3 \\
2 & 4 \\
3 & 4 \\
\end{array}$

$Q_3 := R_E(X_1, X_2), R_E(X_1, X_3), R(X_2, X_3)$
Problem Def. (3-SAT)

Given a propositional formula $\psi = \phi_1 \land \cdots \land \phi_m$ over the variables $x_1, \ldots, x_n$, where each $\phi_i$ is a disjunction of three atomic formulas (each has the form $x_i$ or $\neg x_i$), determine whether $\psi$ is satisfiable.
Given $\psi = \varphi_1 \land \cdots \land \varphi_m$ we construct:

1. A relation symbol $R_i$ for each $\varphi_i$
2. An atomic formula $\phi_i = R_i(x, y, z)$ where $x, y$ and $z$ are the variables that appear in $\varphi_i$
3. $Q(x_1, \ldots, x_n) :- \phi_1, \ldots, \phi_m$
4. The instance $I$ has in the relation $R_i$ all 7 tuples $(b_1, b_2, b_3) \in \{0, 1\}^3$ that satisfy $\phi_i$

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

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

\[
\begin{array}{c|c|c}
I & R_1 & R_2 \\
\hline
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]
It is sometimes more comfortable to work with RA joins (and projection) instead of CQs
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:

\[ \alpha(J) \text{ and } Q(I) \text{ are "the same" } \]

That is, there is a straightforward translation between the two.
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.
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.
- $\alpha(J)$ 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?

$$Q(x,y) :\neg R(x,y,Avia), R(y,z,x), S(x,x)$$
Let $Q(x) :- \varphi_1(x, y), \ldots, \varphi_m(x, y)$ and $I$ be over $S$. 

Translation
Translation

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

- Let \( Q(x) \leftarrow \varphi_1(x, y), \ldots, \varphi_m(x, y) \) 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
Translation

- Let $Q(x) :\neg \varphi_1(x, y), \ldots, \varphi_m(x, y)$ 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 from the $A_i$s
In the remainder of this lecture, a \textit{CQ expression} is an RA expression of the form
\[
\pi_A(R_1 \Join \cdots \Join R_k)
\]

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

1. Introduction

2. Acyclic Joins

3. Algorithm for Acyclic Joins (Yannakakis)

4. Joins with Hypertree Decompositions
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 \(\alpha = \pi_A(R_1 \Join \cdots \Join R_k)\) 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

$$\pi_{x,y}(R(x, y, z) \bowtie S(x, u) \bowtie T(y, z, w))$$
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, $\text{nodes}(T) = \text{edges}(\mathcal{H})$
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, $\text{nodes}(T) = \text{edges}(\mathcal{H})$.
  - For every $v \in \text{nodes}(\mathcal{H})$, the nodes of $T$ that contain $v$ form a connected subtree of $T$. 

---

20/52
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 Diagram](image-url)
Ear Removal

- An *ear* of a hypergraph $\mathcal{H}$ is a hyperedge $e$ of $\mathcal{H}$ such that for some other hyperedge $e' \neq e$, every node in $e$ is either in $e'$ or does not appear in any other hyperedge (i.e., unique to $e$).
Ear Removal

- An *ear* of a hypergraph $\mathcal{H}$ is a hyperedge $e$ of $\mathcal{H}$ such that for some other hyperedge $e' \neq e$, every node in $e$ is either in $e'$ or does not appear in any other hyperedge (i.e., unique to $e$).
- An *ear removal* on $\mathcal{H}$ is the operation of obtaining a new hypergraph $\mathcal{H}'$ by removing an ear $e$ of $\mathcal{H}$. 

An ear of a hypergraph $\mathcal{H}$ is a hyperedge $e$ of $\mathcal{H}$ such that for some other hyperedge $e' \neq e$, every node in $e$ is either in $e'$ or does not appear in any other hyperedge (i.e., unique to $e$).

An ear removal on $\mathcal{H}$ is the operation of obtaining a new hypergraph $\mathcal{H}'$ by removing an ear $e$ of $\mathcal{H}$. 

---

Example:

```
  y  z  x  u  w
  y  z  x  u  w
  y  z  x  u  w
  y  z  x  u  w
```

21/52
Ear Removal

- An **ear** of a hypergraph $\mathcal{H}$ is a hyperedge $e$ of $\mathcal{H}$ such that for some other hyperedge $e' \neq e$, every node in $e$ is either in $e'$ or does not appear in any other hyperedge (i.e., unique to $e$).
- 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 for some other hyperedge $e' \neq e$, every node in $e$ is either in $e'$ or does not appear in any other hyperedge (i.e., unique to $e$).
- 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:
Acyclic Hypergraphs

**Proposition**

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

Proposition

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

1. \( \mathcal{H} \) has a join tree.
**Proposition**

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

1. $\mathcal{H}$ has a join tree.
2. By repeatedly applying ear removal, 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, 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
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.
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*
  - 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), then acyclicity is the usual notion of graph acyclicity (forest).
Acyclic CQs

- A CQ expression $\alpha$ is \textit{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?*

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

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

1. Introduction
2. Acyclic Joins
3. Algorithm for Acyclic Joins (Yannakakis)
4. Joins with Hypertree Decompositions
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 $\alpha = \pi_A(R_1 \bowtie \cdots \bowtie R_k)$, instance $I$

1. Compute a join tree $T$ for $\mathcal{H}_\alpha$
2. Apply a *full reduction* to $I$ according to $T$
3. Compute $\alpha(I)$ in leaf-to-root order according to $T$, projecting out every *redundant variable*
Computing a Join Tree

- This can be done (in polynomial time) by the ear-removal procedure
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
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 \( R_v \)
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: $\pi_{x,y}(R(x,y,z) \Join S(x,u) \Join T(y,z,w))$

```
\begin{align*}
R_v &= R \\
R_{v'} &= T \\
R_{v''} &= S
\end{align*}
```
Intuition on Full Reduction (1)
Intuition on Full Reduction (2)
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 \( v \)
Applying a Full Reduction

- 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 $v$
  2. Root-to-leaf (out):
     1. for all nodes $v$ of $T$ in root-to-leaf order do
     2. for all children $c$ of $v$ do
     3. $r_c := r_c \times r_v$
Leaf-to-Root Join

- For each node \( v \) of \( T \), let:
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. let $c_1, \ldots, c_k$ be the children of $v$;
3. $\text{result}(v) := \pi_{O_v,P_v}(r_v \Join \text{result}(c_1) \Join \cdots \Join \text{result}(c_k))$
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. let $c_1, \ldots, c_k$ be the children of $v$;
3. $\text{result}(v) := \pi_{O_v, P_v}(r_v \bowtie \text{result}(c_1) \bowtie \cdots \bowtie \text{result}(c_k))$

- The result is $\text{result}(\text{root}(T))$
Proof idea:

Every tuple that is deleted 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 \( \text{result} \) (every tuple can be extended to a join tuple).

Consequently:

We compute the correct result.

The size of each \( \text{result} \) is polynomial in the size of the final output.
Proof idea:

- Every tuple that is deleted 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 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 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:
Proof idea:

- Every tuple that is deleted 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 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 result($v$) (every tuple can be extended to a join tuple)
  - Consequently:
    - We compute the correct result
    - The size of each result($v$) is polynomial in the size of the final output
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Acyclic Joins</td>
</tr>
<tr>
<td>3</td>
<td>Algorithm for Acyclic Joins (Yannakakis)</td>
</tr>
<tr>
<td>4</td>
<td>Joins with Hypertree Decompositions</td>
</tr>
</tbody>
</table>
Intuition (1)
Intuition (1)
Intuition (2)
Intuition (2)
Intuition (2)
Intuition (3)
Intuition (3)
Intuition (3)
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 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 \( H \) 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, \( \{t \in \text{nodes}(T) \mid v \in \chi(t)\} \) 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 nodes($\mathcal{H}$), so that:
    - For every hyperedge $e \in$ 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, \{ $t \in$ 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

\[39/52\]
Another Example (Think Join)
Another Example (Think Join)
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 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
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, $\chi(t) \subseteq \bigcup_{e \in \lambda(t)} e$

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


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 $\bigcup_{e \in \lambda(t)} e$
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:
  - $(T, \chi)$ is a tree decomposition of $\mathcal{H}$
  - $\lambda$ is a function that maps every node $t$ of $T$ to a subset of $\text{edges}(\mathcal{H})$ 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 \textit{GHDs} of \( \mathcal{H} \).
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 $\mathcal{H}_\alpha$. 

The generalized hypertree width $(\text{ghw})$ 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 $\text{ghw}$ is 1.
Generalized Hypertree Width

- The \textit{generalized hypertree width (ghw)} of a hypergraph $\mathcal{H}$ is \textit{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.
Utilizing Bounded ghw

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

\[ \text{Given an instance } I, \text{ we can compute } \alpha(I) \text{ as follows:} \]

1. For each node $t$ of $T$, compute the relation $r(t) := \pi_{\chi(t)}(t) \mid_{e \in \lambda(t)}$
2. Next, for each relation $r_i$, find a node $t$ such that $\chi(t)$ contains all the attributes of $R_i$ and set:
   \[ r(t) := r(t) \setminus r_i \]
   That is, delete from $r(t)$ every tuple that cannot be joined with any tuple from $r_i$. 

\[ \text{That is, delete from } r(t) \text{ every tuple that cannot be joined with any tuple from } r_i. \]
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
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

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

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

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 a 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]


End of lecture 4
Computing Joins