CSE544
Data Management

Lectures 11-12
Advanced Query Processing
Announcements

- HW3 due on Friday
- No lecture on Monday (President’s day)
- Project milestone due next Friday
Advanced Query Processing

Current optimization techniques: optimal plan given current statistics

- Ignores asymptotic runtime
- Sometimes asymptotic is provably bad

Advanced techniques: find optimal asymptotic runtime
Examples

SELECT count(*)
FROM Author;
Answer: 2419705
Time: < 1s
Asymptotic: O(N)

SELECT count(*)
FROM Publication;
Answer: 4659997
Time: < 1s
Asymptotic: O(N)

SELECT count(*)
FROM Author, Publication;
Answer: 2419705 * 4659997
Timeout
Asymptotic: O(N²)
Should be: O(N)
Examples

SELECT count(*)
FROM Author x,
    Authored y,
    Publication z
WHERE x.authorid = y.authorid
    and y.pubid = z.pubid
    and z.year < 2015

Optimize this! (At home…)
Outline

• Acyclic queries, Yannakakis algorithm

• Tree decomposition of cyclic queries

• Worst-case optimal algorithm; next week
Conjunctive Queries

• A CQ is:

\[ Q(X) : - R_1(X_1), R_2(X_2), ..., R_m(X_m) \]

• Same as a single datalog rule
Types of CQ

- **Full CQ**: all variables are head variables
  
  \[ Q(x,y,z,u) : - R(x,y), S(y,z), T(z,u) \]
  
  \[ Q(*) : - \ldots \]

- **Boolean CQ**: no variables are head variables
  
  \[ Q() : - R(x,y), S(y,z), T(z,u) \]

- **CQ with aggregates**:
  
  \[ Q(x, \text{sum}(u)) : - R(x,y), S(y,z), T(z,u) \]
Generalized Distributivity Law

• Basic idea: group-by commutes with join if we write it the right way
Generalized Distributivity Law

\[ Q(\text{count}(\ast)) = R(x,y), S(y,z) \]
Generalized Distributivity Law

\[ Q(\text{count}(*)) = R(x,y), S(y,z) \]

<table>
<thead>
<tr>
<th>R</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>d</td>
<td>f</td>
</tr>
<tr>
<td>g</td>
<td>h</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>y</td>
</tr>
<tr>
<td>b</td>
<td>g</td>
</tr>
<tr>
<td>b</td>
<td>k</td>
</tr>
<tr>
<td>h</td>
<td>m</td>
</tr>
</tbody>
</table>
Generalized Distributivity Law

\[ Q(\text{count}(*)) = R(x,y), S(y,z) \]

\[
\begin{array}{c|c}
R: & x & y \\
\hline
a & b \\
\hline
c & b \\
\hline
d & f \\
\hline
g & h \\
\end{array}
\]

\[
\begin{array}{c|c}
S: & y & z \\
\hline
b & g \\
\hline
b & k \\
\hline
h & m \\
\end{array}
\]

Answer = 5
Generalized Distributivity Law

\[ Q(\text{count}(\ast)) = R(x,y), S(y,z) \]

R: \[
\begin{array}{cc}
    x & y \\
    a & b \\
    c & b \\
    d & f \\
    g & h \\
\end{array}
\]

S: \[
\begin{array}{cc}
    y & z \\
    b & g \\
    b & k \\
    h & m \\
\end{array}
\]

Answer = 5

Runtime = \(O(N^2)\)
Generalized Distributivity Law

\[ Q(\text{count}(\ast)) = R(x,y), S(y,z) \]

**R:**

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>d</td>
<td>f</td>
</tr>
<tr>
<td>g</td>
<td>h</td>
</tr>
</tbody>
</table>

**S:**

<table>
<thead>
<tr>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>g</td>
</tr>
<tr>
<td>b</td>
<td>k</td>
</tr>
<tr>
<td>h</td>
<td>m</td>
</tr>
</tbody>
</table>

Answer = 5

Runtime = O(N²)

A(y, \text{count}(x) \text{ as } c) = R(x,y)
B(y, \text{count}(z) \text{ as } d) = S(y,z)
Q(\text{sum}(c*d)) = A(y,c), B(y,d)
Generalized Distributivity Law

\[ Q(\text{count}(\star)) = R(x,y), S(y,z) \]

**R:**

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>d</td>
<td>f</td>
</tr>
<tr>
<td>g</td>
<td>h</td>
</tr>
</tbody>
</table>

**S:**

<table>
<thead>
<tr>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>g</td>
</tr>
<tr>
<td>b</td>
<td>k</td>
</tr>
<tr>
<td>h</td>
<td>m</td>
</tr>
</tbody>
</table>

\[ \text{Answer} = 5 \]

\[ \text{Runtime} = O(N^2) \]

\[ \sum_{y} \text{count}(\star) \]

**A:**

<table>
<thead>
<tr>
<th>y</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
</tr>
<tr>
<td>h</td>
<td>1</td>
</tr>
</tbody>
</table>

**B:**

<table>
<thead>
<tr>
<th>y</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>h</td>
<td>1</td>
</tr>
</tbody>
</table>

**Answer:** 5

**Runtime:** \(O(N^2)\)
Generalized Distributivity Law

\[ Q(\text{count}(\ast)) = R(x,y), S(y,z) \]

<table>
<thead>
<tr>
<th>R:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>h</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>y</td>
<td>z</td>
</tr>
<tr>
<td>b</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>k</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>m</td>
<td></td>
</tr>
</tbody>
</table>

Answer = 5

Runtime = \( O(N^2) \)

\[ \sum(c \cdot d) = A(y,c), B(y,d) \]

\begin{tabular}{c|c}
  A: & y & c \\
  & b & 2 \\
  & f & 1 \\
  & h & 1 \\
\end{tabular}

\begin{tabular}{c|c}
  B: & y & c \\
  & b & 2 \\
  & h & 1 \\
\end{tabular}

\begin{tabular}{c|c|c}
  A \bowtie B: & y & c & d \\
  & b & 2 & 2 \\
  & h & 1 & 1 \\
\end{tabular}
Generalized Distributivity Law

\[ Q(\text{count}(\ast)) = R(x,y), S(y,z) \]

**R:**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>h</td>
<td></td>
</tr>
</tbody>
</table>

**S:**

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>k</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>m</td>
<td></td>
</tr>
</tbody>
</table>

\[ \land \land_{y} \gamma \text{count}(\ast) \]

Answer = 5

\[ \land \land_{y} \gamma \text{sum}(c\ast d) \]

Runtime = \(O(N^2)\)

**A(y,count(x) as c) = R(x,y)\)**

**B(y,count(z) as d) = S(y,z)\)**

\[ Q(\text{sum}(c\ast d)) = A(y,c), B(y,d) \]

**A:**

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**B:**

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**A \bowtie B:**

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**A**: \(y\leftarrow c\)

**B**: \(y\leftarrow d\)

\[ R(x,y) \bowtie S(y,z) \]

Runtime = \(O(N^2)\)
**Generalized Distributivity Law**

\[ Q(\text{count}(*)) = R(x,y), S(y,z) \]

### Example

**R:**
- x: a, b
- y: c, b
- z: c, b

**A:**
- y: b, 2
- f: 1
- h: 1

**B:**
- y: b, 2
- h: 1

**Output:**
- Answer = 5
- Runtime = O(N^2)

**Diagram:**
- \( \gamma \text{count}(*) \)
- \( \gamma \text{sum}(c*d) \)

**Runtime:**
- O(N)
- O(N^2)
Acyclic Queries

Q is *acyclic* if its atoms can be placed in a tree $T$ such that for every variable the set of nodes that contain that variable form a connected component.
Acyclic Queries

Q is *acyclic* if its atoms can be placed in a tree T such that for every variable the set of nodes that contain that variable form a connected component.

T is called *join tree*
Acyclic Queries

Q is *acyclic* if its atoms can be placed in a tree T such that for every variable the set of nodes that contain that variable form a connected component
Acyclic Queries

Q is **acyclic** if its atoms can be placed in a tree T such that for every variable the set of nodes that contain that variable form a connected component.

T is called the **join tree**.
Acyclic Queries

Q is *acyclic* if its atoms can be placed in a tree T such that for every variable the set of nodes that contain that variable form a connected component.

T is called a *join tree*.

E.g. z forms a connected component.
Acyclic Queries

Q is **acyclic** if its atoms can be placed in a tree $T$ such that for every variable the set of nodes that contain that variable form a connected component.

T is called *join tree*.

\[ R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]
Acyclic Queries

Q is *acyclic* if its atoms can be placed in a tree T such that for every variable the set of nodes that contain that variable form a connected component.

T is called a *join tree*.

R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m)

R(x,y), S(y,z), T(z,x)
Acyclic Queries

Q is *acyclic* if its atoms can be placed in a tree T such that for every variable the set of nodes that contain that variable form a connected component.

T is called the *join tree*.
Acyclic Queries

Q is acyclic if its atoms can be placed in a tree T such that for every variable the set of nodes that contain that variable form a connected component.

T is called join tree.
A Theorem

Q = an acyclic query Q that is:
• Boolean, or
• Full, or
• Aggregate with $\leq 1$ group-by variable

Theorem Q can be computed in time*:
$$\tilde{O}(|Input| + |Output|)$$

* $\tilde{O}$ means plus a logarithmic factor (for sorting)
Yannakakis Algorithm

• Step 1: semi-join reduction
  – Pick any root node in the join tree of Q
  – Semi-join reduction from leaves to root
  – Semi-join reduction from root to leaves

• Step 2:
  – Compute the joins bottom up,
  – Push group-by down
Example: Full CQ

\[ Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]
Example: Full CQ

\[
Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m)
\]
Example: Full CQ

\[ Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]
Example: Full CQ

\[ Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]

-- Leaves to root:
K :- K \bowtie L
Example: Full CQ

\[ Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]

-- Leaves to root:
\[ K \dashv K \bowtie L \]
\[ S \dashv S \bowtie T \]
Example: Full CQ

\[ Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]

-- Leaves to root:
\[ K \text{ :- } K \bowtie L \]
\[ S \text{ :- } S \bowtie T \]
\[ S \text{ :- } S \bowtie K \]
Example: Full CQ

\( Q(\ast) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \)

--- Leaves to root:
- \( K \vdash K \times L \)
- \( S \vdash S \times T \)
- \( S \vdash S \times K \)
- \( R \vdash R \times S \)
Example: Full CQ

\[ Q(*) = R(x, y), S(y, z, u), T(y, z, w), K(z, v), L(v, m) \]

--- Leaves to root:
- \( K :: K \Join L \)
- \( S :: S \Join T \)
- \( S :: S \Join K \)
- \( R :: R \Join S \)

--- Root to leaves:
Example: Full CQ

\[ Q(\star) = R(x, y), S(y, z, u), T(y, z, w), K(z, v), L(v, m) \]

- **Leaves to root:**
  - \( K : - K \Join L \)
  - \( S : - S \Join T \)
  - \( S : - S \Join K \)
  - \( R : - R \Join S \)

- **Root to leaves:**
  - \( S : - S \Join R \)
Example: Full CQ

\[ Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]

-- Leaves to root:
- \( K : \Delta K \Join L \)
- \( S : \Delta S \Join T \)
- \( S : \Delta S \Join K \)
- \( R : \Delta R \Join S \)

-- Root to leaves:
- \( S : \Delta S \Join R \)
- \( T : \Delta T \Join S \)
Example: Full CQ

\[ Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]

-- Leaves to root:
\[ K : - K \bowtie L \]
\[ S : - S \bowtie T \]
\[ S : - S \bowtie K \]
\[ R : - R \bowtie S \]

-- Root to leaves:
\[ S : - S \bowtie R \]
\[ T : - T \bowtie S \]
\[ K : - K \bowtie S \]
Example: Full CQ

$Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m)$

---

- **Leaves to root:**
  - $K : - K \bowtie L$
  - $S : - S \bowtie T$
  - $S : - S \bowtie K$
  - $R : - R \bowtie S$

- **Root to leaves:**
  - $S : - S \bowtie R$
  - $T : - T \bowtie S$
  - $K : - K \bowtie S$
  - $L : - L \bowtie K$
Example: Full CQ

\[ Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]

Join (any order in the tree)

-- Leaves to root:
- \( K \bowtie L \)
- \( S \bowtie T \)
- \( S \bowtie K \)
- \( R \bowtie S \)

-- Root to leaves:
- \( S \bowtie R \)
- \( T \bowtie S \)
- \( K \bowtie S \)
- \( L \bowtie K \)
Example: Full CQ

$Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m)$

Join (any order in the tree)

---

---

-- Leaves to root:
K :- K $\bowtie$ L
S :- S $\bowtie$ T
S :- S $\bowtie$ K
R :- R $\bowtie$ S

-- Root to leaves:
S :- S $\bowtie$ R
T :- T $\bowtie$ S
K :- K $\bowtie$ S
L :- L $\bowtie$ K
Example: Full CQ

Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m)

Join (any order in the tree)

--- Leaves to root:
K :: K × L
S :: S × T
S :: S × K
R :: R × S

--- Root to leaves:
S :: S × R
T :: T × S
K :: K × S
L :: L × K

\[ \bowtie y \]
\[ \bowtie z \]
\[ \bowtie v \]
\[ \bowtie y,z \]
\[ \bowtie y,z,u \]
\[ \bowtie y,z,u,w \]
\[ \bowtie z,v,m \]
Example: Full CQ

\[ Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]

---

Join (any order in the tree)

---

-- Leaves to root:
- \( K \cdot K \leftarrow L \)
- \( S \cdot S \leftarrow T \)
- \( S \cdot S \leftarrow K \)
- \( R \cdot R \leftarrow S \)

---

-- Root to leaves:
- \( S \cdot S \leftarrow R \)
- \( T \cdot T \leftarrow S \)
- \( K \cdot K \leftarrow S \)
- \( L \cdot L \leftarrow K \)
Example: Full CQ

Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m)

Join (any order in the tree)

-- Leaves to root:
K :: K ⋊ L
S :: S ⋊ T
S :: S ⋊ K
R :: R ⋊ S

-- Root to leaves:
S :: S ⋊ R
T :: T ⋊ S
K :: K ⋊ S
L :: L ⋊ K

x,y,z,u,w,v,m
Example: Full CQ

\[ Q(\ast) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]

Join (any order in the tree)

---

---

-- Leaves to root:
- \( K \leq K \bowtie L \)
- \( S \leq S \bowtie T \)
- \( S \leq S \bowtie K \)
- \( R \leq R \bowtie S \)

-- Root to leaves:
- \( S \leq S \bowtie R \)
- \( T \leq T \bowtie S \)
- \( K \leq K \bowtie S \)
- \( L \leq L \bowtie K \)

Runtime:
- Every semi-join takes time \( \tilde{O}(|Input|) \)
- Every join takes time \( \tilde{O}(|Output|) \)
Example: Full CQ

\[ Q(*) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]

Join (any order in the tree)

R(x,y) \bowtie y \rightarrow x,y,u,w,v,m

K(z,v) \bowtie z \rightarrow y,z,u,w,v,m

S(y,z,u) \bowtie u \rightarrow y,z,u,w,v,m

T(y,z,w) \bowtie w \rightarrow y,z,u,w,v,m

L(v,m) \bowtie v \rightarrow y,z,u,w,v,m

K(z,v) \bowtie \emptyset \rightarrow z,v,m

T(y,z,w) \bowtie \emptyset \rightarrow y,z,w,v,m

S(y,z,u) \bowtie \emptyset \rightarrow y,z,u,w,v,m

L(v,m) \bowtie \emptyset \rightarrow v,m

R(x,y) \bowtie \emptyset \rightarrow x,y,u,v,m

What happens if we skip the semi-joins?

Runtime:
- Every semi-join takes time \( \tilde{O}(|Input|) \)
- Every join takes time \( \tilde{O}(|Output|) \)
Example: CQ with Aggregates

\[ Q(x,\text{sum}(m)) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]
Example: CQ with Aggregates

\[
Q(x, \text{sum}(m)) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m)
\]

Semi-join as before

Too big!
Example: CQ with Aggregates

\[ Q(x, \text{sum}(m)) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]
Example: CQ with Aggregates

\[ Q(x, \text{sum}(m)) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]
Example: CQ with Aggregates

\[ Q(x, \text{sum}(m)) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]
Example: CQ with Aggregates

\[ Q(x, \text{sum}(m)) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]
Example: CQ with Aggregates

Example CQ with Aggregates:

\[ Q(x, \text{sum}(m)) = R(x, y), S(y, z, u), T(y, z, w), K(z, v), L(v, m) \]
Example: CQ with Aggregates

\[ Q(x, \text{sum}(m)) = R(x, y), S(y, z, u), T(y, z, w), K(z, v), L(v, m) \]
Example: CQ with Aggregates

\[ Q(x, \text{sum}(m)) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]

Semi-join as before

\[ \gamma_{y, \text{sum}(w*u*t \rightarrow k)} \]
\[ \gamma_{z, \text{sum}(m*s \rightarrow t)} \]
\[ \gamma_{v, \text{sum}(m)} \]
\[ \gamma_{y, \text{count}(w)} \]
\[ \gamma_{y, \text{count}(u)} \]
\[ \gamma_{y, \text{count}(w)} \]
\[ \gamma_{y, \text{count}(u)} \]
\[ \gamma_{v, \text{sum}(m)} \]

Runtime:
- Semi-joins: \( \tilde{O}(|Input|) \)
- Join/group-by: \( \tilde{O}(|Input|) \)
Example: CQ with Aggregates

\[ Q(x, \text{sum}(m)) = R(x,y), S(y,z,u), T(y,z,w), K(z,v), L(v,m) \]

Runtime:
- Semi-joins: \( \tilde{O}(|\text{Input}|) \)
- Join/group-by: \( \tilde{O}(|\text{Input}|) \)
What about group-by multiple attributes?

\[ Q(x, z, m, \text{sum}(w)) : - \ldots \]

- Can apply the same principle, but runtime may be polynomial in Input or Output
Discussion

What is the query is disconnected?

• Simply compute each connected component separately, then take their cartesian product, or regular product, as needed.

<Select:
SELECT count(*)
FROM Author, Publication;
</Select>

<Select:  
SELECT x.firstName, y.year, count(*)
FROM Author x, Publication y
GROUP By x.firstName, y.year;
</Select>
Discussion

Which join order do we choose?

• Yannakakis algorithm doesn’t specify: any join order ensures runtime is:
  \( \tilde{O}(|Input| + |Output|) \)

• BUT: join order may impact the constant significantly, and in practice that matters
Discussion

• Some acyclic queries have more than one join tree, and each tree has several join orders
• Example: $Q(x) = R(x), S(x), T(x)$
Discussion

• Database optimizers rarely do semi-join reduction
  – When they do, they sometimes call it a *magic set optimization* (we’ll explain next)

• Reason: when semi-join is ineffective, then it increases cost by a factor of 3
Discussion

• Magic set optimizations
• Semi-join reductions can also be applied to recursive datalog program
• Called *magic set optimizations*; quite complicated
Discussion

- Magic set optimizations
- Semi-join reductions can also be applied to recursive datalog program
- Called *magic set optimizations*; quite complicated

\[
\begin{align*}
T(x,y) & : \text{Parent}(x,y) \\
T(x,y) & : T(x,z), \text{Parent}(z,y) \\
Q(y) & : T(\text{‘Alice’},y)
\end{align*}
\]
Discussion

• Magic set optimizations
• Semi-join reductions can also be applied to recursive datalog program
• Called *magic set optimizations*; quite complicated

\[
\begin{align*}
T(x,y) & \leftarrow \text{Parent}(x,y) \\
T(x,y) & \leftarrow T(x,z), \text{Parent}(z,y) \\
Q(y) & \leftarrow T(\text{‘Alice’},y) \\
Q(y) & \leftarrow \text{Parent}(‘Alice’,y) \\
Q(y) & \leftarrow Q(x), \text{Parent}(x,y)
\end{align*}
\]
Discussion

• A full reducer for Q is a sequence of semi-joins after which every tuple contributes to at least one answer

• **Theorem.** Q has a full reducer iff it is acyclic

• **Proof:** if Q is acyclic, then Yannakakis’ algorithm. If Q is cyclic, assume w.l.o.g.:

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

(show in class that it has no full reducer)
Testing if Q is Acyclic

An _ear_ of Q is an atom R(X) with the following property:

1. Let $X' \subseteq X$ be the set of join variables (occurring some other atom)
2. There exists some other atom S(Y) such that $X' \subseteq Y$
Testing if Q is Acyclic

An ear of Q is an atom R(X) with the following property:

- Let \( X' \subseteq X \) be the set of join variables (occurring some other atom)
- There exists some other atom S(Y) such that \( X' \subseteq Y \)

GYO algorithm (Graham, Yu, Özsoyoğlu) for acyclicity:

- While Q has an ear R(X), remove R(X) from Q
- If all atoms were removed, then Q is acyclic
- If atoms remain but there is no ear, then Q is cyclic
Outline

• Acyclic queries, Yannakakis algorithm

• Tree decomposition of cyclic queries

• Worst-case optimal algorithm; next week
Tree Decomposition

**Def**  *Tree decomposition* is \((T, \chi), \chi: \text{Nodes}(T) \to 2^{\text{Vars}(Q)}\) s.t.:

1. \(\forall A \in \text{Atoms}(Q) \exists t \in \text{Nodes}(T), \text{Vars}(A) \subseteq \chi(t)\)
2. \(\forall x \in \text{Vars}(Q), \{t \mid x \in \chi(t)\}\) is connected
Tree Decomposition

Def *Tree decomposition* is \((T, \chi)\), \(\chi: \text{Nodes}(T) \rightarrow 2^{\text{Vars}(Q)}\) s.t.:

1. \(\forall A \in \text{Atoms}(Q) \exists t \in \text{Nodes}(T), \ \text{Vars}(A) \subseteq \chi(t)\)
2. \(\forall x \in \text{Vars}(Q), \{t \mid x \in \chi(t)\}\) is connected

\[Q(x,\ldots,m) = R(x,y) \land A(y,s) \land B(x,s) \land S(y,z,u) \land T(y,z,w) \land C(z,w,t) \land D(w,t,y) \land E(t,y,z) \land K(z,v) \land F(z,v) \land L(v,m)\]
**Def** *Tree decomposition* is \((T, \chi)\), \(\chi: \text{Nodes}(T) \rightarrow 2^{\text{Vars}(Q)}\) s.t.:

1. \(\forall A \in \text{Atoms}(Q) \; \exists t \in \text{Nodes}(T), \; \text{Vars}(A) \subseteq \chi(t)\)
2. \(\forall x \in \text{Vars}(Q), \{t \mid x \in \chi(t)\} \) is connected

\[Q(x, ..., m) = R(x,y) \land A(y,s) \land B(x,s) \land S(y,z,u) \land T(y,z,w) \land C(z,w,t) \land D(w,t,y) \land E(t,y,z) \land K(z,v) \land F(z,v) \land L(v,m)\]
Def *Tree decomposition* is $(T, \chi)$, $\chi: \text{Nodes}(T) \rightarrow 2^{\text{Vars}(Q)}$ s.t.:

1. $\forall A \in \text{Atoms}(Q) \exists t \in \text{Nodes}(T), \text{Vars}(A) \subseteq \chi(t)$
2. $\forall x \in \text{Vars}(Q), \{t \mid x \in \chi(t)\}$ is connected

$$Q(x, \ldots, m) = R(x, y) \land A(y, s) \land B(x, s) \land S(y, z, u) \land T(y, z, w) \land C(z, w, t) \land D(w, t, y) \land E(t, y, z) \land K(z, v) \land F(z, v) \land L(v, m)$$

Where does the atom $R(x, y)$ occur?
Def: Tree decomposition is \((T, \chi)\), \(\chi: \text{Nodes}(T) \rightarrow 2^{\text{Vars}(Q)}\) s.t.:

1. \(\forall A \in \text{Atoms}(Q) \exists t \in \text{Nodes}(T), \ \text{Vars}(A) \subseteq \chi(t)\)
2. \(\forall x \in \text{Vars}(Q), \{t \mid x \in \chi(t)\}\) is connected

\[Q(x,\ldots,m) = R(x,y) \land A(y,s) \land B(x,s) \land S(y,z,u) \land T(y,z,w) \land C(z,w,t) \land D(w,t,y) \land E(t,y,z) \land K(z,v) \land F(z,v) \land L(v,m)\]

Where does the atom \(R(x,y)\) occur? Here
**Def** *Tree decomposition* is \((T, \chi), \chi: \text{Nodes}(T) \rightarrow 2^{\text{Vars}(Q)}\) s.t.: 

1. \(\forall A \in \text{Atoms}(Q) \exists t \in \text{Nodes}(T), \text{Vars}(A) \subseteq \chi(t)\)
2. \(\forall x \in \text{Vars}(Q), \{t \mid x \in \chi(t)\}\) is connected

\[
Q(x, ..., m) = R(x,y) \land A(y,s) \land B(x,s) \land S(y,z,u) \land T(y,z,w) \land C(z,w,t) \land D(w,t,y) \\
\land E(t,y,z) \land K(z,v) \land F(z,v) \land L(v,m)
\]

Where does the atom \(R(x,y)\) occur? 
Is the set of nodes containing \(z\) connected?

\(x,y,s\) 
\(y,z,u\) 
\(y,z,w,t\) 
\(z,v\) 
\(v,m\)
**Tree Decomposition**

**Def** *Tree decomposition* is $(T, \chi)$, $\chi: \text{Nodes}(T) \rightarrow 2^{\text{Vars}(Q)}$ s.t.:

1. $\forall A \in \text{Atoms}(Q) \exists t \in \text{Nodes}(T), \text{Vars}(A) \subseteq \chi(t)$
2. $\forall x \in \text{Vars}(Q)$, $\{t \mid x \in \chi(t)\}$ is connected

$$Q(x,\ldots,m) = R(x,y) \land A(y,s) \land B(x,s) \land S(y,z,u) \land T(y,z,w) \land C(z,w,t) \land D(w,t,y) \land E(t,y,z) \land K(z,v) \land F(z,v) \land L(v,m)$$

Where does the atom $R(x,y)$ occur? Is the set of nodes containing $z$ connected? Yes, yes, yes.
**Def** Tree decomposition is \((T, \chi), \chi: \text{Nodes}(T) \rightarrow 2^{\text{Vars}(Q)}\) s.t.:

1. \(\forall A \in \text{Atoms}(Q) \quad \exists t \in \text{Nodes}(T), \quad \text{Vars}(A) \subseteq \chi(t)\)
2. \(\forall x \in \text{Vars}(Q), \quad \{t \mid x \in \chi(t)\} \text{ is connected}\)

\[Q(x,...,m) = R(x,y) \land A(y,s) \land B(x,s) \land S(y,z,u) \land T(y,z,w) \land C(z,w,t) \land D(w,t,y) \land E(t,y,z) \land K(z,v) \land F(z,v) \land L(v,m)\]

These are called bags.
Tree Decomposition

**Def** Tree decomposition is \((T, \chi)\), \(\chi: \text{Nodes}(T) \rightarrow 2^{\text{Vars}(Q)}\) s.t.:

1. \(\forall A \in \text{Atoms}(Q) \ \exists t \in \text{Nodes}(T), \ \text{Vars}(A) \subseteq \chi(t)\)
2. \(\forall x \in \text{Vars}(Q), \ \{t | x \in \chi(t)\}\) is connected

\[
Q(x,\ldots,m) = R(x,y) \land A(y,s) \land B(x,s) \land S(y,z,u) \land T(y,z,w) \land C(z,w,t) \land D(w,t,y) \\
\land E(t,y,z) \land K(z,v) \land F(z,v) \land L(v,m)
\]

full CQ: \(Q_t(x,y,s) = R(x,y) \land A(y,s) \land B(x,s)\)
**Tree Decomposition**

**Def** *Tree decomposition* is \((T, \chi)\), \(\chi: \text{Nodes}(T) \rightarrow 2^{\text{Vars}(Q)}\) s.t.:

1. \(\forall A \in \text{Atoms}(Q) \exists t \in \text{Nodes}(T), \; \text{Vars}(A) \subseteq \chi(t)\)
2. \(\forall x \in \text{Vars}(Q), \{t \mid x \in \chi(t)\} \) is connected

\[
Q(x,\ldots,m) = R(x,y) \land A(y,s) \land B(x,s) \land S(y,z,u) \land T(y,z,w) \land C(z,w,t) \land D(w,t,y) \\
\land E(t,y,z) \land K(z,v) \land F(z,v) \land L(v,m)
\]

**Full CQ:** \(Q_t(x,y,s) = R(x,y) \land A(y,s) \land B(x,s)\)
Def Tree decomposition is $(T, \chi)$, $\chi: \text{Nodes}(T) \rightarrow 2^{\text{Vars}(Q)}$ s.t.: (1) $\forall A \in \text{Atoms}(Q) \exists t \in \text{Nodes}(T), \text{Vars}(A) \subseteq \chi(t)$ (2) $\forall x \in \text{Vars}(Q), \{t | x \in \chi(t)\}$ is connected

$$Q(x,\ldots,m) = R(x,y) \land A(y,s) \land B(x,s) \land S(y,z,u) \land T(y,z,w) \land C(z,w,t) \land D(w,t,y) \land E(t,y,z) \land K(z,v) \land F(z,v) \land L(v,m)$$

Computing $Q(D)$:
(1) Compute all full CQ’s $Q_t$
(2) Run Yannakakis’ on the join tree
Time $O(N?? + |\text{Output}|)$
Recap

To compute a query Q proceed as follows
1. Find a tree decomposition of Q
2. For each tree node (“bag”) compute its local query $Q_t$
3. Run Yannakakis on the resulting acyclic query

Runtime is dominated step 2
Will discuss step 2 next
Tree-width

\[ \text{Def } tw(Q) = \min_T \max_{t \in \text{Nodes}(T)} |x(t)| - 1 \]
Tree-width

Definition

\[ \text{Def } tw(Q) = \min_T \max_{t \in \text{Nodes}(T)} |x(t)| - 1 \]

This is the standard definition in graph theory.
Tree-width

\[
\text{Def } \text{tw}(Q) = \min_T \max_{t \in \text{Nodes}(T)} |x(t)| - 1
\]

\[
\text{tw}(Q) = 3
\]
Tree-width

**Def** \( \text{tw}(Q) = \min_T \max_{t \in \text{Nodes}(T)} |X(t)| - 1 \)

\[ \text{tw}(Q) = 3 \]

Naïve iteration for \( Q_t \): Runtime for \( Q \): \( O(N^{\text{tw}(Q)+1} + |\text{Output}|) \)
Def $tw(Q) = \min_T \max_{t \in \text{Nodes}(T)} |\chi(t)| - 1$

This is the standard definition in graph theory. Tree-width gives the complexity of the most naïve algorithm.

$tw(Q) = 3$

Naïve iteration for $Q_t$: Runtime for $Q$: $O(N^{tw(Q)+1} + |\text{Output}|)$
Generalized Hypertree Width

\[ \text{Def } \text{ghtw}(Q) = \min_T \max_{t \in \text{Nodes}(T)} \rho(Q_t) \]

\( \rho = \) edge covering number
Generalized Hypertree Width

**Def** \( \text{ghtw}(Q) = \min_T \max_{t \in \text{Nodes}(T)} \rho(Q_t) \)

\( \rho = \text{edge covering number} \)

- \( R(x,y) \), \( A(y,s), B(x,s) \)
- \( S(y,z,u) \)
- \( T(y,z,w), C(z,w,t) \), \( D(w,t,y), E(t,y,z) \)
- \( K(z,v), F(z,v) \)
- \( L(v,m) \)
**Generalized Hypertree Width**

**Def** $\text{ghtw}(Q) = \min_T \max_{t \in \text{Nodes}(T)} \rho(Q_t)$

$\rho = \text{edge covering number}$

- $R(x,y), A(y,s), B(x,s)$
- $S(y,z,u)$
- $T(y,z,w), C(z,w,t), D(w,t,y), E(t,y,z)$
- $K(z,v), F(z,v)$
- $L(v,m)$
Generalized Hypertree Width

\[ \text{Def } \text{ghtw}(Q) = \min_T \max_{t \in \text{Nodes}(T)} \rho(Q_t) \]

\( \rho = 2 \)

\( R(x,y), A(y,s), B(x,s) \)

\( S(y,z,u) \)

\( T(y,z,w), C(z,w,t) \)

\( D(w,t,y), E(t,y,z) \)

\( K(z,v), F(z,v) \)

\( L(v,m) \)

\( \rho = \text{edge covering number} \)
Generalized Hypertree Width

**Definition**

$$\text{ghtw}(Q) = \min_T \max_{t \in \text{Nodes}(T)} \rho(Q_t)$$

- **\( \rho = 2 \)**
- \( R(x,y), A(y,s), B(x,s) \)
- \( S(y,z,u) \)
- \( T(y,z,w), C(z,w,t), D(w,t,y), E(t,y,z) \)
- \( K(z,v), F(z,v) \)
- \( L(v,m) \)

**Note:** \( \rho \) = edge covering number
**Generalized Hypertree Width**

**Definition**

\[
\text{ghtw}(Q) = \min_T \max_{t \in \text{Nodes}(T)} \rho(Q_t)
\]

\(\rho\) = edge covering number

\(\rho = 2\)

- \(R(x,y)\), \(A(y,s), B(x,s)\)
- \(S(y,z,u)\)
- \(T(y,z,w), C(z,w,t)\), \(D(w,t,y), E(t,y,z)\)

\(\rho = 1\)

- \(K(z,v), F(z,v)\)
- \(L(v,m)\)
Generalized Hypertree Width

**Definition:**

\[ \text{ghtw}(Q) = \min_T \max_{t \in \text{Nodes}(T)} \rho(Q_t) \]

\( \rho = \) edge covering number

\[ \rho = \begin{cases} 2 & \text{for } R(x,y), A(y,s), B(x,s) \\ 1 & \text{for } S(y,z,u), K(z,v), F(z,v), L(v,m) \\ 2 & \text{for } T(y,z,w), C(z,w,t), D(w,t,y), E(t,y,z) \end{cases} \]

\[ \text{ghtw}(Q) = 2 \]
Generalized Hypertree Width

**Definition**
\[ \text{ghtw}(Q) = \min_T \max_{t \in \text{Nodes}(T)} \rho(Q_t) \]

\[ \rho = \text{edge covering number} \]

\[ \rho = 2 \]
- \( R(x,y), A(y,s), B(x,s) \)
- \( S(y,z,u) \)
- \( T(y,z,w), C(z,w,t) \)
- \( D(w,t,y), E(t,y,z) \)

\[ \rho = 1 \]
- \( K(z,v), F(z,v) \)
- \( L(v,m) \)

\[ \rho = 2 \]

**Nested loop join for** \( Q_t \):

**Runtime for** \( Q \):
\[ O(N^{\text{ghtw}(Q)} + |\text{Output}|) \]
Generalized Hypertree Width

\textbf{Def} \( ghtw(Q) = \min_T \max_{t \in \text{Nodes}(T)} \rho(Q_t) \)

\( \rho = \) edge covering number

\( ghtw(Q) = 2 \)

\( GHTW \) is the complexity of a still naïve algorithm

Nested loop join for \( Q_t \):
Runtime for \( Q \): \( O(N^{ghtw(Q)} + |\text{Output}|) \)
**Fractional Hypertree Width**

**Def** \( \text{fhtw}(Q) = \min_{T} \max_{t \in \text{Nodes}(T)} \rho^*(Q_t) \)

- \( \rho^* = 3/2 \) for \( R(x,y), A(y,s), B(x,s) \)
- \( \rho^* = 4/3 \) for \( T(y,z,w), C(z,w,t), D(w,t,y), E(t,y,z) \)

- \( \rho^* = 1 \) for \( S(y,z,u), K(z,v), F(z,v) \)
- \( \rho^* = 1 \) for \( L(v,m) \)

- \( fhtw(Q) = 3/2 \)

- Runtime for \( Q \): \( O(N^{\text{fhtw}(Q)} + |\text{Output}|) \)

- \( \rho^* = \) Fractional edge covering number

Next lecture
Outline

• Acyclic queries, Yannakakis algorithm

• Tree decomposition of cyclic queries

• Worst-case optimal algorithm; next week