Finite Model Theory
Lecture 6: Conjunctive Queries

Spring 2025

Query Containment for CQ – Wrapup

Review: Problem Definition

Q_1 is contained in Q_2 if $\forall \boldsymbol{D}$, $Q_1(\boldsymbol{D}) \subseteq Q_2(\boldsymbol{D})$. Notation: $Q_1 \subseteq Q_2$

Q_1 is equivalent to Q_2 if $\forall \boldsymbol{D}$, $Q_1(\boldsymbol{D}) = Q_2(\boldsymbol{D})$. Notation: $Q_1 \equiv Q_2$.

$$\boxed{Q_1 \equiv Q_2} \text{ iff } \boxed{Q_1 \subseteq Q_2 \text{ and } Q_2 \subseteq Q_1}$$

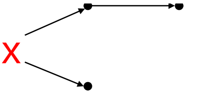
Review: The Homomorphism Criterion

$$\boxed{Q_1 \subseteq Q_2} \text{ iff } \boxed{\exists h : Q_2 \to Q_1} \text{ iff } \boxed{\boldsymbol{D}_{Q_1} \vDash Q_2}$$

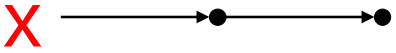
Review: The Homomorphism Criterion

$$\boxed{Q_1 \subseteq Q_2} \text{ iff } \boxed{\exists h : Q_2 \rightarrow Q_1} \text{ iff } \boxed{\boldsymbol{D}_{Q_1} \models Q_2}$$

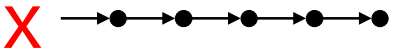
$$Q_1(x) = E(x, y) \wedge E(y, z) \wedge E(x, w)$$



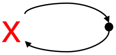
$$Q_2(x) = E(x, u) \wedge E(u, v)$$



$$Q_3(x) = E(x, u_1) \wedge E(u_1, u_2) \wedge \cdots \wedge E(u_4, u_5)$$



$$Q_4(x) = E(x, y) \wedge E(y, x)$$



$$\boxed{Q_4 \subset Q_3 \subset Q_1 \equiv Q_2}$$

Review: Query Containment for $CQ(<, \leq, \neq)$

Homomorphism is a sufficient condition for containment of $CQ(<, \leq, \neq)$

$$Q = R(x, y, z) \wedge (x < y) \wedge (y < z) \qquad Q' = R(u, v, w) \wedge (u \leq w)$$

Review: Query Containment for $CQ(<, \leq, \neq)$

Homomorphism is a sufficient condition for containment of $CQ(<, \leq, \neq)$

$$Q = R(x, y, z) \wedge (x < y) \wedge (y < z) \qquad Q' = R(u, v, w) \wedge (u \leq w)$$

$h : (u, v, w) \mapsto (x, y, z)$ maps $u \leq w$ to $x \leq z$, and $Q \vDash x \leq z$.

$$\boxed{Q \subseteq Q'}$$

Review: Query Containment for $CQ(<, \leq, \neq)$

Homomorphism is not necessary for containment of $CQ(<, \leq, \neq)$

$$Q = S(x, y) \wedge S(y, z) \wedge (x < z) \qquad Q' = S(u, v) \wedge (u < v)$$

$\boxed{Q \subseteq Q'}$ but there is no homomorphism $Q' \to Q$

Review: Query Containment for $CQ(<, \leq, \neq)$

$Q_{\leq}$ is the extension of Q with a total preorder on $\text{Vars}(Q) \cup \text{Const}(Q)$

Theorem (Necessary and Sufficient Condition)

Let Q, Q' be $CQ^{<,\leq,\neq}$ queries. The following conditions are equivalent:

(1) $Q \subseteq Q'$ $\quad$ *($\forall \boldsymbol{D}$, if $\boldsymbol{D} \models Q$ then $\boldsymbol{D} \models Q'$)*

(2) For any consistent total preorder $\leq$ on Q, $\exists h : Q' \to Q_{\leq}$.

Proof: please see the slides of the previous lecture.

Example

$$\boxed{Q = S(x,y) \wedge S(y,z) \wedge (x < z)} \qquad \boxed{Q' = S(u,v) \wedge (u < v)}$$

Let's prove that $\boxed{Q \subseteq Q'}$.

Example

$$Q = S(x,y) \wedge S(y,z) \wedge (x < z) \qquad Q' = S(u,v) \wedge (u < v)$$

Let's prove that $Q \subseteq Q'$.

5 consistent total preorders on Q:

$$\begin{aligned}
Q_1 =& S(x,y) \wedge S(y,z) \wedge (y < x) \wedge (y < z) \\
Q_2 =& S(x,y) \wedge S(y,z) \wedge (x = y) \wedge (y < z) \\
Q_3 =& S(x,y) \wedge S(y,z) \wedge (x < y) \wedge (y < z) \\
Q_4 =& S(x,y) \wedge S(y,z) \wedge (x < y) \wedge (y = z) \\
Q_5 =& S(x,y) \wedge S(y,z) \wedge (x < y) \wedge (z < y)
\end{aligned}$$

Example

$$\boxed{Q = S(x,y) \wedge S(y,z) \wedge (x < z)} \qquad \boxed{Q' = S(u,v) \wedge (u < v)}$$

Let's prove that $\boxed{Q \subseteq Q'}$.

5 consistent total preorders on Q:

$$\begin{aligned}
Q_1 =& S(x,y) \wedge S(y,z) \wedge (y < x) \wedge (y < z) \\
Q_2 =& S(x,y) \wedge S(y,z) \wedge (x = y) \wedge (y < z) \\
Q_3 =& S(x,y) \wedge S(y,z) \wedge (x < y) \wedge (y < z) \\
Q_4 =& S(x,y) \wedge S(y,z) \wedge (x < y) \wedge (y = z) \\
Q_5 =& S(x,y) \wedge S(y,z) \wedge (x < y) \wedge (z < y)
\end{aligned}$$

In each case, either $(u,v) \mapsto (x,y)$ or $(u,v) \mapsto (y,z)$ is a homomorphism.

Example

$$Q = S(x,y) \wedge S(y,z) \wedge (x < z) \qquad Q' = S(u,v) \wedge (u < v)$$

Let's prove that $Q \subseteq Q'$.

5 consistent total preorders on Q:

$$\begin{aligned}
Q_1 =& S(x,y) \wedge S(y,z) \wedge (y < x) \wedge (y < z)\\
Q_2 =& S(x,y) \wedge S(y,z) \wedge (x = y) \wedge (y < z)\\
Q_3 =& S(x,y) \wedge S(y,z) \wedge (x < y) \wedge (y < z)\\
Q_4 =& S(x,y) \wedge S(y,z) \wedge (x < y) \wedge (y = z)\\
Q_5 =& S(x,y) \wedge S(y,z) \wedge (x < y) \wedge (z < y)
\end{aligned}$$

In each case, either $(u,v) \mapsto (x,y)$ or $(u,v) \mapsto (y,z)$ is a homomorphism.

Example

$$\boxed{Q = S(x,y) \wedge S(y,z) \wedge (x < z)} \qquad \boxed{Q' = S(u,v) \wedge (u < v)}$$

Let's prove that $\boxed{Q \subseteq Q'}$.

5 consistent total preorders on Q:

$$
\begin{aligned}
Q_1 =& S(x,y) \wedge S(y,z) \wedge (y < x) \wedge (y < z) \\
Q_2 =& S(x,y) \wedge S(y,z) \wedge (x = y) \wedge (y < z) \\
Q_3 =& S(x,y) \wedge S(y,z) \wedge (x < y) \wedge (y < z) \\
Q_4 =& S(x,y) \wedge S(y,z) \wedge (x < y) \wedge (y = z) \\
Q_5 =& S(x,y) \wedge S(y,z) \wedge (x < y) \wedge (z < y)
\end{aligned}
$$

In each case, either $(u,v) \mapsto (x,y)$ or $(u,v) \mapsto (y,z)$ is a homomorphism.

Complexity of Query Containment for $CQ(<, \leq, \neq)$

Theorem

The problem given $Q, Q' \in CQ(<, \leq, \neq)$, check $Q \subseteq Q'$ is Π_2^p-complete.

Complexity of Query Containment for $CQ(<, \leq, \neq)$

Theorem

The problem given $Q, Q' \in CQ(<, \leq, \neq)$, check $Q \subseteq Q'$ is Π_2^p-complete.

Review: query containment for CQ is NP-complete.

Complexity of Query Containment for $CQ(<, \leq, \neq)$

Theorem

The problem given $Q, Q' \in CQ(<, \leq, \neq)$, check $Q \subseteq Q'$ is Π_2^p-complete.

Review: query containment for CQ is NP-complete.
Reduction from 3CNF Φ. Example:

$$\Phi = (\neg X \vee \neg Y \vee Z) \wedge (\neg X \vee Y \vee \neg Z) \wedge (X \vee U \vee W).$$

Complexity of Query Containment for $CQ(<, \leq, \neq)$

Theorem

The problem given $Q, Q' \in CQ(<, \leq, \neq)$, check $Q \subseteq Q'$ is Π_2^p-complete.

Review: query containment for CQ is NP-complete.
Reduction from 3CNF Φ. Example:

$$\Phi = (\neg X \vee \neg Y \vee Z) \wedge (\neg X \vee Y \vee \neg Z) \wedge (X \vee U \vee W).$$

$$Q'_\Phi = C(z, x, y) \wedge C(y, x, z) \wedge A(x, u, w)$$

$$Q = A(0, 0, 1) \wedge \ldots \wedge D(1, 1, 0)$$

in class: describe Q

Complexity of Query Containment for $CQ(<, \leq, \neq)$

Theorem

The problem given $Q, Q' \in CQ(<, \leq, \neq)$, check $Q \subseteq Q'$ is Π_2^p-complete.

Review: query containment for CQ is NP-complete.
Reduction from 3CNF Φ. Example:

$$\Phi = (\neg X \vee \neg Y \vee Z) \wedge (\neg X \vee Y \vee \neg Z) \wedge (X \vee U \vee W).$$

$$Q'_\Phi = C(z, x, y) \wedge C(y, x, z) \wedge A(x, u, w)$$

$$Q = A(0, 0, 1) \wedge \ldots \wedge D(1, 1, 0)$$

in class: describe Q

$h : Q'_\Phi \to Q$ is a homomorphism iff $h(\Phi) = \texttt{True}$

Complexity of Query Containment for $CQ(<, \leq, \neq)$

Theorem

The problem given $Q, Q' \in CQ(<, \leq, \neq)$, check $Q \subseteq Q'$ is Π_2^p-complete.

Proof: Membership in Π_2^p follows from:

$Q \subseteq Q'$ iff for all extensions $Q_\leq$, there exists a homomorphisms $Q' \to Q_\leq$.

This is in Π_2^p by definition.

It remains to prove Π_2^P-hardness.

Complexity of Query Containment for $CQ(<, \leq, \neq)$

Theorem

The problem given $Q, Q' \in CQ(<, \leq, \neq)$, check $Q \subseteq Q'$ is Π_2^p-complete.

Proof: Reduction from $\forall\exists 3CNF$: $\boxed{\Psi = \forall X_1 \cdots \forall X_k \exists X_{k+1} \cdots \exists X_n \Phi}$

Complexity of Query Containment for $CQ(<, \leq, \neq)$

Theorem

The problem given $Q, Q' \in CQ(<, \leq, \neq)$, check $Q \subseteq Q'$ is Π_2^p-complete.

Proof: Reduction from $\forall\exists 3CNF$: $\boxed{\Psi = \forall X_1 \cdots \forall X_k \exists X_{k+1} \cdots \exists X_n \Phi}$

Start with Q, Q'_Φ as before: $\boxed{h : Q'_\Phi \to Q}$ iff $\boxed{h(\Phi) = \texttt{True}}$

Complexity of Query Containment for $CQ(<, \leq, \neq)$

Theorem

The problem given $Q, Q' \in CQ(<, \leq, \neq)$, check $Q \subseteq Q'$ is Π_2^p-complete.

Proof: Reduction from $\forall\exists 3CNF$: $\boxed{\Psi = \forall X_1 \cdots \forall X_k \exists X_{k+1} \cdots \exists X_n \Phi}$

Start with Q, Q'_Φ as before: $\boxed{h : Q'_\Phi \to Q}$ iff $\boxed{h(\Phi) = \texttt{True}}$

For each universal variable X_i:

- add $S(0, u_i, v_i) \wedge S(1, v_i, w_i) \wedge (u_i < w_i)$ to Q.
- add $S(x_i, s_i, t_i) \wedge (s_i < t_i)$ to Q'_Φ.

Complexity of Query Containment for $CQ(<, \leq, \neq)$

Theorem

The problem given $Q, Q' \in CQ(<, \leq, \neq)$, check $Q \subseteq Q'$ is Π_2^p-complete.

Proof: Reduction from $\forall\exists 3CNF$: $\Psi = \forall X_1 \cdots \forall X_k \exists X_{k+1} \cdots \exists X_n \Phi$

Start with Q, Q'_Φ as before: $h : Q'_\Phi \to Q$ iff $h(\Phi) = \texttt{True}$

For each universal variable X_i:

- add $S(0, u_i, v_i) \wedge S(1, v_i, w_i) \wedge (u_i < w_i)$ to Q.
- add $S(x_i, s_i, t_i) \wedge (s_i < t_i)$ to Q'_Φ.

$Q \subseteq Q'_\Phi$ iff for every extension $Q_\leq$, $\exists h : Q'_\Phi \to Q_\leq$

Complexity of Query Containment for $CQ(<, \leq, \neq)$

Theorem

The problem given $Q, Q' \in CQ(<, \leq, \neq)$, check $Q \subseteq Q'$ is Π_2^p-complete.

Proof: Reduction from $\forall\exists 3CNF$: $\Psi = \forall X_1 \cdots \forall X_k \exists X_{k+1} \cdots \exists X_n \Phi$

Start with Q, Q'_Φ as before: $h : Q'_\Phi \to Q$ iff $h(\Phi) = \texttt{True}$

For each universal variable X_i:

- add $S(0, u_i, v_i) \wedge S(1, v_i, w_i) \wedge (u_i < w_i)$ to Q.
- add $S(x_i, s_i, t_i) \wedge (s_i < t_i)$ to Q'_Φ.

$Q \subseteq Q'_\Phi$ iff for every extension $Q_\leq$, $\exists h : Q'_\Phi \to Q_\leq$

For some $Q_\leq$, $(x_i, s_i, t_i) \overset{h}{\mapsto} (0, u_i, v_i)$, for others $(x_i, s_i, t_i) \overset{h}{\mapsto} (1, v_i, w_i)$

$Q \subseteq Q'_\Phi$ iff Ψ is $\texttt{True}$

Summary

- A few extensions of CQ still have decidable containment: inequalities, safe negation $\neg$, certain aggregates $\text{sum}, \text{min}, \text{max}, \text{count}$.

- But while containment/equivalence for pure CQ/UCQ is very elegant, extensions add significant difficulties.

Query Minimization

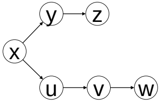
Query Minimization for CQ

Definition (Minimal Query)

Q is minimal if, $\forall Q'$, $Q \equiv Q'$ implies $|\texttt{Atoms}(Q)| \leq |\texttt{Atoms}(Q')|$.
The minimization problem is: given Q, find $Q_{\text{min}} \equiv Q$ s.t. Q_{min} is minimal.

A minimal query is also called a core.

$$Q = E(x, y) \wedge E(y, z) \wedge E(x, u) \wedge E(u, v) \wedge E(v, w)$$



Query Minimization for CQ

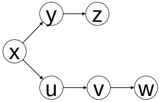
Definition (Minimal Query)

Q is minimal if, $\forall Q'$, $Q \equiv Q'$ implies $|\texttt{Atoms}(Q)| \leq |\texttt{Atoms}(Q')|$.
The minimization problem is: given Q, find $Q_{\text{min}} \equiv Q$ s.t. Q_{min} is minimal.

A minimal query is also called a core.

$$Q = E(x, y) \wedge E(y, z) \wedge E(x, u) \wedge E(u, v) \wedge E(v, w)$$

$$Q_{\text{min}} = E(x, u) \wedge E(u, v) \wedge E(v, w)$$



Properties of Minimal CQs

Let $h : Q \to Q$ be a homomorphism; then $\boxed{Q \equiv Im(h)}$ why????

Properties of Minimal CQs

Let $h : Q \to Q$ be a homomorphism; then $\boxed{Q \equiv Im(h)}$ why????

If Q is minimal, then h is an isomorphism.

Properties of Minimal CQs

Let $h : Q \to Q$ be a homomorphism; then $\boxed{Q \equiv Im(h)}$ why????

If Q is minimal, then h is an isomorphism.

Theorem

If $Q \equiv Q'$ and both are minimal, then they are isomorphic.

Proof: Since $Q \equiv Q'$, $\exists h : Q \to Q'$, $\exists h' : Q' \to Q$.

Properties of Minimal CQs

Let $h : Q \to Q$ be a homomorphism; then $\boxed{Q \equiv Im(h)}$ why????

If Q is minimal, then h is an isomorphism.

Theorem

If $Q \equiv Q'$ and both are minimal, then they are isomorphic.

Proof: Since $Q \equiv Q'$, $\exists h : Q \to Q'$, $\exists h' : Q' \to Q$.

Since Q is minimal, $h' \circ h : Q \to Q$ is an isomorphism.

Properties of Minimal CQs

Let $h : Q \to Q$ be a homomorphism; then $\boxed{Q \equiv Im(h)}$ why????

If Q is minimal, then h is an isomorphism.

Theorem

If $Q \equiv Q'$ and both are minimal, then they are isomorphic.

Proof: Since $Q \equiv Q'$, $\exists h : Q \to Q'$, $\exists h' : Q' \to Q$.

Since Q is minimal, $h' \circ h : Q \to Q$ is an isomorphism.

Then both h, h' are isomorphisms.

Query Minimization Procedure

Let Q be a CQ with m atoms. We compute $Q_{\min} \equiv Q$.

- Remove some atom A from Q.
  Call Q' the resulting query (with $m-1$ atoms).

Query Minimization Procedure

Let Q be a CQ with m atoms. We compute $Q_{\text{min}} \equiv Q$.

- Remove some atom A from Q.
  Call Q' the resulting query (with $m-1$ atoms).
- Observe that $\exists h : Q' \to Q$.

Query Minimization Procedure

Let Q be a CQ with m atoms. We compute $Q_{\min} \equiv Q$.

- Remove some atom A from Q.
  Call Q' the resulting query (with $m-1$ atoms).
- Observe that $\exists h : Q' \to Q$.
- If $\exists h : Q \to Q'$, then $Q \equiv Q'$: replace Q with Q' and repeat.

Query Minimization Procedure

Let Q be a CQ with m atoms. We compute $Q_{\text{min}} \equiv Q$.

- Remove some atom A from Q.
  Call Q' the resulting query (with $m-1$ atoms).
- Observe that $\exists h : Q' \to Q$.
- If $\exists h : Q \to Q'$, then $Q \equiv Q'$: replace Q with Q' and repeat.
- Otherwise, try another atom A.

Query Minimization Procedure

Let Q be a CQ with m atoms. We compute $Q_{\text{min}} \equiv Q$.

- Remove some atom A from Q.
  Call Q' the resulting query (with $m-1$ atoms).
- Observe that $\exists h : Q' \to Q$.
- If $\exists h : Q \to Q'$, then $Q \equiv Q'$: replace Q with Q' and repeat.
- Otherwise, try another atom A.

When no more change, stop and return Q: this is the minimal query equivalent to the original.

Discussion

- For each CQ Q there exists a minimized query equivalent to Q,
- The mimal subquery is unique up to isomorphism.
- It can be found as subquery of Q, using the minimization procedure.
- Statements above fail once we add $\neq$ or $<$ or $\leq$. See HW2.

Acyclic Queries

Background: Natural Joins, Semi-Joins

The join of A, B returns all variables: $\boxed{(A \bowtie B)(x, y, z) = A(x, y) \wedge B(y, z)}$

We can compute[1] $A \bowtie B$ in time $\tilde{O}(|A| + |B| + |A \bowtie B|)$

[1] $\tilde{O}$ means a log-factor, in order to sort A, B.

Background: Natural Joins, Semi-Joins

The join of A, B returns all variables: $(A \bowtie B)(x, y, z) = A(x, y) \wedge B(y, z)$

We can compute[1] $A \bowtie B$ in time $\tilde{O}(|A| + |B| + |A \bowtie B|)$

The semi-join returns only A's vars: $(A \ltimes B)(x, y) = A(x, y) \wedge B(y, z)$

We can compute $A \ltimes B$ in time $\tilde{O}(|A|)$ and $|A \ltimes B| \leq |A \bowtie B|$.

[1] $\tilde{O}$ means a log-factor, in order to sort A, B.

Problem Statement

Compute $Q(\boldsymbol{D})$, where Q is a Boolean CQ or a Full[2] CQ:

$$\begin{aligned} & Q_{\text{bool}}() = A_1(\boldsymbol{x}_1) \wedge A_2(\boldsymbol{x}_2) \wedge \cdots \\ \text{or} \quad & Q_{\text{full}}(\boldsymbol{x}) = A_1(\boldsymbol{x}_1) \wedge A_2(\boldsymbol{x}_2) \wedge \cdots \end{aligned}$$

[2]Full CQ: means all variables are head variables

Problem Statement

Compute $Q(\boldsymbol{D})$, where Q is a Boolean CQ or a Full[2] CQ:

$$\begin{aligned} & Q_{\text{bool}}() = A_1(\boldsymbol{x}_1) \wedge A_2(\boldsymbol{x}_2) \wedge \cdots \\ \text{or} \quad & Q_{\text{full}}(\boldsymbol{x}) = A_1(\boldsymbol{x}_1) \wedge A_2(\boldsymbol{x}_2) \wedge \cdots \end{aligned}$$

Approach 1: loop over each variable. Time = $O(|\text{Dom}(\boldsymbol{D})|^{|\text{Vars}(Q)|})$.

[2] Full CQ: means all variables are head variables

Problem Statement

Compute $Q(\boldsymbol{D})$, where Q is a Boolean CQ or a Full[2] CQ:

$$\begin{aligned} & Q_{\text{bool}}() = A_1(\boldsymbol{x}_1) \wedge A_2(\boldsymbol{x}_2) \wedge \cdots \\ \text{or} \quad & Q_{\text{full}}(\boldsymbol{x}) = A_1(\boldsymbol{x}_1) \wedge A_2(\boldsymbol{x}_2) \wedge \cdots \end{aligned}$$

Approach 1: loop over each variable. Time = $O(|\text{Dom}(\boldsymbol{D})|^{|\text{Vars}(Q)|})$.

Approach 2: $((A_1 \bowtie A_2) \bowtie A_3) \bowtie A_4 \ldots$. Time = $\tilde{O}(\sum_i |A_i| + \sum_i |A_1 \bowtie \cdots A_i|)$

[2] Full CQ: means all variables are head variables

Problem Statement

Compute $Q(\boldsymbol{D})$, where Q is a Boolean CQ or a Full[2] CQ:

$$
\begin{aligned}
& Q_{\text{bool}}() = A_1(\boldsymbol{x}_1) \wedge A_2(\boldsymbol{x}_2) \wedge \cdots \\
\text{or} \quad & Q_{\text{full}}(\boldsymbol{x}) = A_1(\boldsymbol{x}_1) \wedge A_2(\boldsymbol{x}_2) \wedge \cdots
\end{aligned}
$$

Approach 1: loop over each variable. Time = $O(|\text{Dom}(\boldsymbol{D})|^{|\text{Vars}(Q)|})$.

Approach 2: $((A_1 \bowtie A_2) \bowtie A_3) \bowtie A_4 \ldots$. Time = $\tilde{O}(\sum_i |A_i| + \sum_i |A_1 \bowtie \cdots A_i|)$

When Q is acyclic, then we can compute $Q(\boldsymbol{D})$ in time $\boxed{\tilde{O}(|\boldsymbol{D}| + |Q(\boldsymbol{D})|)}$

[2] Full CQ: means all variables are head variables

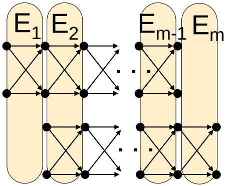
Why Linear Time is Difficult

Example: $Q(x_0, x_1, \ldots, x_m) = E_1(x_0, x_1) \wedge E_2(x_1, x_2) \wedge \cdots \wedge E_m(x_{m-1}, x_m)$

Why Linear Time is Difficult

Example: $Q(x_0, x_1, \ldots, x_m) = E_1(x_0, x_1) \wedge E_2(x_1, x_2) \wedge \cdots \wedge E_m(x_{m-1}, x_m)$

$|E_1| = 4$, $|E_2| = \cdots = |E_{m-1}| = 8$, $|E_m| = 4$.



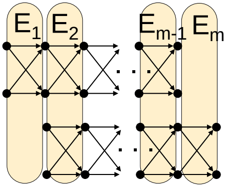
Why Linear Time is Difficult

Example: $Q(x_0, x_1, \ldots, x_m) = E_1(x_0, x_1) \wedge E_2(x_1, x_2) \wedge \cdots \wedge E_m(x_{m-1}, x_m)$

$|E_1| = 4,\ |E_2| = \cdots = |E_{m-1}| = 8,\ |E_m| = 4.$

$|\boldsymbol{D}| = O(m),\ Q(\boldsymbol{D}) = \varnothing$

$|E_1 \bowtie \cdots \bowtie E_{m-1}| = 2^{m+1} + 2^m$



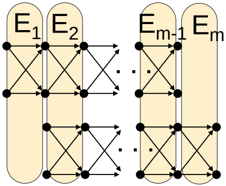
Why Linear Time is Difficult

Example: $Q(x_0, x_1, \ldots, x_m) = E_1(x_0, x_1) \wedge E_2(x_1, x_2) \wedge \cdots \wedge E_m(x_{m-1}, x_m)$

$|E_1| = 4,\ |E_2| = \cdots = |E_{m-1}| = 8,\ |E_m| = 4.$

$|\boldsymbol{D}| = O(m),\ Q(\boldsymbol{D}) = \emptyset$

$|E_1 \bowtie \cdots \bowtie E_{m-1}| = 2^{m+1} + 2^m$



Any join order will exceed the time $\tilde{O}(|\boldsymbol{D}| + |Q(\boldsymbol{D})|)$

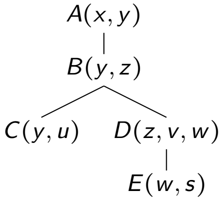
Acyclic CQ

A join tree is a tree T whose nodes are the atoms of Q, which satisfies the running intersection property: for any variable x, the set of nodes that contain x forms a connected component.

Definition

Q is acyclic if it admits a join tree T.

Acyclic: $Q = A(x,y) \wedge B(y,z) \wedge C(y,u) \wedge D(z,v,w) \wedge E(w,s)$



Acyclic CQ

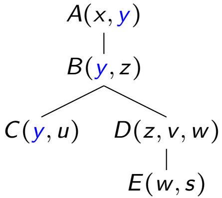
A join tree is a tree T whose nodes are the atoms of Q, which satisfies the running intersection property: for any variable x, the set of nodes that contain x forms a connected component.

Definition

Q is acyclic if it admits a join tree T.

Acyclic: $Q = A(x, y) \wedge B(y, z) \wedge C(y, u)$
$\wedge D(z, v, w) \wedge E(w, s)$

E.g. running intersection for y



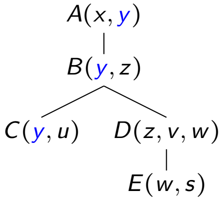
Acyclic CQ

A join tree is a tree T whose nodes are the atoms of Q, which satisfies the running intersection property: for any variable x, the set of nodes that contain x forms a connected component.

Definition

Q is acyclic if it admits a join tree T.

Acyclic: $Q = A(x, y) \wedge B(y, z) \wedge C(y, u) \wedge D(z, v, w) \wedge E(w, s)$



E.g. running intersection for y

Not acyclic: $A(x, y) \wedge B(y, z) \wedge C(z, x)$. why?

Yannakakis' Algorithm for Acyclic CQ Q (Boolean or Full)

Step 1: Bottom-up Semi-join Reduction

$$D := D \ltimes E$$

$$B := B \ltimes C$$

$$B := B \ltimes D$$

$$A := A \ltimes B \qquad \text{if } Q \text{ is Boolean, return } A$$

Yannakakis' Algorithm for Acyclic CQ Q (Boolean or Full)

Step 1: Bottom-up Semi-join Reduction

$D := D \ltimes E$

$B := B \ltimes C$

$B := B \ltimes D$

$A := A \ltimes B$ if Q is Boolean, return A

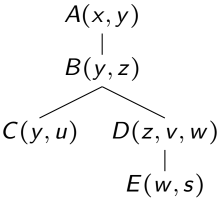
Step 2: Top-down Join Computation:

$T_1 := A \bowtie B$

$T_2 := T_1 \bowtie C$

$T_3 := T_2 \bowtie D$

$T_4 := T_3 \bowtie E$ if Q is Full, return T_4



Yannakakis' Algorithm for Acyclic CQ Q (Boolean or Full)

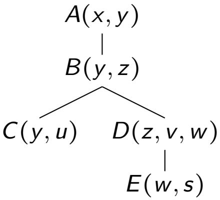
Step 1: Bottom-up Semi-join Reduction

$D := D \ltimes E$

$B := B \ltimes C$

$B := B \ltimes D$

$A := A \ltimes B$ if Q is Boolean, return A



Step 2: Top-down Join Computation:

$T_1 := A \bowtie B$

$T_2 := T_1 \bowtie C$

$T_3 := T_2 \bowtie D$

$T_4 := T_3 \bowtie E$ if Q is Full, return T_4

$$\boxed{\text{Time} = O(|\text{Input}| + |\text{Output}|)}$$

Proof of the Algorithm Using Four Identities

$\boxed{(A \bowtie B)(x, y, z) = A(x, y) \wedge B(y, z)}$, $\boxed{(A \ltimes B)(x, y) = A(x, y) \wedge B(y, z)}$

(1) $A \bowtie B = (A \ltimes B) \bowtie B$. Step 1 does not change Q's output.

Proof of the Algorithm Using Four Identities

$\boxed{(A \bowtie B)(x,y,z) = A(x,y) \wedge B(y,z)}$, $\boxed{(A \ltimes B)(x,y) = A(x,y) \wedge B(y,z)}$

(1) $A \bowtie B = (A \ltimes B) \bowtie B$. Step 1 does not change Q's output.

Proof: follows from $Q_1 \equiv Q_2$, where:

$$Q_1(x,y,z) = A(x,y) \wedge B(y,z)$$
$$Q_2(x,y,z) = A(x,y) \wedge B(y,u) \wedge B(y,z)$$

In class: find homomorphisms $Q_2 \to Q_1$ and $Q_1 \to Q_2$.

Proof of the Algorithm Using Four Identities

$\boxed{(A \bowtie B)(x,y,z) = A(x,y) \wedge B(y,z)}$, $\boxed{(A \ltimes B)(x,y) = A(x,y) \wedge B(y,z)}$

(1) $A \bowtie B = (A \ltimes B) \bowtie B$. Step 1 does not change Q's output.

(2) $A \ltimes B = \Pi_{x,y}(A \bowtie B)$. Step 1 returns correct answer for Boolean Q
$A \bowtie B = \varnothing$ iff $A \ltimes B = \varnothing$.

Proof of the Algorithm Using Four Identities

$\boxed{(A \bowtie B)(x,y,z) = A(x,y) \wedge B(y,z)}$, $\boxed{(A \ltimes B)(x,y) = A(x,y) \wedge B(y,z)}$

(1) $A \bowtie B = (A \ltimes B) \bowtie B$. Step 1 does not change Q's output.

(2) $A \ltimes B = \Pi_{x,y}(A \bowtie B)$. Step 1 returns correct answer for Boolean Q
$A \bowtie B = \varnothing$ iff $A \ltimes B = \varnothing$.

Proof: immediate from the definition

Proof of the Algorithm Using Four Identities

$\boxed{(A \bowtie B)(x,y,z) = A(x,y) \wedge B(y,z)}$, $\boxed{(A \ltimes B)(x,y) = A(x,y) \wedge B(y,z)}$

(1) $A \bowtie B = (A \ltimes B) \bowtie B$. Step 1 does not change Q's output.

(2) $A \ltimes B = \Pi_{x,y}(A \bowtie B)$. Step 1 returns correct answer for Boolean Q
$A \bowtie B = \emptyset$ iff $A \ltimes B = \emptyset$.

(3) $A \ltimes (B \ltimes C) = A \ltimes (B \bowtie C)$, when $\text{Vars}(A) \cap \text{Vars}(C) = \emptyset$:
Step 1 fully reduces each relation: $A := A \ltimes (B \bowtie C \bowtie \cdots)$

Proof of the Algorithm Using Four Identities

$\boxed{(A \bowtie B)(x,y,z) = A(x,y) \wedge B(y,z)}$, $\boxed{(A \ltimes B)(x,y) = A(x,y) \wedge B(y,z)}$

(1) $A \bowtie B = (A \ltimes B) \bowtie B$. Step 1 does not change Q's output.

(2) $A \ltimes B = \Pi_{x,y}(A \bowtie B)$. Step 1 returns correct answer for Boolean Q

$A \bowtie B = \emptyset$ iff $A \ltimes B = \emptyset$.

(3) $A \ltimes (B \ltimes C) = A \ltimes (B \bowtie C)$, when $\text{Vars}(A) \cap \text{Vars}(C) = \emptyset$:

Step 1 fully reduces each relation: $A := A \ltimes (B \bowtie C \bowtie \cdots)$

Proof: both sides are the same query

$$Q_1(x,y) = A(x,y) \wedge B(y,z) \wedge C(z,u)$$
$$Q_2(x,y) = A(x,y) \wedge B(y,z) \wedge C(z,u)$$

Proof of the Algorithm Using Four Identities

$\boxed{(A \bowtie B)(x,y,z) = A(x,y) \wedge B(y,z)}$, $\boxed{(A \ltimes B)(x,y) = A(x,y) \wedge B(y,z)}$

(1) $A \bowtie B = (A \ltimes B) \bowtie B$. Step 1 does not change Q's output.

(2) $A \ltimes B = \Pi_{x,y}(A \bowtie B)$. Step 1 returns correct answer for Boolean Q
$A \bowtie B = \varnothing$ iff $A \ltimes B = \varnothing$.

(3) $A \ltimes (B \ltimes C) = A \ltimes (B \bowtie C)$, when $\text{Vars}(A) \cap \text{Vars}(C) = \varnothing$:
Step 1 fully reduces each relation: $A := A \ltimes (B \bowtie C \bowtie \cdots)$

(4) $A \bowtie (B \ltimes C) = (A \bowtie B) \ltimes C$ Step 2 never exceed final output size:
$|A \bowtie (B \ltimes C)| = |(A \bowtie B) \ltimes C| \leq |A \bowtie B \bowtie C|$

Proof: both sides are the same query (as before)

Yannakakis Algorithm for General CQ

$$Q(x_1, \ldots, x_p) = \exists x_{p+1} \cdots \exists x_k (A_1 \wedge \cdots \wedge A_m)$$

Definition

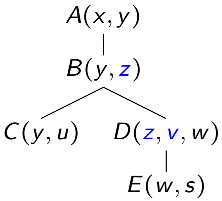
Q is acyclic free-connex if it is acyclic after we add atom $\text{Out}(x_1, \ldots, x_p)$.

If Q is acyclic free-connex, it can be computed in time $O(|\text{Input}| + |\text{Output}|)$.
Otherwise, it cannot[3]

[3]Based on fined-grained complexity assumptions.

Example of a Free-Connex Query

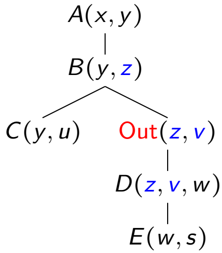
$Q(z, v) =$



Where do we place
Out(z, v)?

Example of a Free-Connex Query

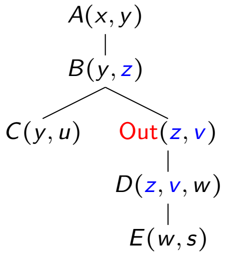
$Q(z, v) =$



Where do we place $\text{Out}(z, v)$?

Example of a Free-Connex Query

$Q(z, v) =$

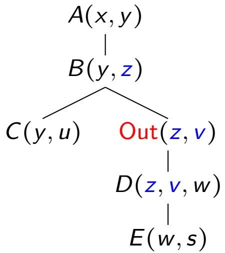


Semijoin Reduction

As before.

Example of a Free-Connex Query

$Q(z, v) =$



Join Computation

$$
\begin{aligned}
T_1(y) &:= A(x, y) \\
T_2(y, z) &:= T_1(y) \bowtie B(y, z) \\
T_3(y) &:= C(y, u) \\
T_4(z) &:= T_2(y, z) \bowtie T_3(y) \\
T_5(w) &:= E(w, s) \\
T_6(z, v) &:= T_5(w) \bowtie D(z, v, w) \\
T_7(z, v) &:= T_6(z, v) \bowtie T_4(z)
\end{aligned}
$$

Return $T_7(z, v)$.

Semijoin Reduction

As before.

The last node in the join is the leaf $\text{Out}(z, v)$, which we don't need to join.

Summary

- Yannakakis' algorithm: Semijoin reduction (up, then down), then joins.
  - Requires the query to be acyclic.
  - Works for full CQs, for Boolean CQs, and for "free-connext" CQs.
  - Related to the Junction-tree Algorithm in graphical models.
- Most SQL queries in practice are acyclic.
- Discussion in class Do database engines run Yannakakis algorithm? If not, why not?

Hypertree Decomposition

Outline

We the query is cyclic, then we compute a tree decomposition and (1) evaluate each node of the tree into a temporary table, (2) run Yannakakis' algorithm on the temporary results.

Background: Tree Decomposition

Fix an undirected graph $G = (V, E)$.

Background: Tree Decomposition

Fix an undirected graph $G = (V, E)$.

A tree decomposition is (T, χ), where T is a tree, $\chi : \text{Nodes}(T) \to 2^V$:

- Running intersection: $\forall x \in V$, $\{n \in \text{Nodes}(T) \mid x \in \chi(n)\}$ is connected.
- For every edge $(x, y) \in E$, $\exists n \in \text{Nodes}(T)$ s.t. $x, y \in \chi(n)$.

Background: Tree Decomposition

Fix an undirected graph $G = (V, E)$.

A tree decomposition is (T, χ), where T is a tree, $\chi : \text{Nodes}(T) \rightarrow 2^V$:

- Running intersection: $\forall x \in V$, $\{n \in \text{Nodes}(T) \mid x \in \chi(n)\}$ is connected.
- For every edge $(x, y) \in E$, $\exists n \in \text{Nodes}(T)$ s.t. $x, y \in \chi(n)$.

Tree width: $tw(T) \stackrel{\text{def}}{=} \max_n |\chi(n)| - 1$ $tw(G) \stackrel{\text{def}}{=} \min_T tw(T)$.

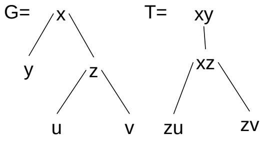
Background: Tree Decomposition

Fix an undirected graph $G = (V, E)$.

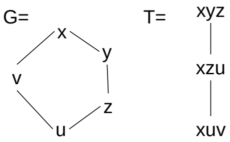
A tree decomposition is (T, χ), where T is a tree, $\chi : \text{Nodes}(T) \to 2^V$:

- Running intersection: $\forall x \in V$, $\{n \in \text{Nodes}(T) \mid x \in \chi(n)\}$ is connected.
- For every edge $(x, y) \in E$, $\exists n \in \text{Nodes}(T)$ s.t. $x, y \in \chi(n)$.

Tree width: $tw(T) \stackrel{\text{def}}{=} \max_n |\chi(n)| - 1$ $\quad tw(G) \stackrel{\text{def}}{=} \min_T tw(T)$.



tw(G) = 1



tw(G) = 2

Discussion

- Tree decomposition of graphs is widely used in graph theory.
- $\chi(n)$ is called a bag.
- If G is a tree, then $tw(G) = 1$.
- If K_n is the clique with n nodes, then $tw(K_n) = n$.
- If $K_{m,n}$ is the complete bipartite graph with m, n nodes, then $tw(K_{m,n}) = \min(m, n)$.
- HW2: compute tree-width of an $m \times n$ grid.

Hypertree Decomposition

Definition

A hypertree decomposition of a query (hypergraph) Q is (T, χ) where T is a tree and $\chi : \text{Nodes}(T) \to 2^{\text{Vars}(Q)}$ such that:

- Running intersection property: $\forall x \in \text{Vars}(Q)$, the set $\{n \in \text{Nodes}(T) \mid x \in \chi(n)\}$ is connected.
- Every atom $R_i(\boldsymbol{x}_i)$ is covered: $\exists n \in \text{Nodes}(T)$ s.t. $\boldsymbol{x}_i \subseteq \chi(n)$

$$Q = R(x, y) \wedge S(y, z) \wedge T(z, u) \wedge K(u, x)$$

$T =$

xyz
|
xuz

Edge Cover

In a graph, an edge cover is a set of edges that includes all nodes.

Edge Cover

In a graph, an edge cover is a set of edges that includes all nodes.

An edge cover of a query Q is a set of atoms $\mathcal{C}$ that includes all variables.
Its edge cover number is $\rho(Q) = \min_{\mathcal{C}} |\mathcal{C}|$.

Compute Q (1) join relations in $\mathcal{C}$ (2) semi-join the rest. Time $\tilde{O}(|\boldsymbol{D}|^{\rho(Q)})$

Edge Cover

In a graph, an edge cover is a set of edges that includes all nodes.

An edge cover of a query Q is a set of atoms $\mathcal{C}$ that includes all variables.
Its edge cover number is $\rho(Q) = \min_{\mathcal{C}} |\mathcal{C}|$.

Compute Q (1) join relations in $\mathcal{C}$ (2) semi-join the rest. Time $\tilde{O}(|\boldsymbol{D}|^{\rho(Q)})$

E.g. $Q(x,y,z) = R(x,y) \wedge S(y,z) \wedge T(z,x)$ edge cover $\mathcal{C} = \{R, S\}$.
Compute: $J(x,y,z) := R(x,y) \bowtie S(y,z)$ $Q(x,y,z) := J(x,y,z) \ltimes T(z,x)$

Hypertree Width

For a subset of variables $\boldsymbol{z} \subseteq \text{Vars}(Q)$ is $\rho(\boldsymbol{z})$ is the edge cover number of Q restricted to $\boldsymbol{z}$.

[4]Warning: sometimes called generalized hypertree width.

Hypertree Width

For a subset of variables $\boldsymbol{z} \subseteq \text{Vars}(Q)$ is $\rho(\boldsymbol{z})$ is the edge cover number of Q restricted to $\boldsymbol{z}$.

Hypertree width:[4] $\text{HTW}(T) \stackrel{\text{def}}{=} \max_n \rho(\chi(n))$ $\text{HTW}(Q) \stackrel{\text{def}}{=} \min_T \text{HTW}(T)$

What is $\text{HTW}(Q)$?

$Q = R(x,y) \wedge S(y,z) \wedge T(z,u) \wedge K(u,x)$

xyz
|
xuz

[4]Warning: sometimes called generalized hypertree width.

Summary: Computing Q Using Tree Decomposition

Assume Q is a full conjunctive query:

- Find a tree decomposition with minimum $\text{HTW}(T)$.
- Compute every bag using a left-deep join plan $(R_1 \bowtie R_2) \bowtie \cdots$ and materialize it.
- Run Yannakakis' algorithm on the result.
- Runtime: $\tilde{O}(|\boldsymbol{D}|^{\text{HTW}(Q)})$.