Join Queries with Negation (and Aggregation)
This Talk

Much progress has been over the last years on faster join algorithms
• worst-case optimal joins
• constant-delay enumeration
• tree decompositions & width measures
• PANDA

What happens when we add negation (and aggregation)?
Conjunctive Queries (CQs)

\[ Q(x_F) = \bigwedge_{K \in \mathcal{E}} R_K(x_K) \]

- variables \( x = \{x_1, \ldots, x_n\} \)
- hypergraph \([n], \mathcal{E}\)
- for a hyperedge \( E \subseteq [n] : x_E = \{x_i\}_{i \in E} \)

- Boolean: \( F = \emptyset \)
- full: \( F = [n] = \{1,2,\ldots,n\} \)
Example: Triangle

\[ Q(x_1, x_3) = R(x_1, x_2) \land S(x_2, x_3) \land T(x_1, x_3) \]
Conjunctive Queries + Negation (CQNs)

\[ Q(x_F) = \bigwedge_{K \in \mathcal{E}^+} R_K(x_K) \land \bigwedge_{K \in \mathcal{E}^-} \neg R_K(x_K) \]

- We need a safety condition: the positive atoms must contain all variables
- The hypergraph \([n, \mathcal{E}^+, \mathcal{E}^-]\) is called the signed hypergraph
Example: Open Triangle

\[ Q(x_2) = R(x_1, x_2) \land S(x_2, x_3) \land \neg T(x_1, x_3) \]
Example: 3-independent set

\[ Q() = V(x_1) \land V(x_2) \land V(x_3) \land \neg R(x_1, x_2) \land \neg R(x_2, x_3) \land \neg R(x_1, x_3) \]

If all positive relations are singleton, we will sometimes ignore them and just write

\[ Q() = \neg R(x_1, x_2) \land \neg R(x_2, x_3) \land \neg R(x_1, x_3) \]
Some Background
A CQ is $\alpha$-acyclic if and only if it admits a join tree

$$Q() = R(x_1, x_2) \land S(x_2, x_3) \land T(x_3, x_4, x_5) \land U(x_3, x_6)$$
The structure of $\alpha$-acyclicity

A node $v$ is an $\alpha$-leaf if the set $\{K \in \mathcal{E} \mid v \in K\}$ contains a maximum element (pivot)

$$Q() = R(x_1, x_2) \land S(x_2, x_3) \land T(x_3, x_1) \land U(x_1, x_2, x_3)$$

$$x_1 : R(x_1, x_2) \land S(x_2, x_3) \land T(x_3, x_1) \land U(x_1, x_2, x_3)$$

$$x_2 : R(x_1, x_2) \land S(x_2, x_3) \land T(x_3, x_1) \land U(x_1, x_2, x_3)$$

$$x_3 : R(x_1, x_2) \land S(x_2, x_3) \land T(x_3, x_1) \land U(x_1, x_2, x_3)$$

All variables are $\alpha$-leaves for this hypergraph!
The structure of $\alpha$-acyclicity

A CQ is $\alpha$-acyclic iff it admits an $\alpha$-elimination sequence. At every step:
1. find any $\alpha$-leaf $x$ (with pivot $R$)
2. remove any relation with variables contained in $R$
3. remove $x$ from $R$

$$Q() = R(x_1, x_2) \land S(x_2, x_3) \land T(x_3, x_1) \land U(x_1, x_2, x_3)$$

$x_1$ is an $\alpha$-leaf: $R(x_1, x_2) \land S(x_2, x_3) \land T(x_3, x_1) \land U(x_1, x_2, x_3)$

$x_2$ is an $\alpha$-leaf: $S(x_2, x_3) \land U(x_2, x_3)$

$x_3$ is an $\alpha$-leaf: $U(x_3)$
A linear-time algorithm

We follow the $\alpha$-elimination sequence. At every step:

1. find any $\alpha$-leaf $x$ (with pivot $R$)
2. for any $T$ with variables contained in $R$, update $R \leftarrow R \bowtie T$ and remove $T$
3. project out $x$ from $R$
A linear-time algorithm

\[ Q() = R(x_1, x_2) \land S(x_2, x_3) \land T(x_3, x_1) \land U(x_1, x_2, x_3) \]

\( x_1 \) is an \( \alpha \)-leaf: \( R(x_1, x_2) \land S(x_2, x_3) \land T(x_3, x_1) \land U(x_1, x_2, x_3) \)
\( x_2 \) is an \( \alpha \)-leaf: \( S(x_2, x_3) \land U(x_2, x_3) \)
\( x_3 \) is an \( \alpha \)-leaf: \( U(x_3) \)
A linear-time characterization for CQs

[Yannakakis ’81] For an $\alpha$-acyclic CQ with input size $N$:

1. if it is **Boolean**, it can be evaluated in linear time $O(N)$

2. if it is **full**, the output can be enumerated with constant delay after linear-time preprocessing, with total time $O(N + \text{OUT})$

3. if it is **full**, we can count the answers in linear time $O(N)$

Moreover, no other CQs admit linear-time algorithms under widely believed conjectures
What is the linear-time characterization for CQs with negation?
The Inclusion-Exclusion Principle

\[ Q(x_1, x_2, x_3) = R(x_1, x_2) \land S(x_2, x_3) \land \neg T(x_1, x_2, x_3) \]

We can rewrite this query using a difference operator:

\[ Q = (R(x_1, x_2) \land S(x_2, x_3)) - (R(x_1, x_2) \land S(x_2, x_3) \land T(x_1, x_2, x_3)) \]

\( Q_1: \text{acyclic CQ} \quad Q_2: \text{acyclic CQ} \)

\[ \#Q = \#Q_1 - \#Q_2 \]
The Inclusion-Exclusion Principle

\[ Q(x) = \bigwedge_{K \in \mathcal{E}^+} R_K(x_K) \land \bigwedge_{K \in \mathcal{E}^-} \neg R_K(x_K) \]

We can generalize this idea via the inclusion-exclusion principle [Brault-Baron '13]:

\[ \#Q = \sum_{S \subseteq \mathcal{E}^-} (-1)^{|S|} \#Q_S \]

where \( Q_S \) is the CQ with hypergraph \(([n], \mathcal{E}^+ \cup S)\)
Signed-acyclicity

If the hypergraph $\mathcal{E}^+ \cup S$ is $\alpha$-acyclic for any $S \subseteq \mathcal{E}^-$ then $\#Q$ (and thus Boolean $Q$) can be evaluated in linear time (data complexity)

- Caveat #1: the algorithm is *exponential* in the size of the query
- Caveat #2: we cannot use this idea to perform *constant-delay enumeration*

A CQ with negation is **signed-acyclic** if $\mathcal{E}^+ \cup S$ is $\alpha$-acyclic for any $S \subseteq \mathcal{E}^-$

[Brault-Baron ’13]
Signed-acyclicity: examples

\[ Q() = R(x_1, x_2) \land S(x_2, x_3) \land T(x_3, x_1) \land U(x_1, x_2, x_3) \]

\[ Q() = R(x_1, x_2) \land S(x_2, x_3) \land T(x_3, x_1) \land \neg U(x_1, x_2, x_3) \]

\[ Q() = \neg R(x_1, x_2) \land \neg S(x_2, x_3) \land \neg T(x_3, x_1) \land \neg U(x_1, x_2, x_3) \]

\[ Q() = \neg R(x_1, x_2) \land \neg S(x_2, x_3) \land \neg T(x_3, x_4) \]
β-acyclicity

- Suppose all positive relations are unary (arity = 1)
- Then signed-acyclicity is equivalent to: any subset of $\mathcal{E}^-$ is $\alpha$-acyclic
- This is equivalent to the notion of β-acyclicity [Duris ’12, Brault-Baron ’14]
- Existing algorithms for β-acyclic CQNs include polylogarithmic factors
A Linear-Time Algorithm
The structure of signed-acyclicity

A node $v$ is a **signed-leaf** if there exists $K \in \mathcal{G}^+$ (pivot) such that:

- **$\alpha$-property**: every positive edge that contains $v$ is contained in $K$
- **$\beta$-property**: $\{ N \in \mathcal{G}^- \mid v \in N, N \subset K \} \cup \{ K \}$ forms a total order w.r.t. inclusion with $K$ as the smallest element

$$Q() = A(x_1, x_2, x_3) \land U(x_3, x_4) \land \neg V(x_4) \land \neg R(x_2, x_3, x_4) \land \neg S(x_1, x_2, x_3, x_4)$$

*pivot for* $x_4$
The structure of signed-acyclicity

A CQ is signed-acyclic \textbf{iff} it admits a signed-elimination sequence. At every step:

1. find any signed-leaf $x$ (with pivot $R$)
2. remove any relation with variables contained in $R$ ($\alpha$-property)
3. remove $x$ from everywhere ($\beta$-property)

$$Q() = A(x_1, x_2, x_3) \land U(x_3, x_4) \land \neg V(x_4) \land \neg R(x_2, x_3, x_4) \land \neg S(x_1, x_2, x_3, x_4)$$
A linear-time algorithm

We follow the signed-elimination sequence. At every step:

1. find any signed-leaf $x$ (with pivot $R$)
2. Semi-join with $R$ and remove any relation with variables contained in $R$ ($\alpha$-property)
3. “Remove” $x$ from every relation that contains it ($\beta$-property)

Item #3 is the challenging one!
The Key Idea

\[ Q() = A(x_1) \land B(x_2) \land \neg R(x_1, x_2) \]

- We cannot afford to scan A for every value of B
- We build a data structure that encodes the “skips”
The Key Idea

\[ Q() = A(x_1) \land B(x_2) \land \neg R(x_1, x_2) \]

To “project out” \( x_1 \) from \( R \), we only keep the values that generate no answer (i.e. \( \{d\} \))

\[ Q'(()) = B(x_2) \land \neg R(x_2) \]
The skipping data structure can also be used to enumerate all results with constant delay.
A linear-time characterization

For a signed-acyclic CQ with negation with input size $N$:

1. If it is **Boolean**, it can be evaluated in linear time $O(N)$

2. If it is **full**, the answers can be enumerated with constant delay after linear-time preprocessing, with total time $O(N + \text{OUT})$

Moreover, the algorithms have polynomial combined complexity
What about projections?

\[ Q(x_F) = \bigwedge_{K \in \mathcal{E}^+} R_K(x_K) \wedge \bigwedge_{K \in \mathcal{E}^-} \neg R_K(x_K) \]

If the signed hypergraph \([n, \mathcal{E}^+, \mathcal{E}^- \cup \{F\})\) is signed-acyclic, the output can be enumerated with constant delay after linear-time preprocessing

- This naturally captures the notion of free-connex CQs
- Any CQN not in this class does not admit a linear-time algorithm under widely believed conjectures
Aggregation
Counting: example

\[ #Q(x_2) = A(x_1) \land B(x_2) \land \neg R(x_1, x_2) \]

For every value of B, count the number of nodes from A that are not connected with it.
Counting: example

\[ \#Q(x_2) = A(x_1) \land B(x_2) \land \neg R(x_1, x_2) \]

We can count by using the skipping DS to find the correct intervals and then compute the partial counts:

- \( a : [1 - 5] \)
- \( b : [3] \)
- \( c : [1 - 2], [4 - 5] \)
We need to compute the partial sums

We can build a data structure in linear time such that we can calculate each partial sum in constant time (OFFLINE PARTIAL SUMS)

Idea: \[ \sum_{i=u}^{v} x_i = \sum_{i=1}^{v} x_i - \sum_{i=1}^{v} x_i \]
For any aggregation, where $\oplus$ forms a semigroup

- we can compute the partial sums in constant time
- but we need preprocessing time $O(N \cdot \alpha(N))$
- $\alpha(N)$ is the inverse Ackermann function
- uses deep results for RangeSum [Yao ‘82, Chazelle ‘91]
Aggregation in Arbitrary CQNs

\[ Q(x_F) = \bigoplus_{x[n] \neq F} \bigotimes_{K \in \mathcal{E}^+} R_K(x_K) \land \bigotimes_{K \in \mathcal{E}^-} \bar{R}_K(x_K) \]

- **Semiring structure** \((D, \oplus, \otimes, 0, 1)\)
- **positive factor**: a list of tuples with their value in \(D\); any tuple outside the list has value 0
- **negative factor**: a list of tuples with their value in \(D\); any tuple outside the list has the same default value \(c \neq 0\)
Aggregation in Arbitrary CQNs

\[ Q(x_F) = \bigoplus_{x_{[n]\setminus F}} \bigotimes_{K \in \mathcal{G}^+} R_K(x_K) \land \bigotimes_{K \in \mathcal{G}^-} \bar{R}_K(x_K) \]

For any semiring, if the signed hypergraph \([n], \mathcal{G}^+, \mathcal{G}^- \cup \{F\}\) is free-connex signed-acyclic, the output can be enumerated with constant delay after preprocessing time \(O(N \cdot \alpha(N))\)

- If the semiring has an additive inverse, the preprocessing time is \(O(N)\)
- The general algorithm follows the elimination sequence, but maintaining the aggregates becomes very complex
Other Remarks
Query Difference

Our techniques also characterize the linear-time behavior for the difference of two CQs with the same output schema: $Q = Q_1 - Q_2$ [Hu & Wang ’23]

$Q = (R(x_1, x_2) \land S(x_2, x_3, x_4)) - (T(x_1, x_2) \land U(x_2, x_3))$

$= (R(x_1, x_2) \land S(x_2, x_3, x_4) \land \neg T(x_1, x_2)) \cup (R(x_1, x_2) \land S(x_2, x_3, x_4) \land \neg U(x_2, x_3))$

Since both resulting CQNs are signed-acyclic, we can enumerate their union with constant-delay enumeration after linear time preprocessing.
Relational Division

- Suppose we want to compute relational division: \( R(x, y)/S(x) \)
- We can rewrite using RA: \( \pi_y(R) - \pi_y((\pi_y(R) \times S) - R) \)
- Define \( R'(y) = \pi_y(R) \), which can be computed in linear time
- The RHS of the difference is the query \( Q(y) = R'(y) \land S(x) \land \neg R(x, y) \) which is free-connex signed-acyclic and thus can be computed in linear time!

Corollary: the division operator can be computed in linear time
Open Questions

What are the appropriate measures of width to have tractability for CQNs?
- nest-set width [Lanzinger ’21]
- generalizations of fractional hyper tree width?

Do our algorithms translate to practice?
- query rewriting techniques [Hu & Wang ’23]
- data structure implementation