Join Queries with Negation (and Aggregation)

Conjunctive Queries with Negation and Aggregation: A Linear Time Characterization [PODS'24]

Hangdong Zhao, Austen Fan, Xiating Ouyang, Paraschos Koutris

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

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)

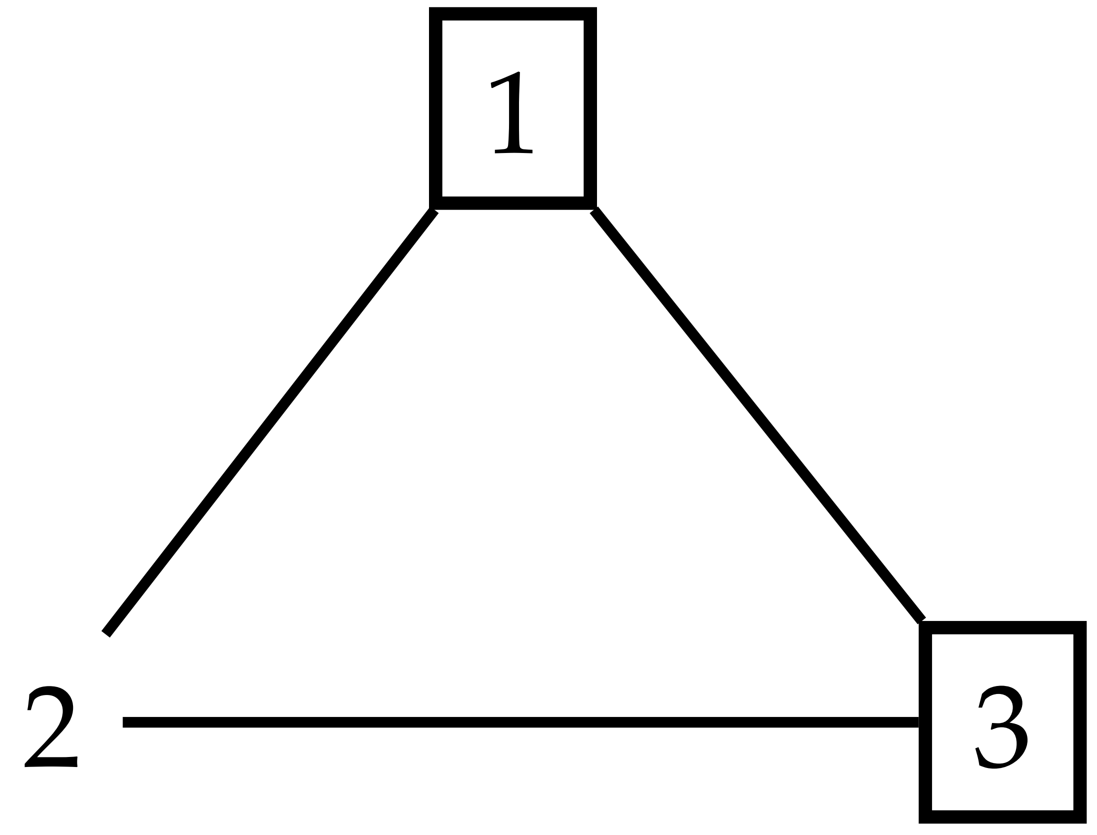
$$\overset{\textit{head}}{Q(\mathbf{x}_F)} = \overset{\textit{body}}{\bigwedge_{K \in \mathscr{E}} R_K(\mathbf{x}_K)}$$

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

- Boolean: $F = \emptyset$
- full: $F = [n] = \{1,2,\ldots,n\}$

Example: Triangle

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



Conjunctive Queries + Negation (CQNs)

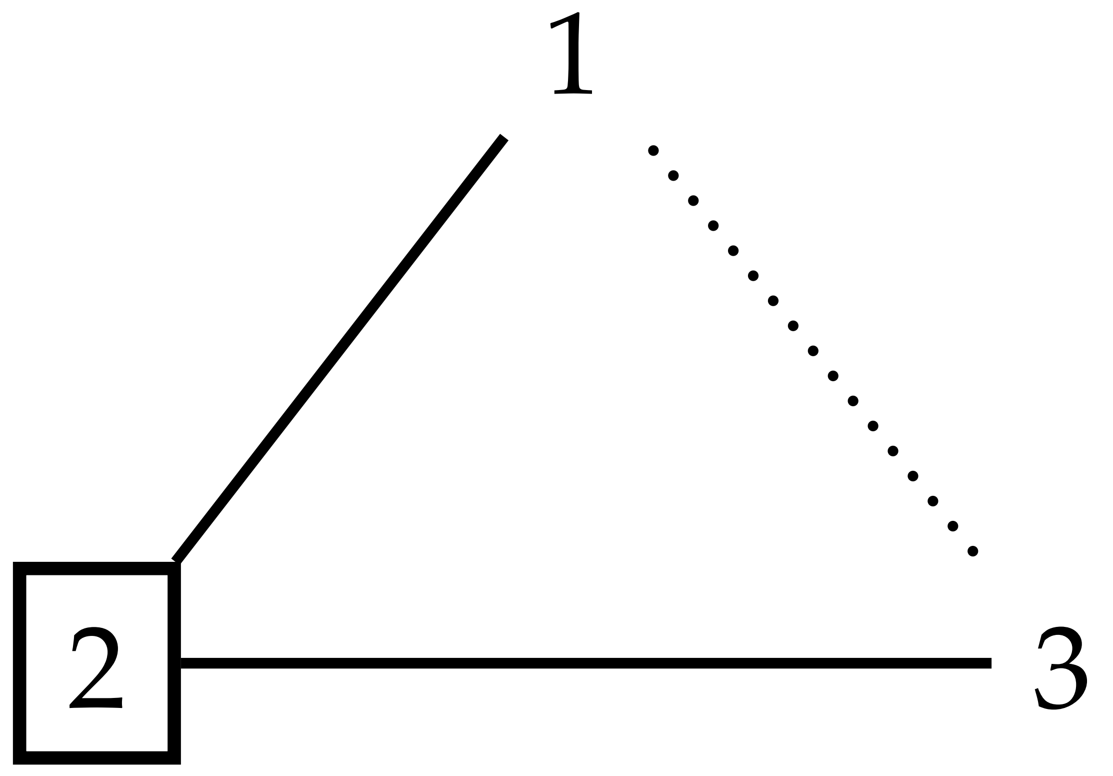
$$\underbrace{Q(\mathbf{x}_F)}_{} = \bigwedge_{K \in \mathcal{E}^+} R_K(\mathbf{x}_K) \wedge \bigwedge_{K \in \mathcal{E}^-} \neg R_K(\mathbf{x}_K)$$

head *positive* *negative*

- 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

$$Q(x_2) = R(x_1, x_2) \wedge S(x_2, x_3) \wedge \neg T(x_1, x_3)$$

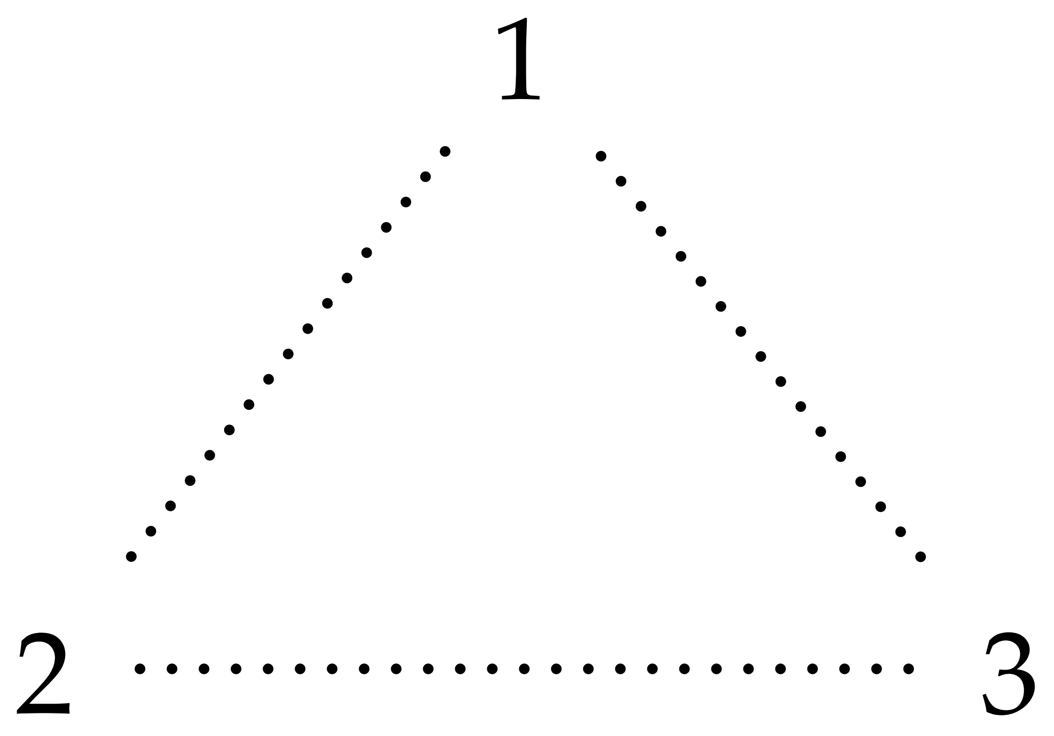


Example: 3-independent set

$$Q() = V(x_1) \wedge V(x_2) \wedge V(x_3) \wedge \neg R(x_1, x_2) \wedge \neg R(x_2, x_3) \wedge \neg R(x_1, x_3)$$

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

$$Q() = \neg R(x_1, x_2) \wedge \neg R(x_2, x_3) \wedge \neg R(x_1, x_3)$$

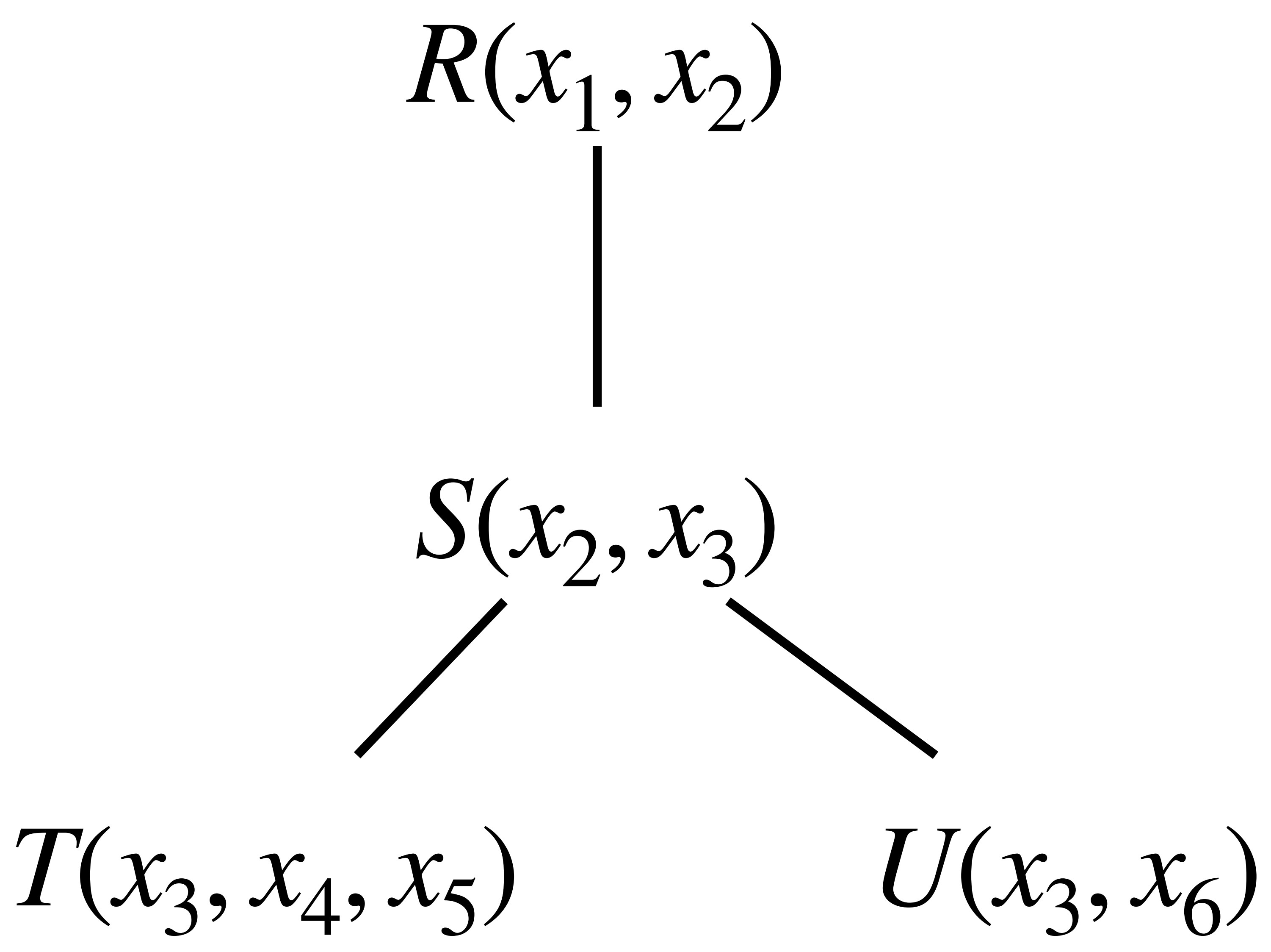


Some Background

α-acyclicity

A CQ is α-acyclic if and only if it admits **a join tree**

$$Q() = R(x_1, x_2) \wedge S(x_2, x_3) \wedge T(x_3, x_4, x_5) \wedge U(x_3, x_6)$$



The structure of α-acyclicity

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

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

$$\begin{aligned}
x_1 &: {\color{red}R(x_1, x_2)} \wedge S(x_2, x_3) \wedge {\color{red}T(x_3, x_1)} \wedge {\color{blue}U(x_1, x_2, x_3)} \\
x_2 &: {\color{red}R(x_1, x_2)} \wedge {\color{red}S(x_2, x_3)} \wedge T(x_3, x_1) \wedge {\color{blue}U(x_1, x_2, x_3)} \\
x_3 &: R(x_1, x_2) \wedge {\color{red}S(x_2, x_3)} \wedge {\color{red}T(x_3, x_1)} \wedge {\color{blue}U(x_1, x_2, x_3)}
\end{aligned}$$

All variables are α-leaves for this hypergraph!

The structure of α-acyclicity

A CQ is α-acyclic **iff** it admits an α-elimination sequence. At every step:

1. find any α-leaf x (with pivot R)
2. remove any relation with variables contained in R
3. remove x from R

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

$$\begin{aligned}
x_1 \text{ is an } \alpha\text{-leaf} &: R(x_1, x_2) \wedge S(x_2, x_3) \wedge T(x_3, x_1) \wedge U(x_1, x_2, x_3) \\
x_2 \text{ is an } \alpha\text{-leaf} &: S(x_2, x_3) \wedge U(x_2, x_3) \\
x_3 \text{ is an } \alpha\text{-leaf} &: U(x_3) \\
&\vdots
\end{aligned}$$

A linear-time algorithm

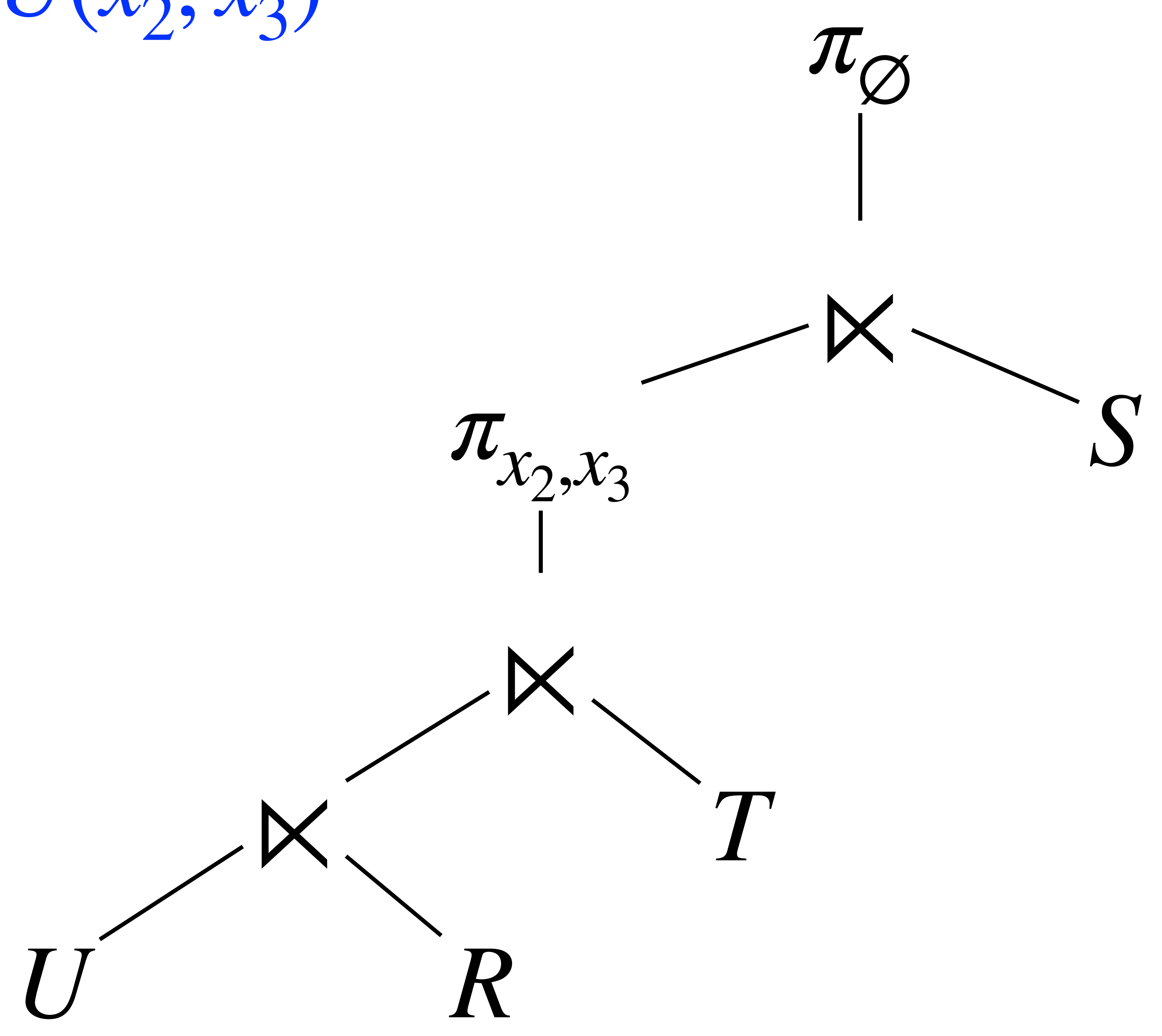
We follow the α-elimination sequence. At every step:

1. find any α-leaf x (with pivot R)
2. for any T with variables contained in R, update $R \leftarrow R \ltimes T$ and remove T
3. project out x from R

A linear-time algorithm

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

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



A linear-time characterization for CQs

[Yannakakis '81] For an α-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 $O(N + \mathsf{OUT})$
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

$$Q(x_1, x_2, x_3) = R(x_1, x_2) \wedge S(x_2, x_3) \wedge \neg T(x_1, x_2, x_3)$$

We can rewrite this query using a **difference** operator:

$$Q = \Big(R(x_1, x_2) \wedge S(x_2, x_3)\Big) - \Big(R(x_1, x_2) \wedge S(x_2, x_3) \wedge T(x_1, x_2, x_3)\Big)$$

Q1: acyclic CQ *Q2: acyclic CQ*

$$\#Q = \#Q_1 - \#Q_2$$

The Inclusion-Exclusion Principle

$$Q(\mathbf{x}) = \bigwedge_{K \in \mathscr{E}^+} R_K(\mathbf{x}_K) \wedge \bigwedge_{K \in \mathscr{E}^-} \neg R_K(\mathbf{x}_K)$$

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

$$\#Q = \sum_{S \subseteq \mathscr{E}^-} (-1)^{|S|} \#Q_S$$

where Q_S is the CQ with hypergraph $([n], \mathscr{E}^+ \cup S)$

Signed-acyclicity

If the hypergraph $\mathcal{E}^+ \cup S$ is α-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 α-acyclic for any $S \subseteq \mathcal{E}^-$

[Brault-Baron '13]

Signed-acyclicity: examples

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

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

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

✓ $Q() = \neg R(x_1, x_2) \wedge \neg S(x_2, x_3) \wedge \neg T(x_3, x_4)$

β-acyclicity

- Suppose all positive relations are unary (arity = 1)
- Then signed-acyclicity is equivalent to: any subset of $\mathscr{E}^-$ is α-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 $K \in \mathscr{E}^+$ (pivot) such that:

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

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

pivot for x_4

The structure of signed-acyclicity

A CQ is signed-acyclic **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 (α-property)
3. remove x from everywhere (β-property)

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

A linear-time algorithm

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

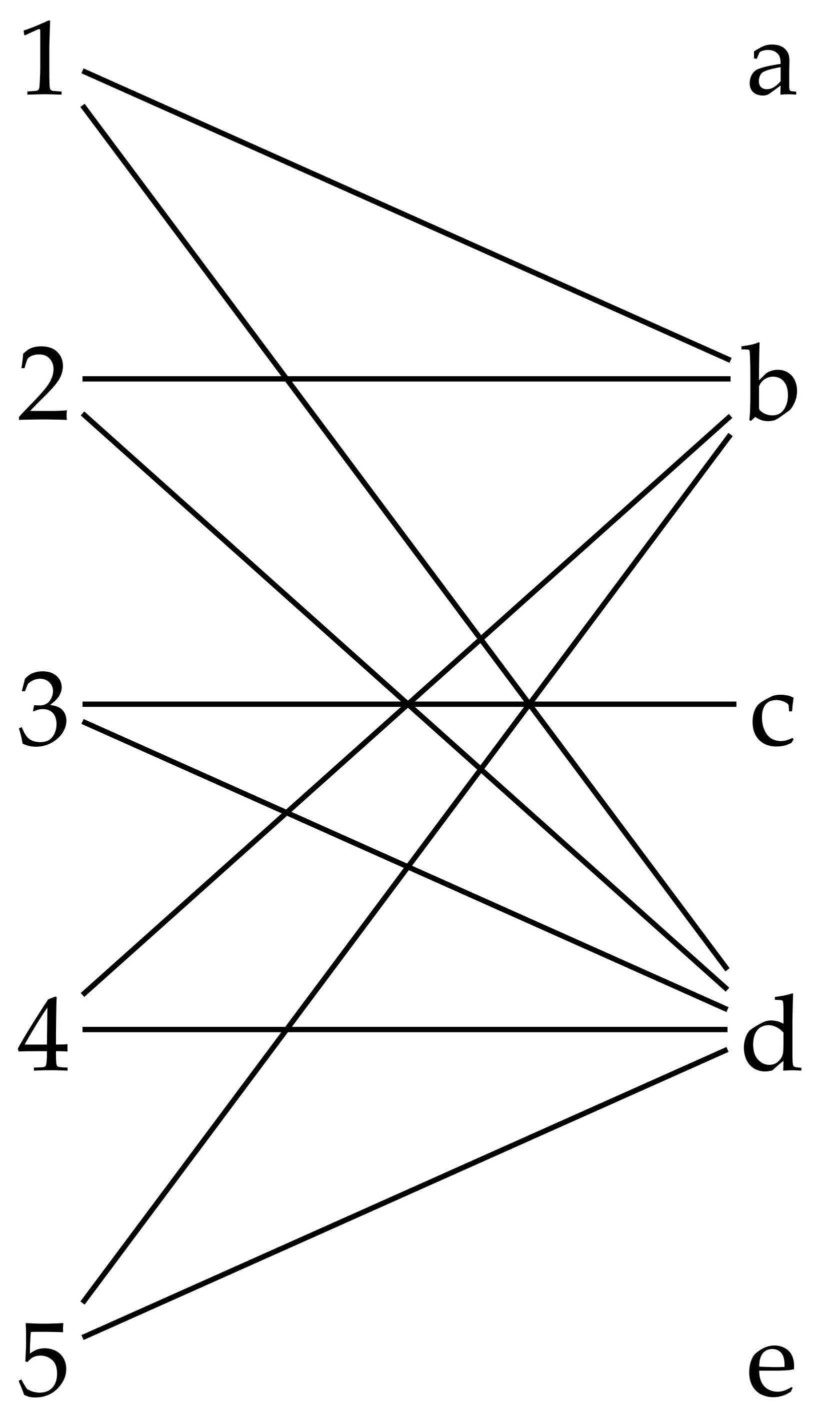
1. find any signed-leaf x (with pivot R)
2. Semi-join with R and remove any relation with variables contained in R (<u>α-property</u>)
3. “Remove” x from every relation that contains it (<u>β-property</u>)

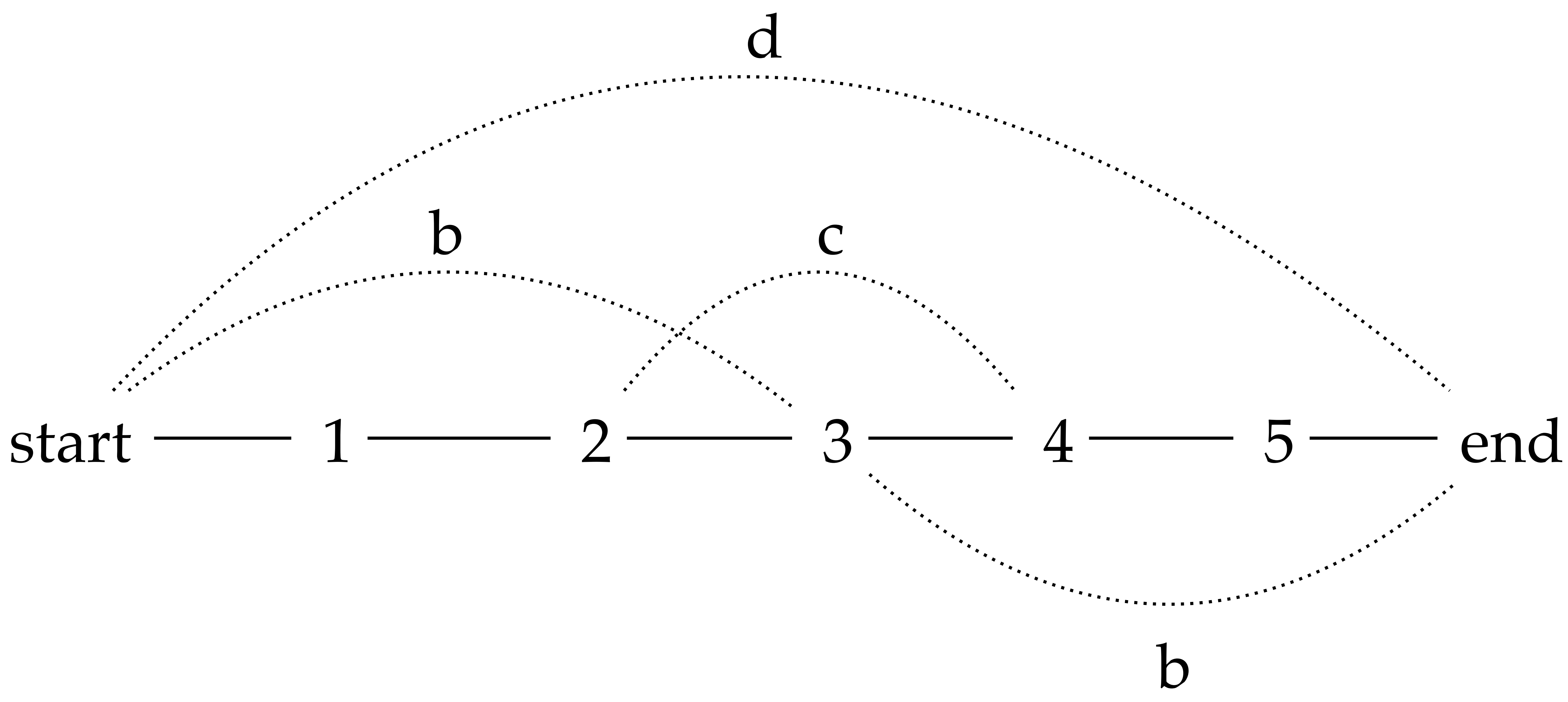
Item #3 is the challenging one!

The Key Idea

$$Q() = A(x_1) \wedge B(x_2) \wedge \neg R(x_1, x_2)$$

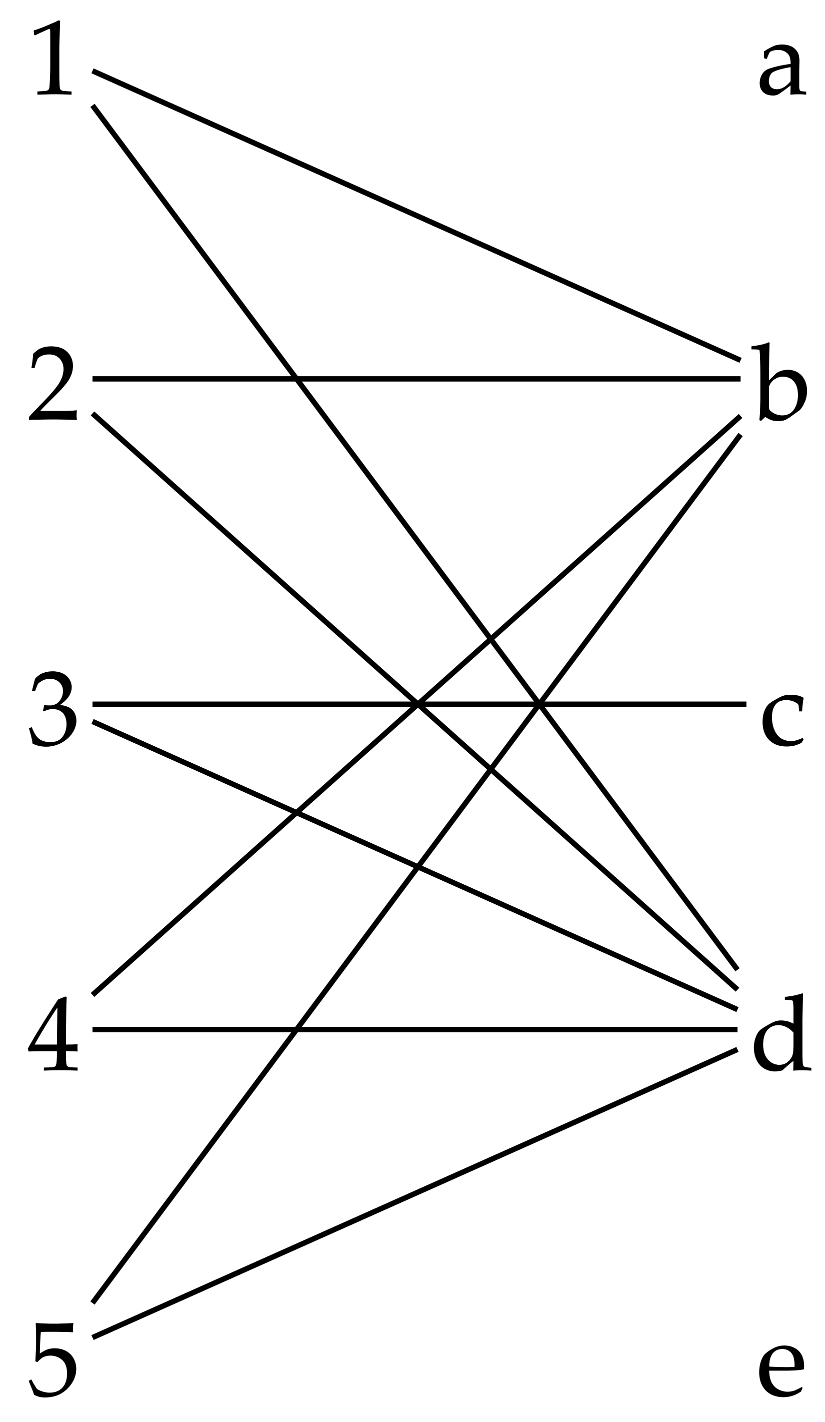
- We cannot afford to scan A for every value of B
- We build a data structure that encodes the "skips"

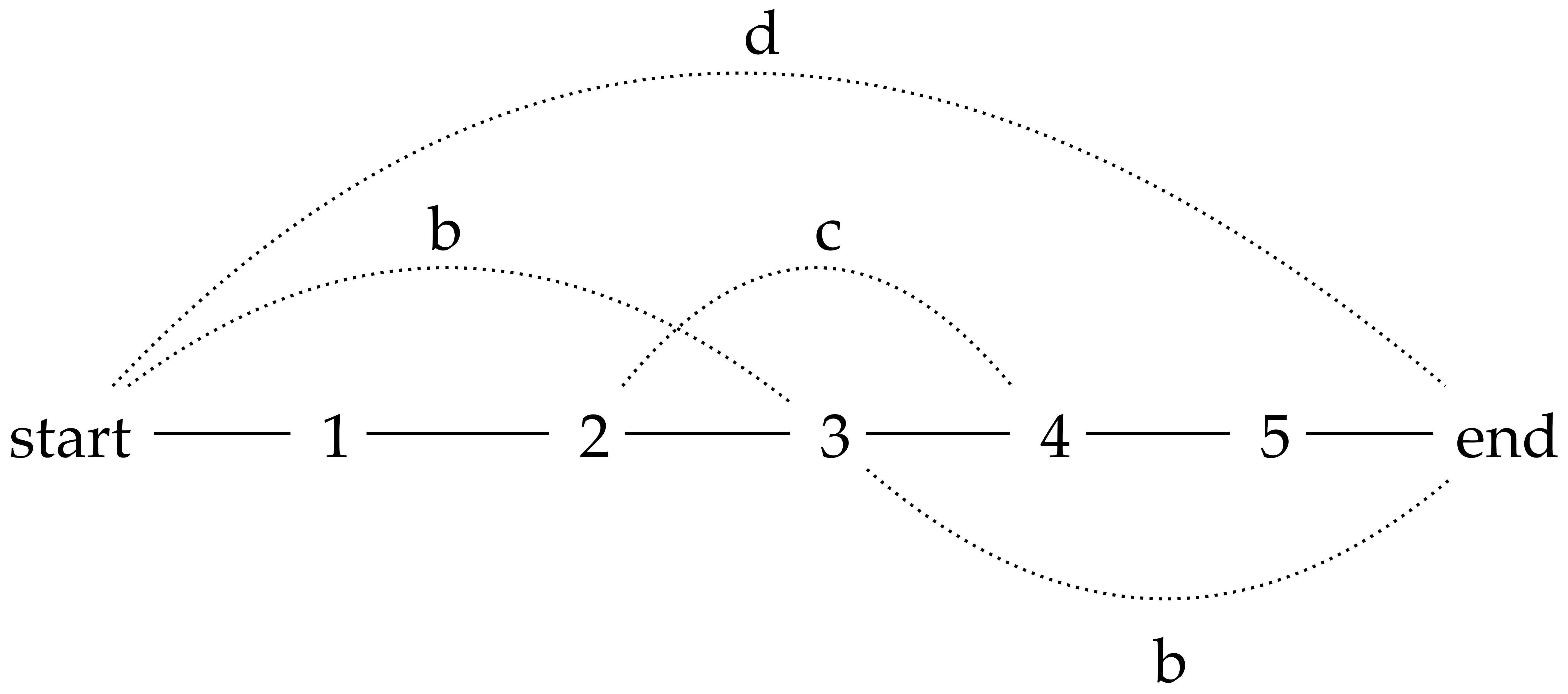




The Key Idea

$Q() = A(x_1) \wedge B(x_2) \wedge \neg R(x_1, x_2)$





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

$Q'() = B(x_2) \wedge \neg R(x_2)$

B

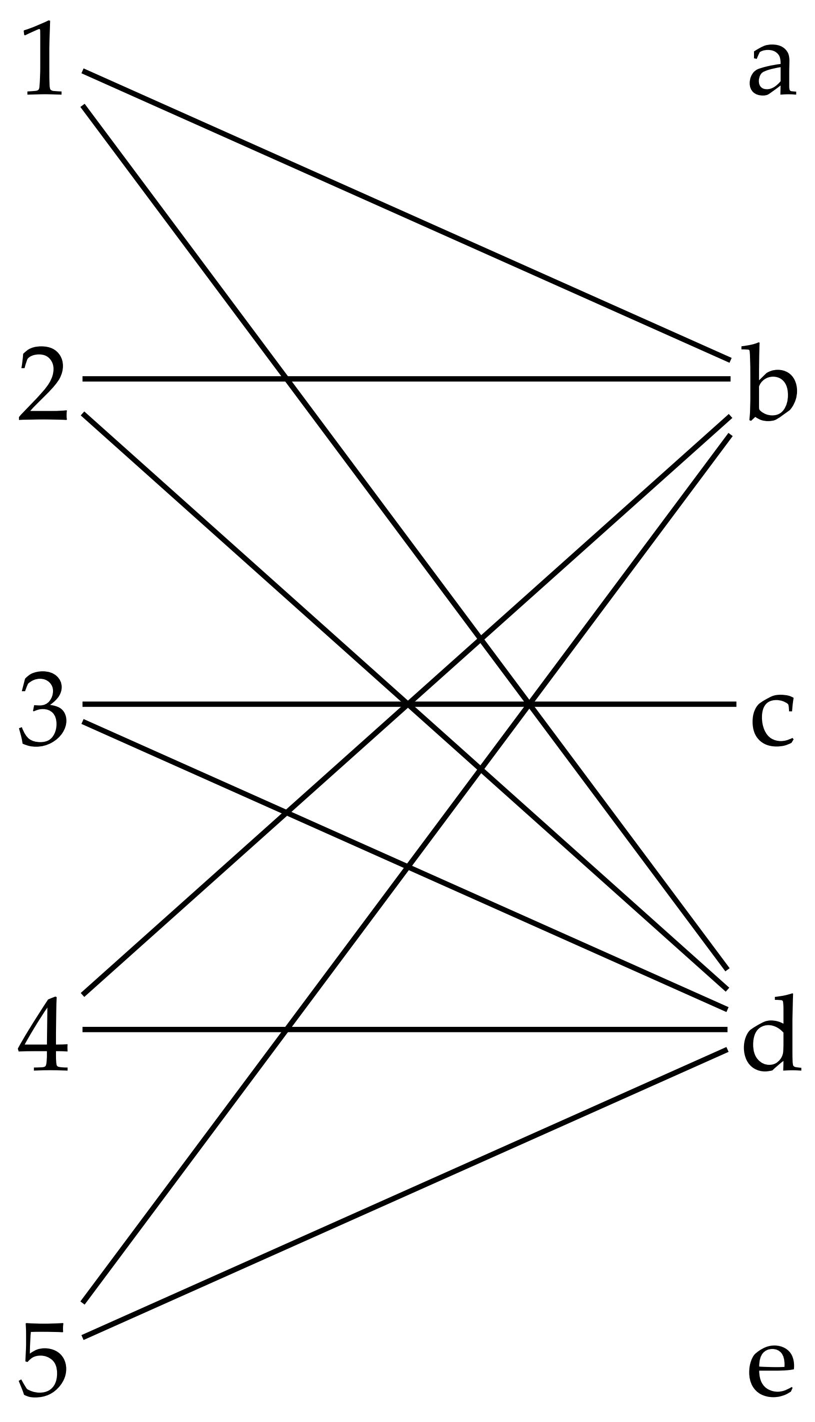
a
b
c
d
e

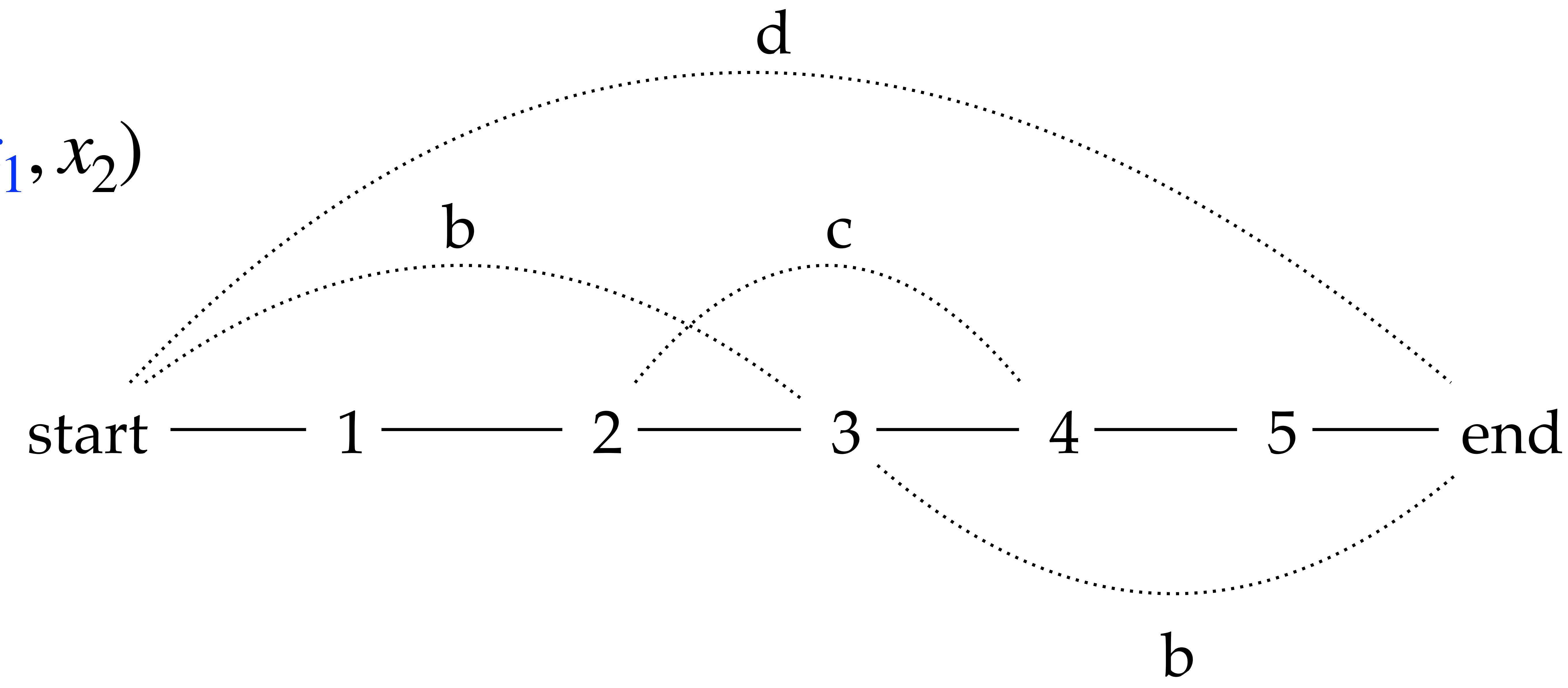
R

d

Enumeration

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





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 $O(N + \mathsf{OUT})$

Moreover, the algorithms have polynomial combined complexity

What about projections?

$$Q(\mathbf{x}_F) = \bigwedge_{K \in \mathscr{E}^+} R_K(\mathbf{x}_K) \wedge \bigwedge_{K \in \mathscr{E}^-} \neg R_K(\mathbf{x}_K)$$

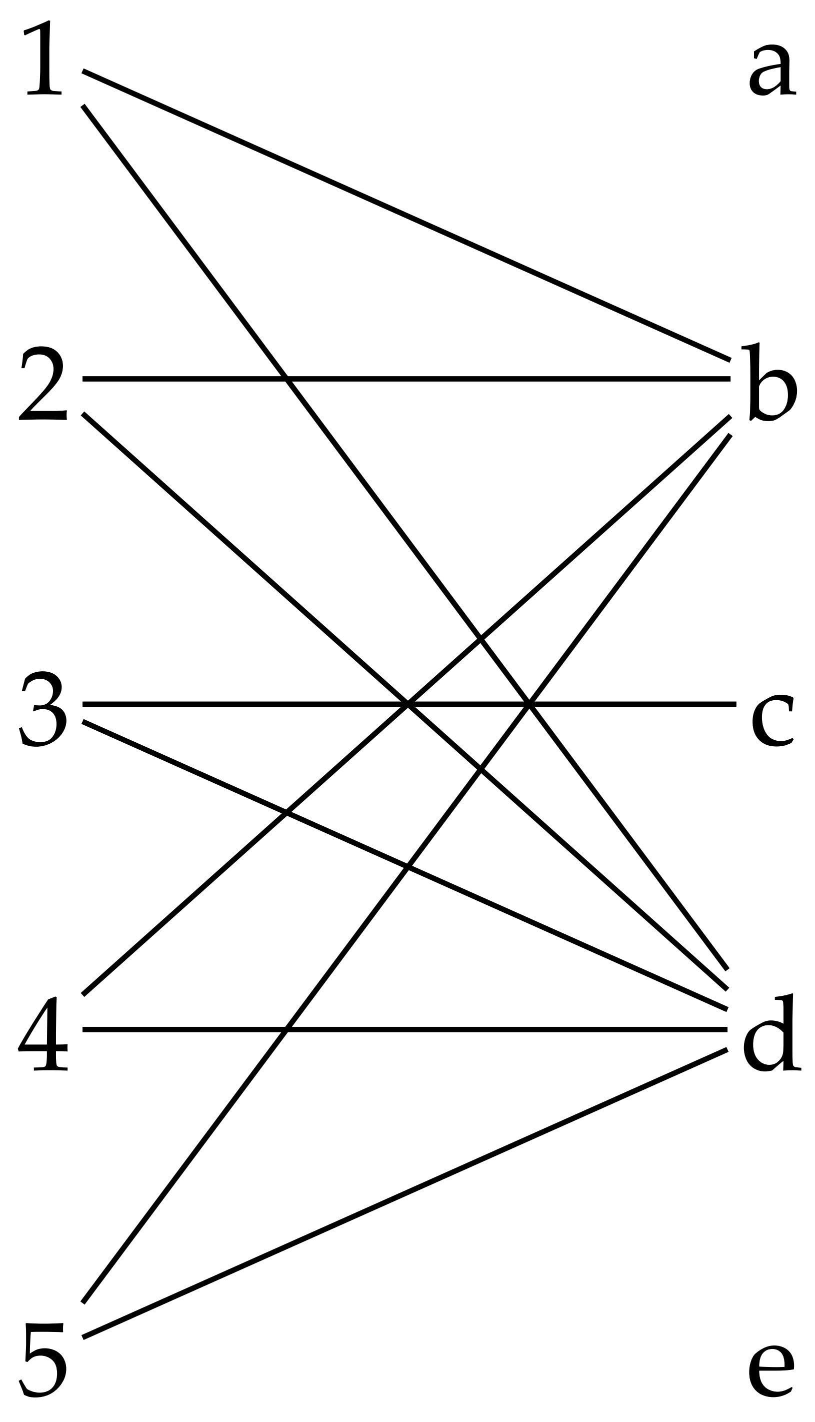
If the signed hypergraph $([n], \mathscr{E}^+, \mathscr{E}^- \cup \{F\})$ is signed-acyclic, the output can be enumerated with constant delay after linear-time preprocessing

- 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

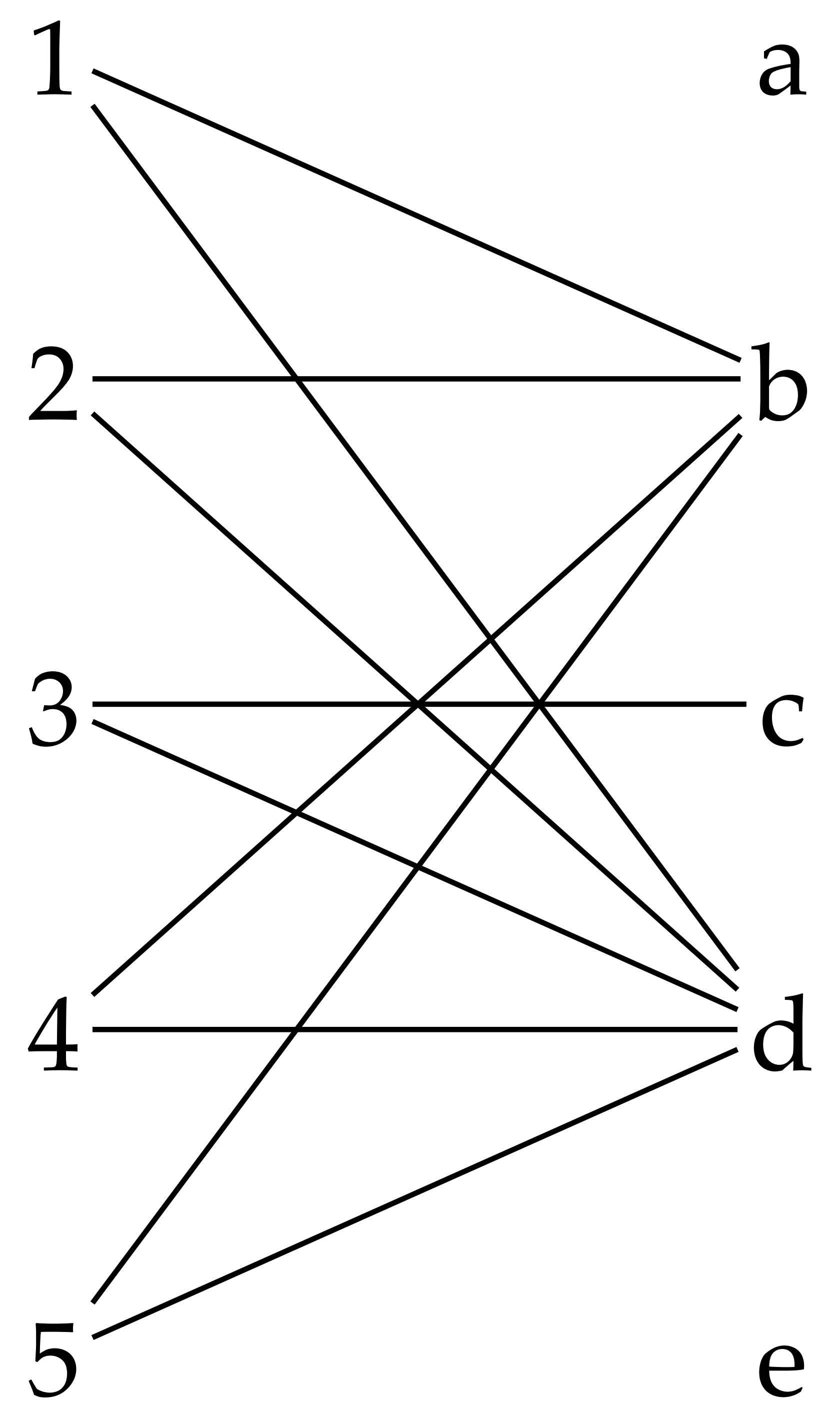
$$\#Q(x_2) = A(x_1) \wedge B(x_2) \wedge \neg R(x_1, x_2)$$

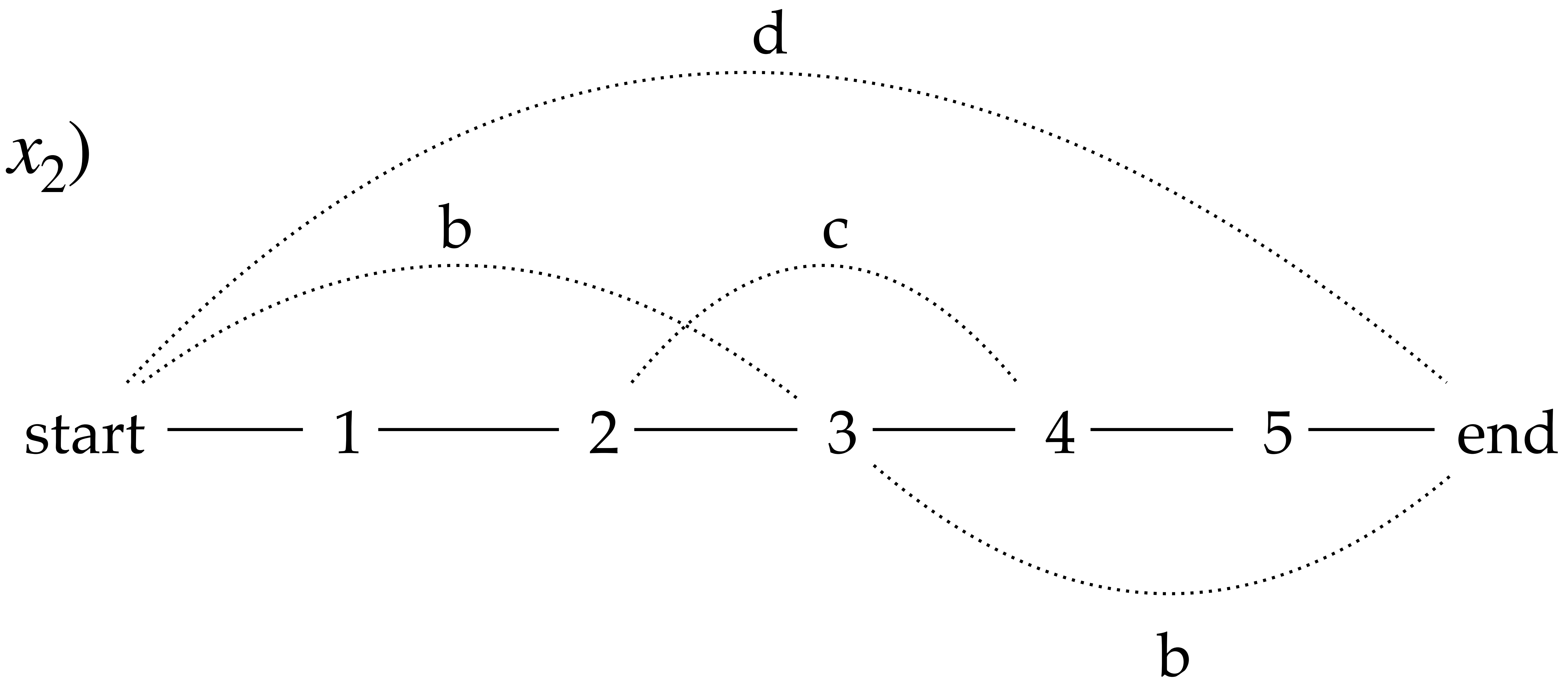


For every value of B, count the number of nodes from A that are not connected with it

Counting: example

$$\#Q(x_2) = A(\color{blue}{x_1}) \wedge B(x_2) \wedge \neg R(\color{blue}{x_1}, x_2)$$





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

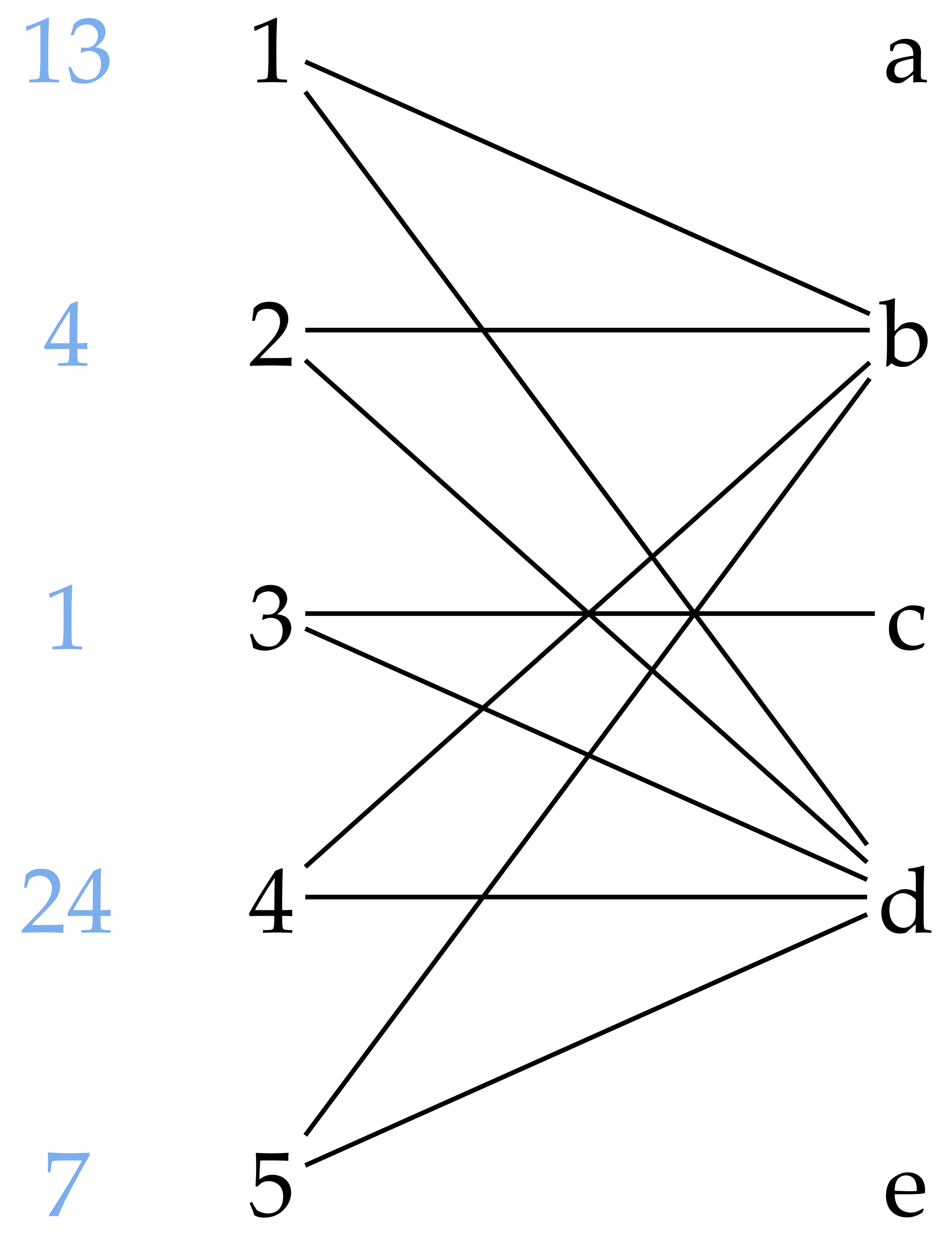
$a : [1 - 5]$

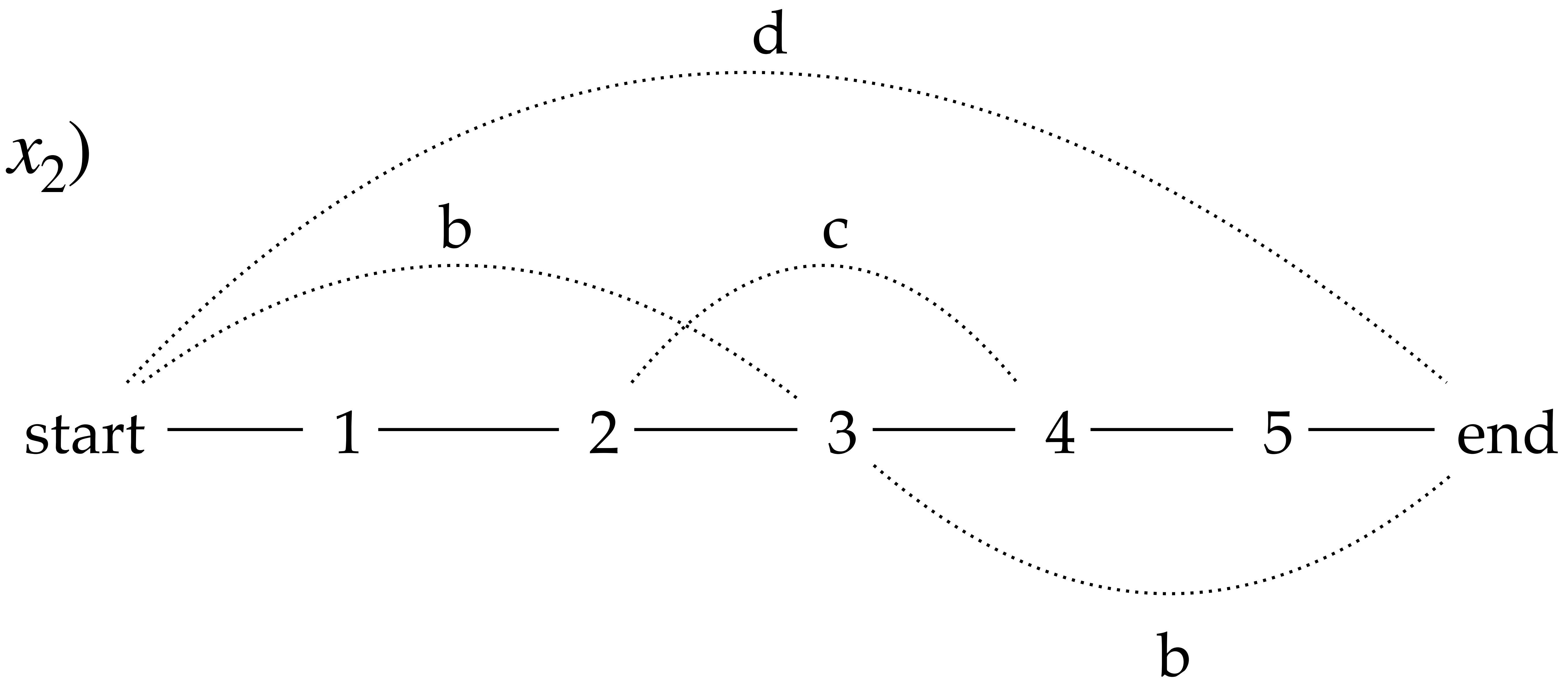
$b : [3]$

$c : [1 - 2], [4 - 5]$

Summing: example

$$\#Q(x_2) = A(x_1) \wedge B(x_2) \wedge \neg R(x_1, x_2)$$



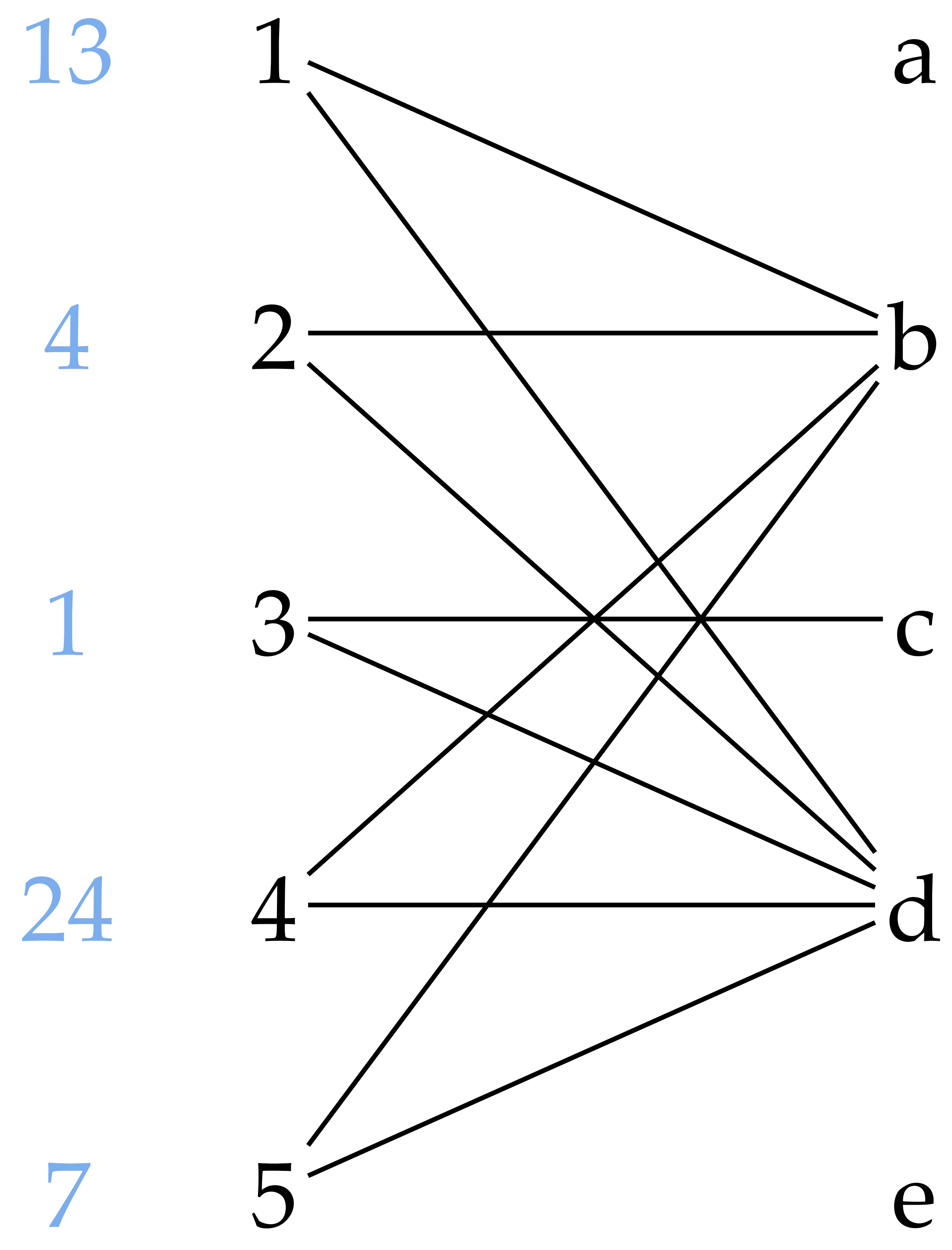


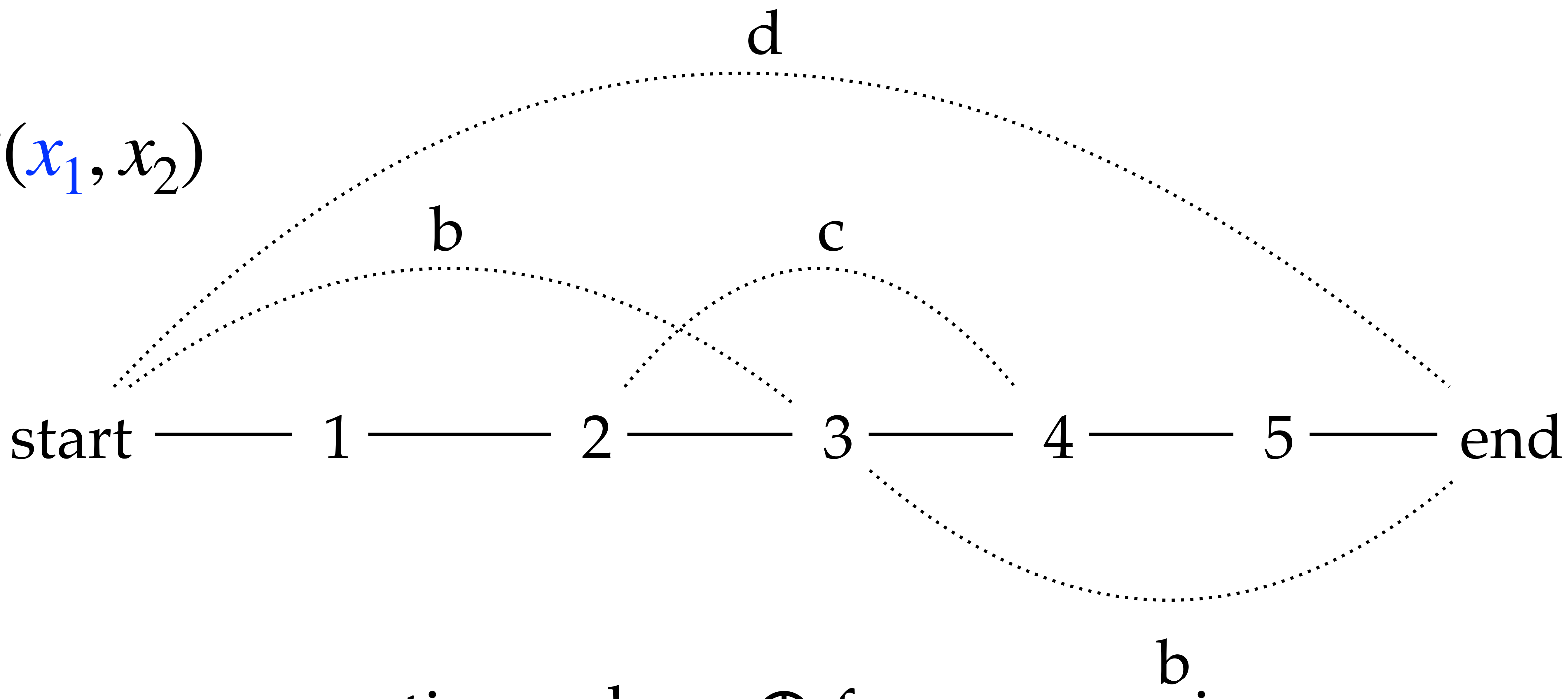
- We need to compute the partial sums
- We can build a data structure in linear time such that we can calculate each partial sum in constant time (OFFLINE PARTIAL SUMS)

Idea: $\sum_{i=u}^{v} x_i = \sum_{i=1}^{v} x_i - \sum_{i=1}^{v} x_i$

General Aggregation

$$\#Q(x_2) = \bigoplus_{x_1} A(x_1) \wedge B(x_2) \wedge \neg R(x_1, x_2)$$





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

$$Q(\mathbf{x}_F) = \bigoplus_{\mathbf{x}_{[n]\setminus F}} \bigotimes_{K \in \mathscr{E}^+} R_K(\mathbf{x}_K) \wedge \bigotimes_{K \in \mathscr{E}^-} \bar{R}_K(\mathbf{x}_K)$$

positive factor *negative factor*

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

Aggregation in Arbitrary CQNs

$$Q(\mathbf{x}_F) = \bigoplus_{\mathbf{x}_{[n]\setminus F}} \bigotimes_{K\in\mathscr{E}^+} R_K(\mathbf{x}_K) \wedge \bigotimes_{K\in\mathscr{E}^-} \bar{R}_K(\mathbf{x}_K)$$

For any semiring, if the signed hypergraph $([n], \mathscr{E}^+, \mathscr{E}^- \cup \{F\})$ 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]

$$
\begin{aligned}
Q &= \big(R(x_1, x_2) \wedge S(x_2, x_3, x_4)\big) - \big(T(x_1, x_2) \wedge U(x_2, x_3)\big) \\
&= \big(R(x_1, x_2) \wedge S(x_2, x_3, x_4) \wedge \neg T(x_1, x_2)\big) \cup \big(R(x_1, x_2) \wedge S(x_2, x_3, x_4) \wedge \neg U(x_2, x_3)\big)
\end{aligned}
$$

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 $Q(y) = R'(y) \wedge S(x) \wedge \neg R(x, y)$ 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