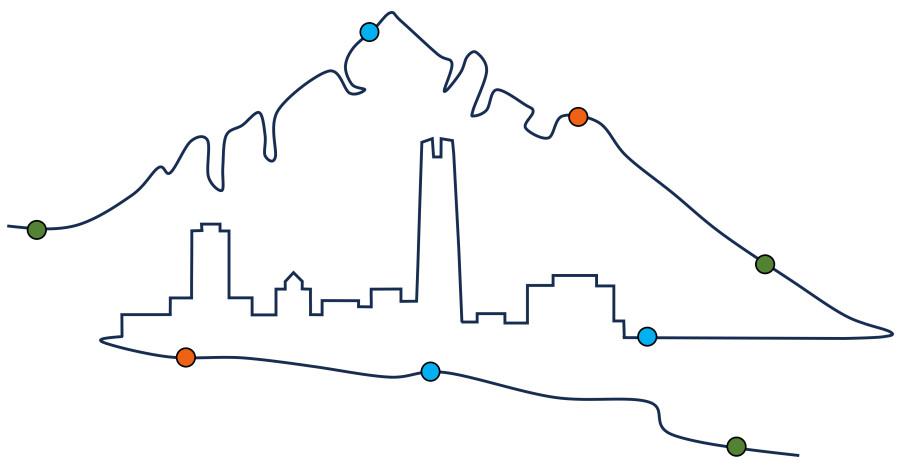
Distinct Shortest Walk Enumeration for RPQs

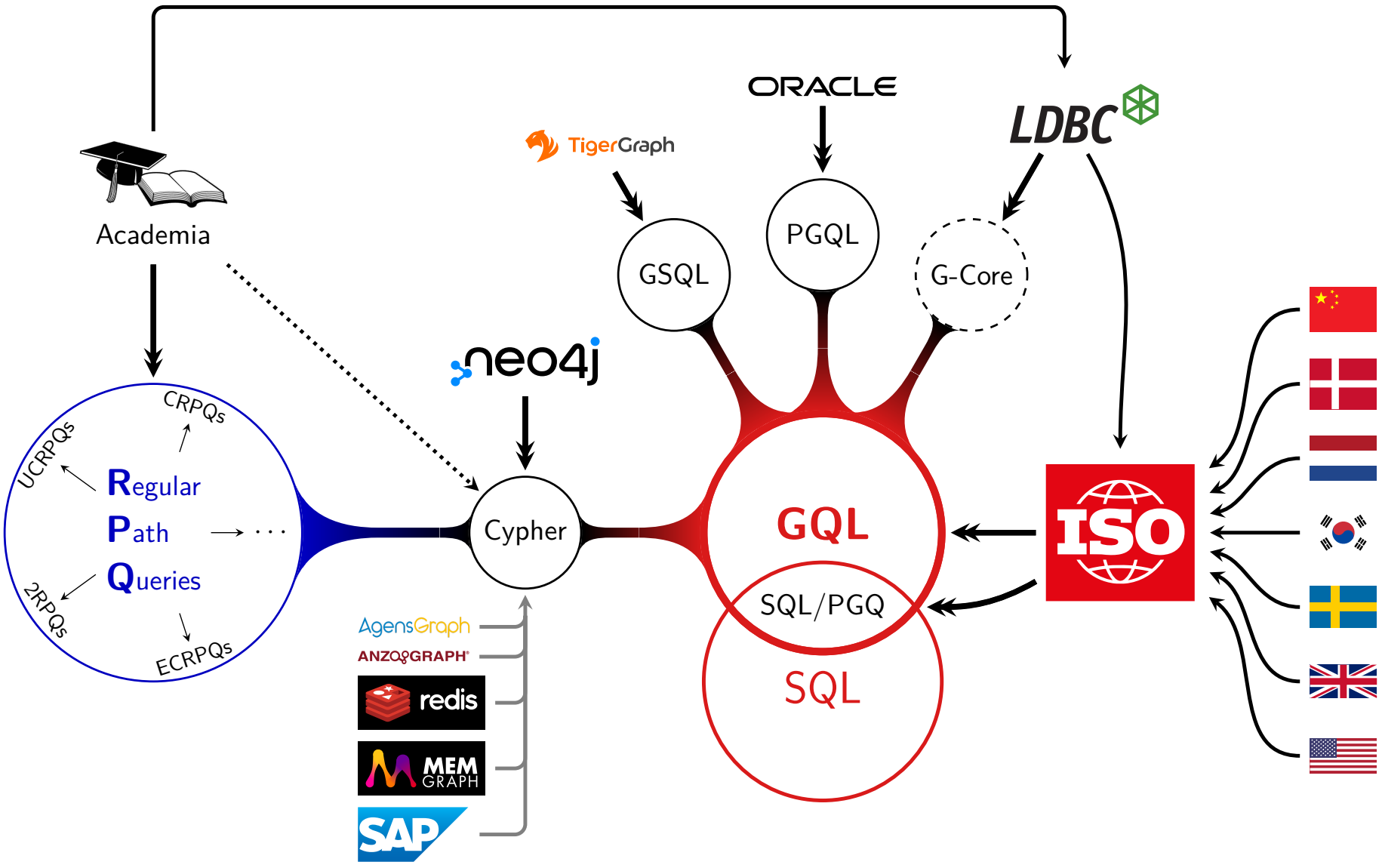
Claire DAVID, Nadime FRANCIS and Victor MARSAULT

LIGM, Université Gustave Eiffel, CNRS - France

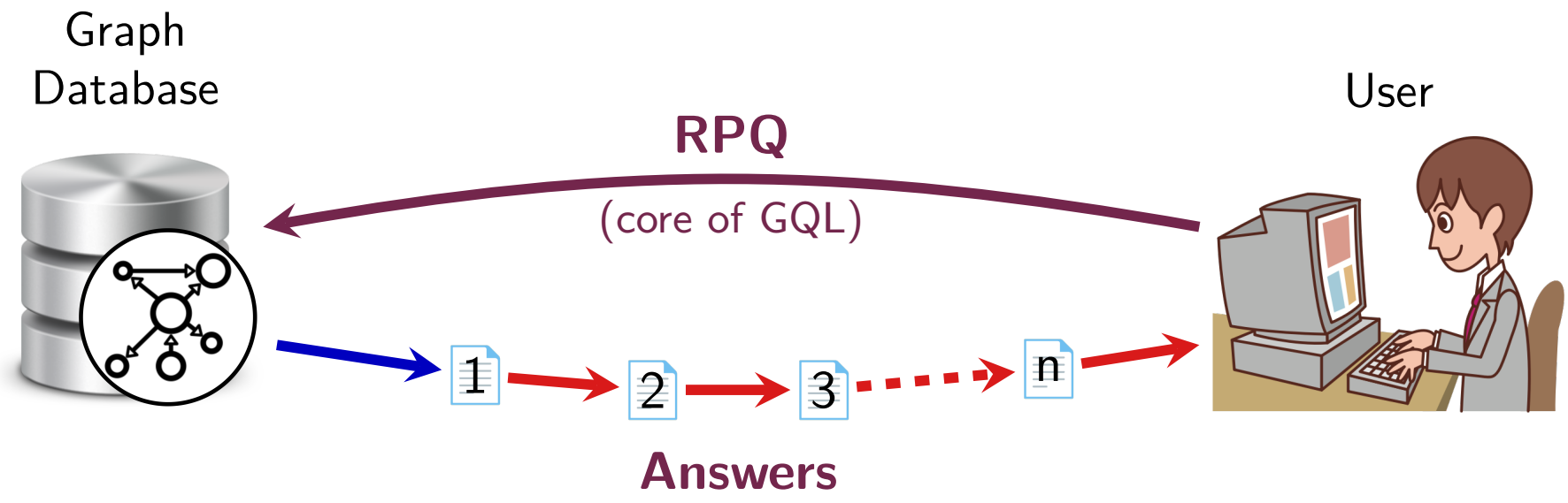


ACM SIGMOD/PODS'24 9-14 June 2024 Santiago, Chile

Ecosystem around GQL



Our problem

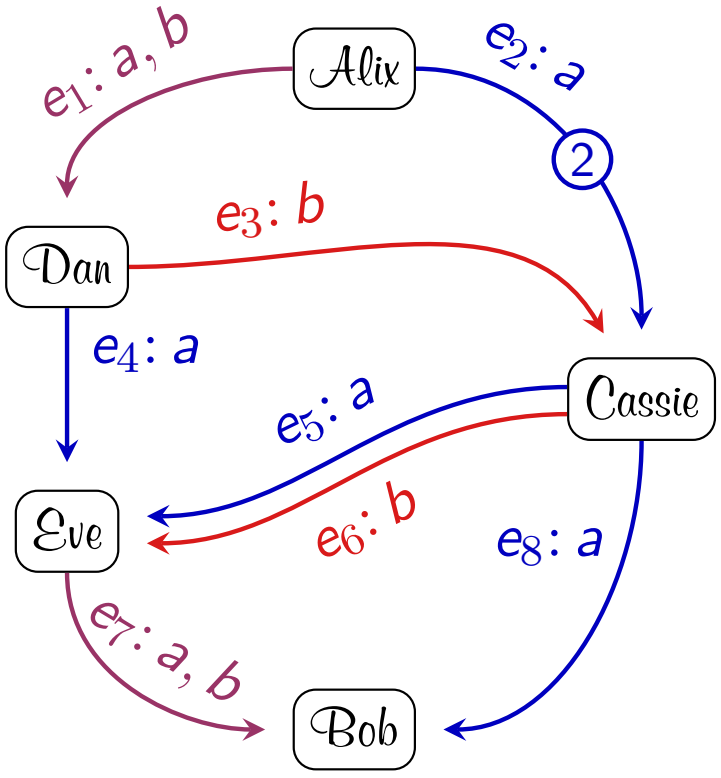


How to enumerate answers

- Without duplicate
- With good complexity
  - **Preprocessing**: time before first answer
  - **Delay**: time between consecutive answers
  - Memory Usage

Data model: Multi-labeled multi-edge graph

- Finite label alphabet: $\Sigma = \{a, b\}$
- Vertices: $\{\mathcal{A}lix, \mathcal{B}ob, \ldots, \mathcal{E}ve\}$
- Labelled edges: $\{e_1, \ldots, e_7\}$
  - e_1 carries two labels: a and b
  - e_2 carries one label: a
  - ...



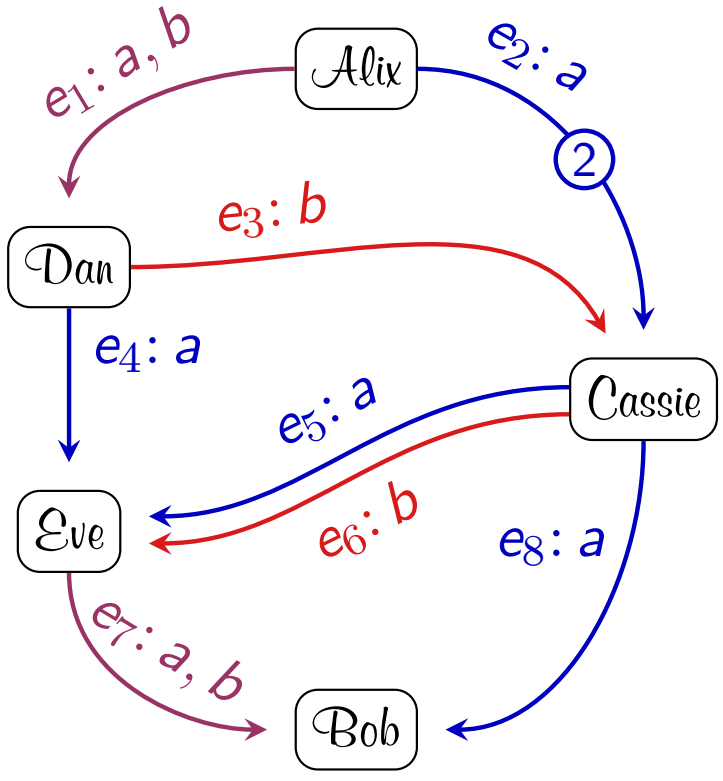
⚠ Graphs are multi-edge and multi-labeled like GQL data model

- **Walk** = consistent sequence of edges
  $e_1 e_4 e_7$ is a walk $e_1 e_8$ is not a walk
- **Labels of a walk** = concatenation of its edge labels
  $e_1 e_4 e_7$ carries four labels: *aaa baa aab bab*

Query language : RPQ

$Q ::= \mathbf{A}$ Atoms
$Q \cdot Q$ Concatenation
$Q + Q$ Disjunction
Q^* Kleene star
where $\mathbf{A}$ is a label in the graph.

Ex: $Q = b^* \cdot a \cdot (a + b)^*$



- **Match to** Q = **walk** in D with a label that conforms to Q

$e_1 e_4 e_7$ matches Q : it carries *bab*
$e_1 e_3 e_5 e_7$ matches Q : it carries *bbab*

⚠ $e_1 e_4 e_7$ matches Q three times and $e_1 e_3 e_5 e_7$ four times
⚠ Matches may have different length

Problem: Distinct Shortest Walk Enumeration

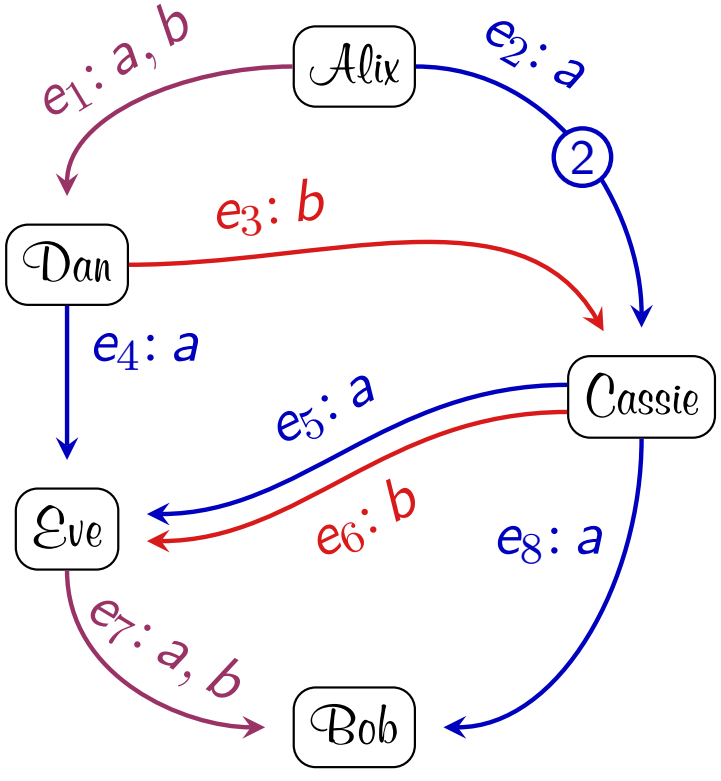
Enumerate all **shortest** walks in D from s to t matching Q **without duplicates**.

Ex: $s = \mathcal{A}lix$, $t = \mathcal{B}ob$, $Q = b^* \cdot a \cdot (a + b)^*$

- No matches of length 1 or 2
- $e_1\, e_4\, e_7$
- $e_1\, e_3\, e_8$
- $e_2\, e_5\, e_7$
- $e_2\, e_6\, e_7$

} Shortest walks : length 3

- $e_1\, e_3\, e_5\, e_7$ has length >3



⚠ Output each answer only once

Previous results

Martens, Trautner 2018

Distinct enumeration of all shortest walks can be done with

- Polynomial time preprocessing
- Polynomial time delay

Based on Ackerman, Shallit, 2009

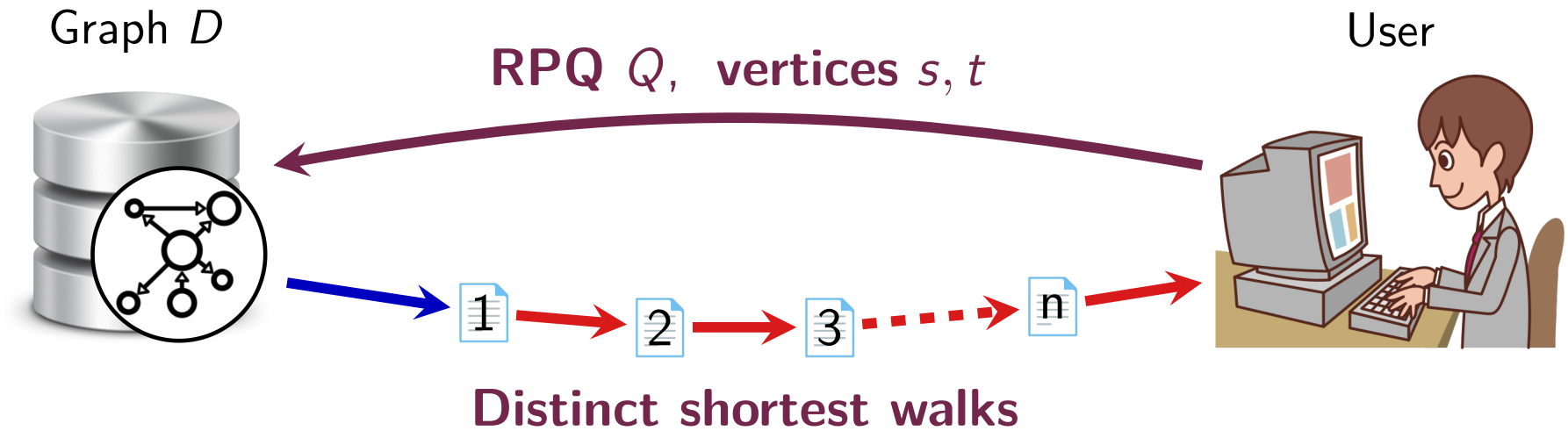
Folklore

When duplicates cannot occur, enumeration can be done with

- $\mathbf{O}\big(|Q| \times |D|\big)$ preprocessing
- $\mathbf{O}(\lambda)$ delay

where λ is the length of one output.

Our contribution



An algorithm to enumerate distinct shortest walks with

- $\mathbf{O}(|Q| \times |D|)$ preprocessing (time before first answer)
- $\mathbf{O}(|Q| \times |\lambda|)$ delay (time between two answers)
- $\mathbf{O}(|Q| \times |D|)$ memory usage

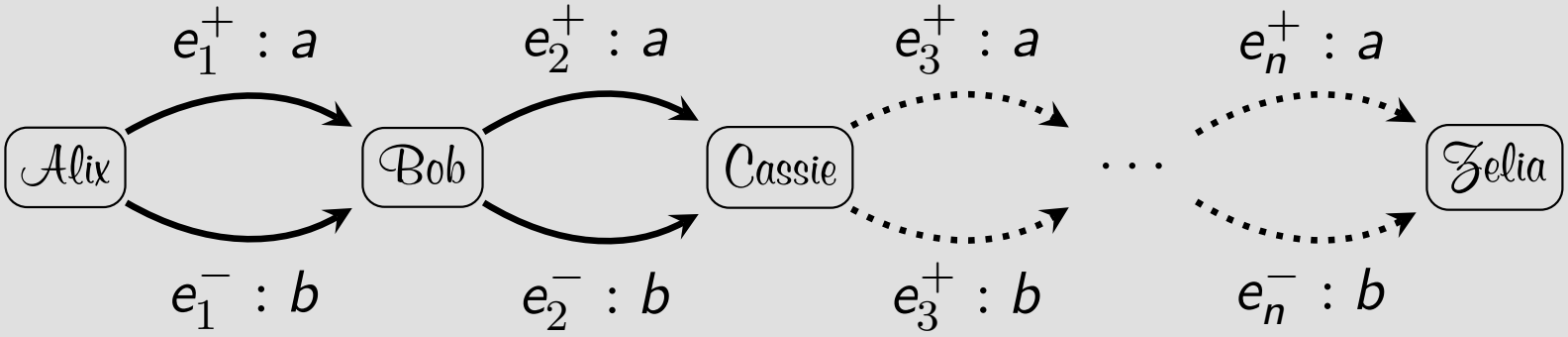
where λ is the length of one output.

Exponentially many answers

There are 2^n walks from $\mathcal{A}lix$ to $\mathcal{Z}elia$ matching Q

Q: $(a+b)^*$

D:

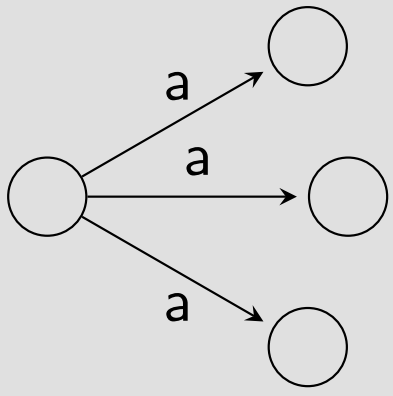


⚠ Memory usage must not be linear in the size of output ⚠

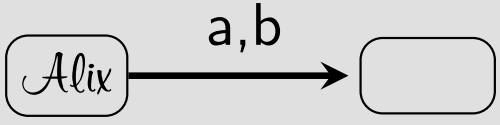
🧐 Where do duplicates comes from?

From the query

Q:



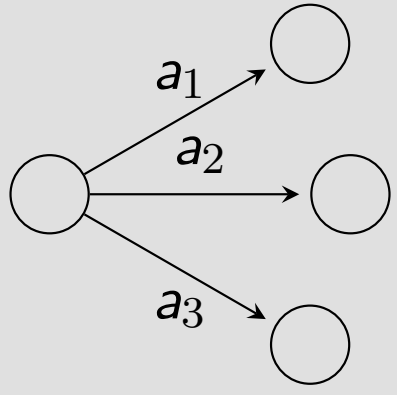
D:



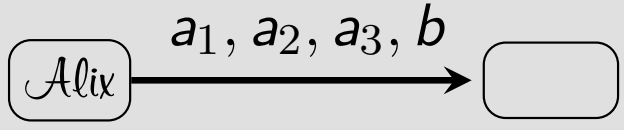
$\Longleftrightarrow$

From the data

Q:



D:



⚠️ Determinizing the query/automaton does not help ⚠️

Exponentially many duplicates

The walk from $\mathcal{Alix}$ to $\mathcal{Zelia}$ matching Q has 2^n duplicates

Q: $(a + b)^*$

D:

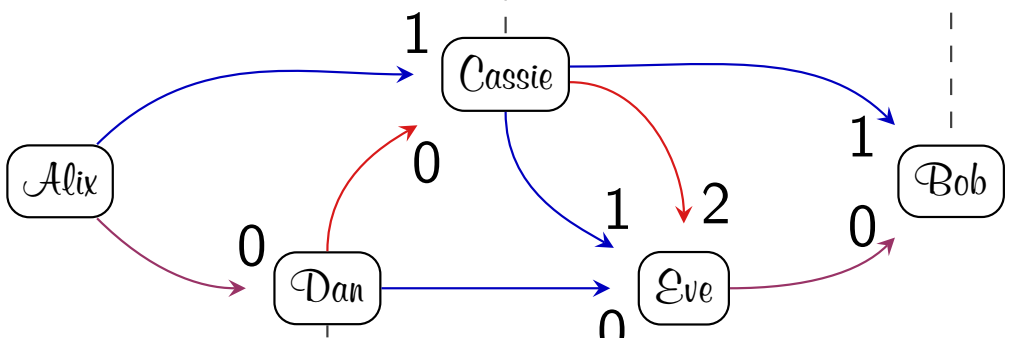
⚠ The algorithm cannot check whether a walk was already output ⚠

Preprocessing : Annotation of D

q	$L_{\mathcal{C}}[q]$	i	$B_{\mathcal{C}}[q][i]$	$C_{\mathcal{C}}[q]$
0	1	0	[]	
		1	[0]	• ←
1	2	0	[0,1]	• ←
		1	[]	

q	$L_{\mathcal{B}}[q]$	i	$B_{\mathcal{B}}[q][i]$	$C_{\mathcal{B}}[q]$
0	2	0	[]	
		1	[0]	• ←
1	3	0	[1,0,1]	• ↺
		1	[1]	• ←

q	$L_{\mathcal{A}}[q]$
0	0
1	$\perp$



q	$L_{\mathcal{D}}[q]$	i	$B_{\mathcal{D}}[q][i]$	$C_{\mathcal{D}}[q]$
0	1	0	[0]	• ←
1	1	0	[0]	• ←

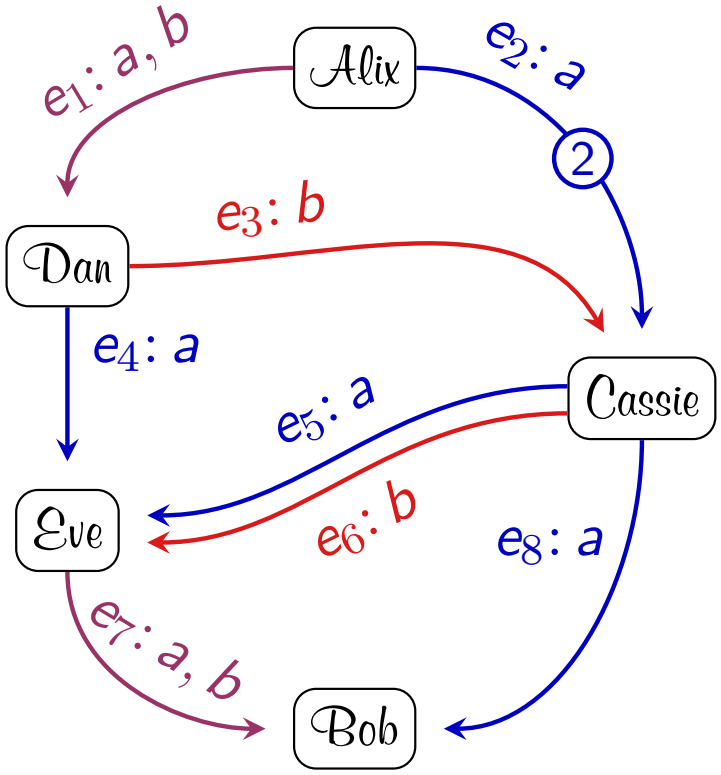
q	$L_{\mathcal{E}}[q]$	i	$B_{\mathcal{E}}[q][i]$	$C_{\mathcal{E}}[q]$
0	2	0	[0]	• ↺
		1	[0]	• ←
		2	[]	
1	2	0	[1]	• ↺
		1	[]	
		2	[0]	• ←

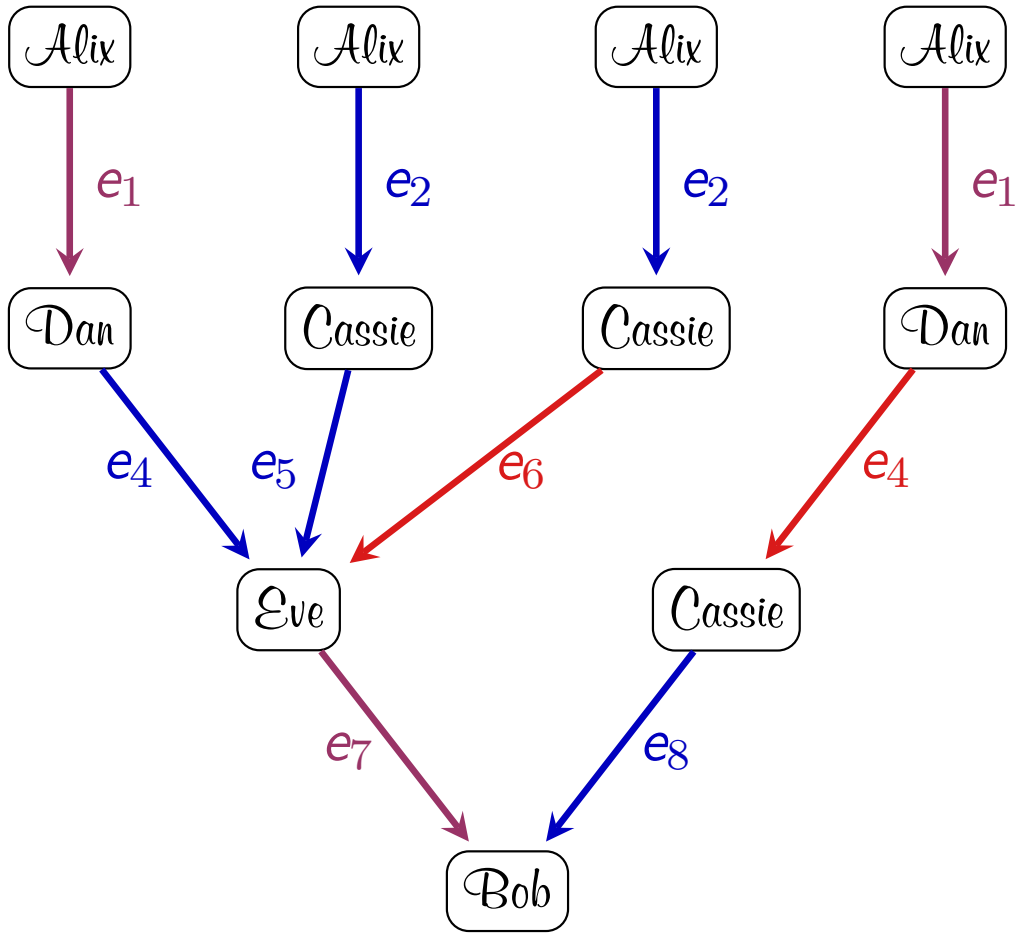
Enumeration strategy: DFS of the backward tree

Inputs:

$Q = b^* \cdot a \cdot (a + b)^*$

$s = \mathcal{A}lix \quad t = \mathcal{B}ob$





Backward tree of answers

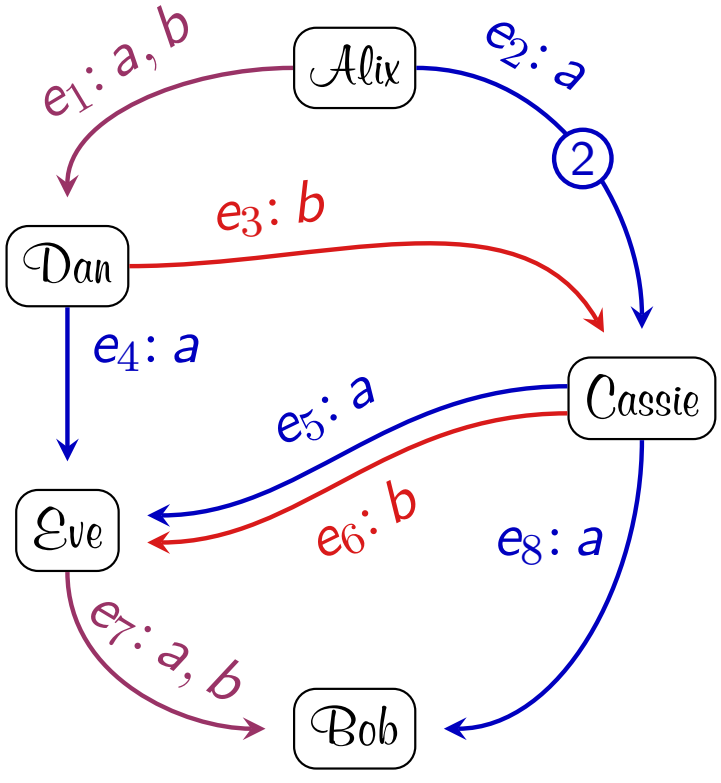
(up to exponential in $|D|$ and $|Q|$)

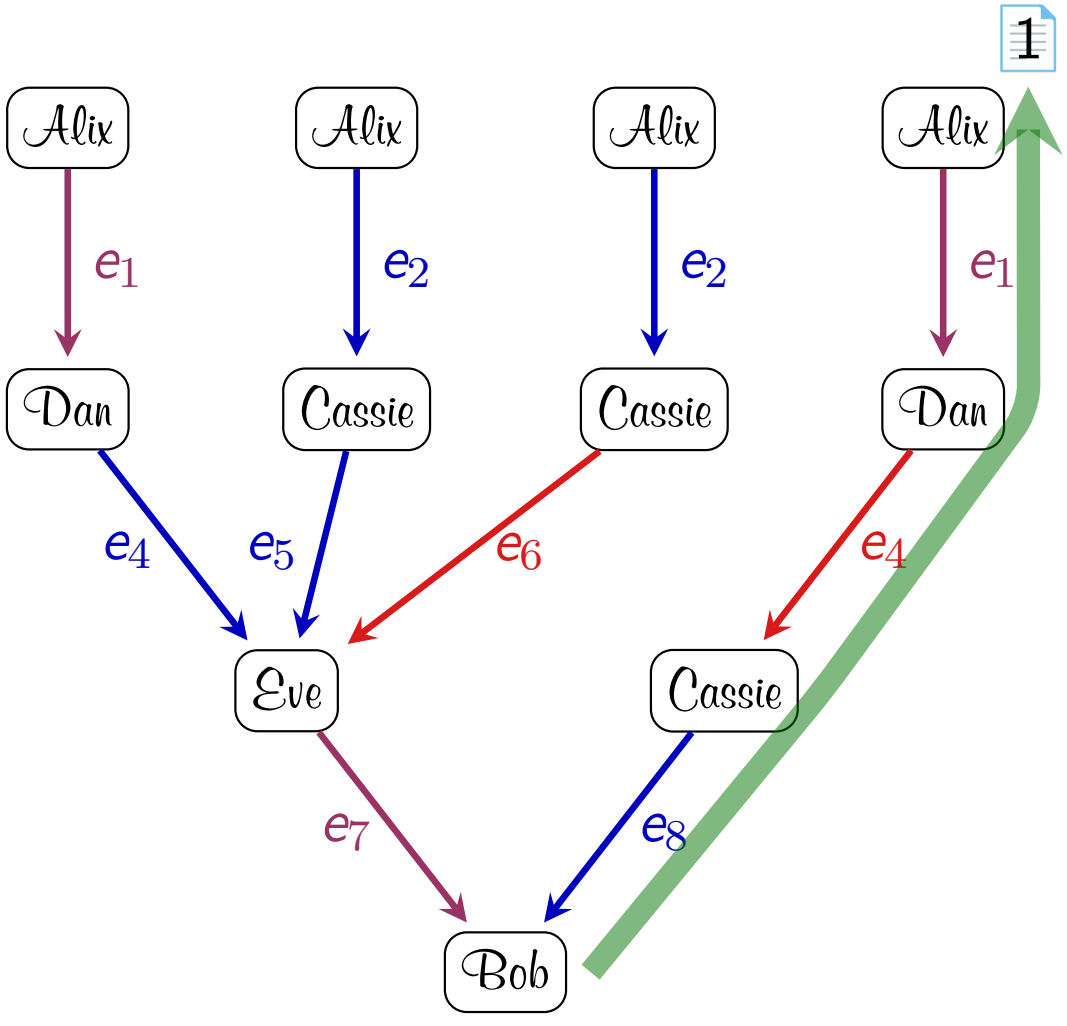
Enumeration strategy: DFS of the backward tree

Inputs:

$Q = b^* \cdot a \cdot (a + b)^*$

$s = \mathcal{A}lix \quad t = \mathcal{B}ob$





Backward tree of answers

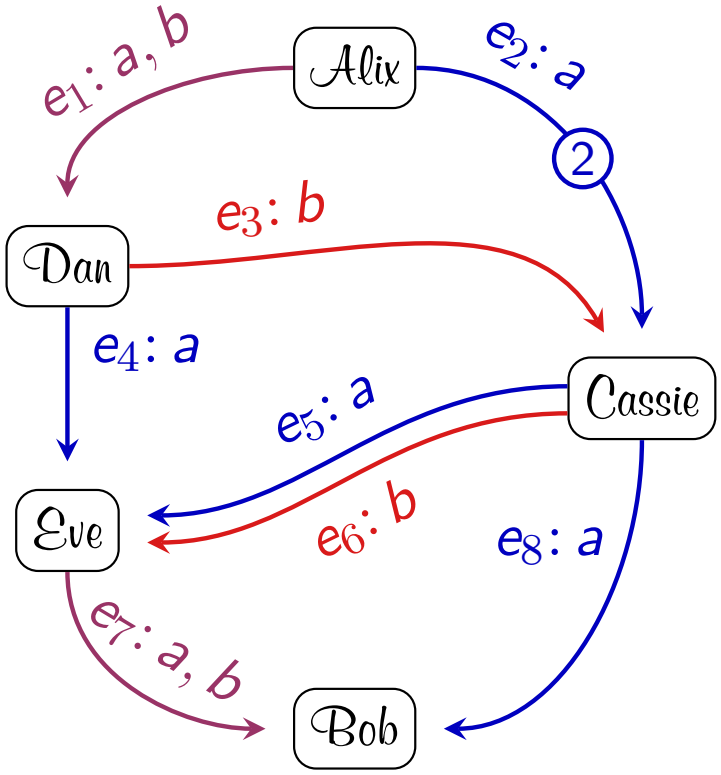
(up to exponential in $|D|$ and $|Q|$)

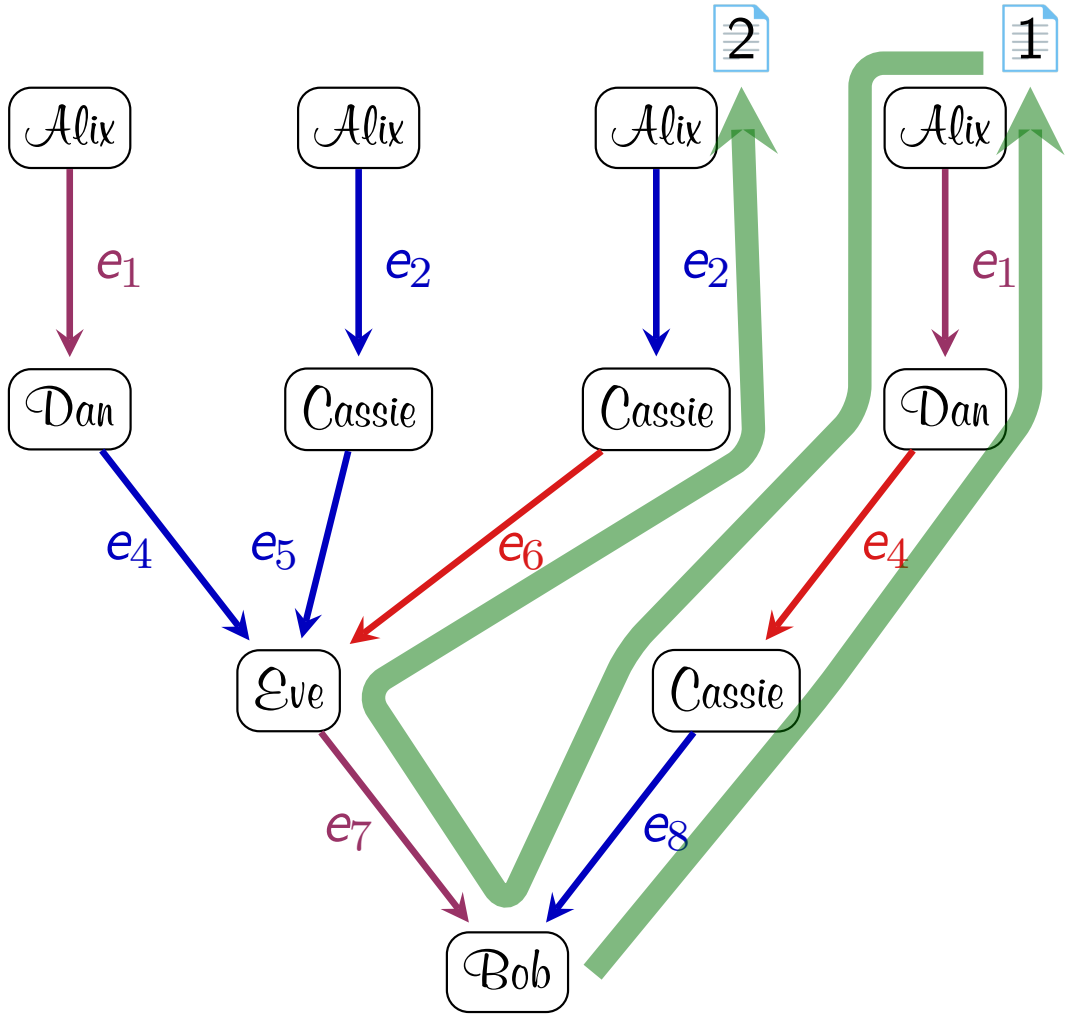
Enumeration strategy: DFS of the backward tree

Inputs:

$Q = b^* \cdot a \cdot (a + b)^*$

$s = \mathcal{A}lix \quad t = \mathcal{B}ob$





Backward tree of answers

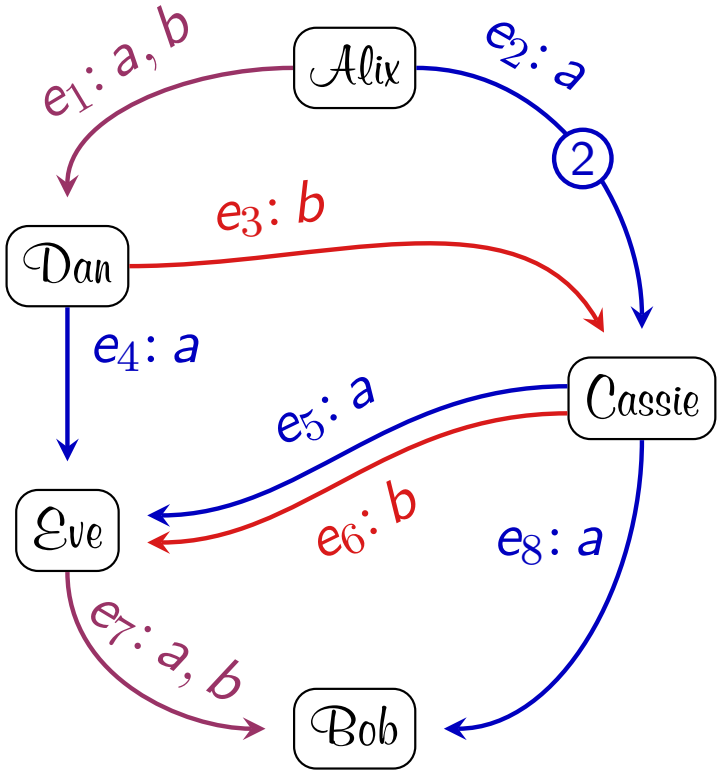
(up to exponential in $|D|$ and $|Q|$)

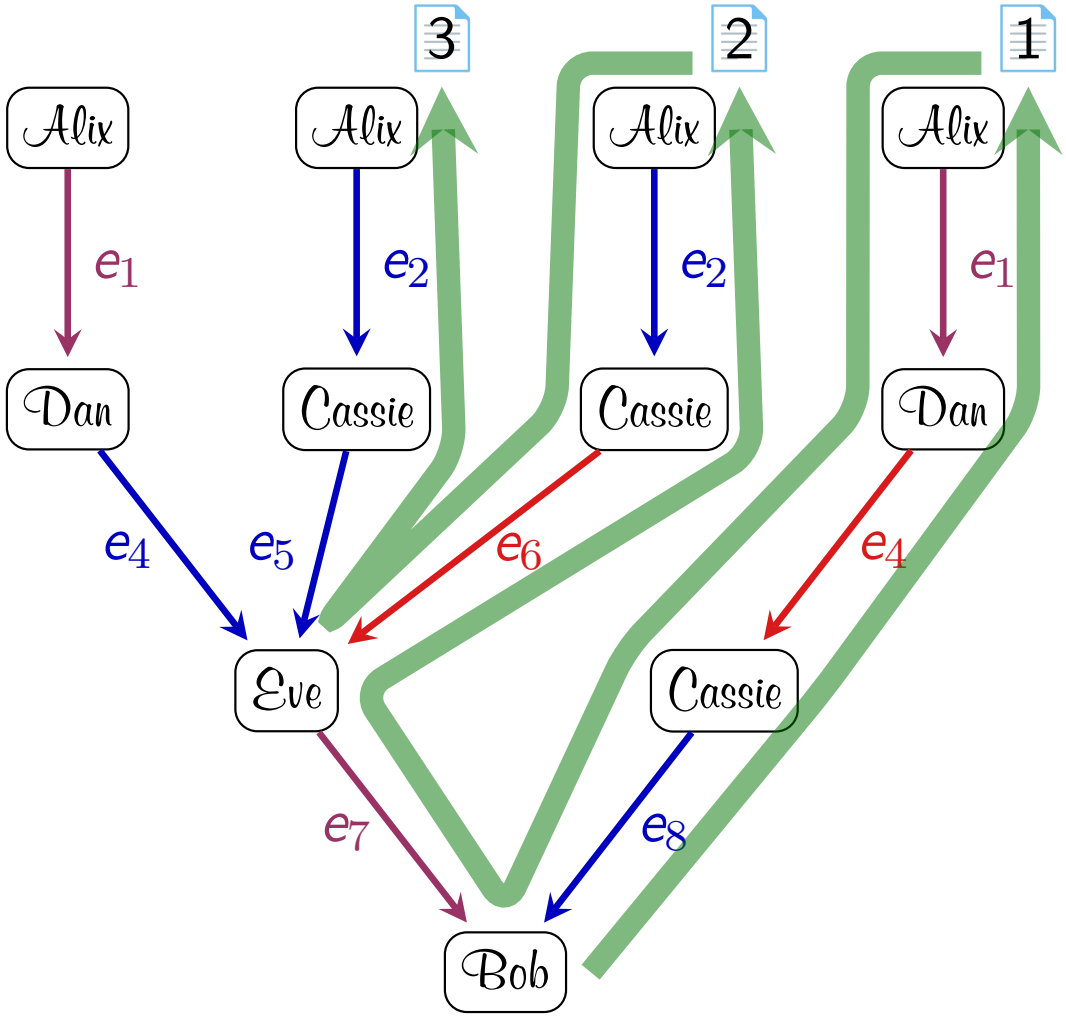
Enumeration strategy: DFS of the backward tree

Inputs:

$Q = b^* \cdot a \cdot (a + b)^*$

$s = \mathcal{A}lix \quad t = \mathcal{B}ob$





Backward tree of answers

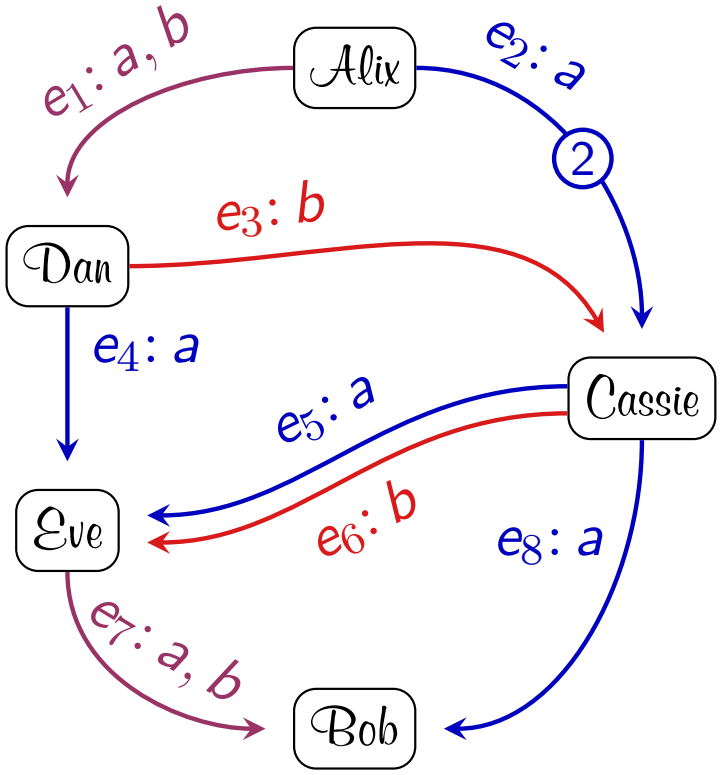
(up to exponential in $|D|$ and $|Q|$)

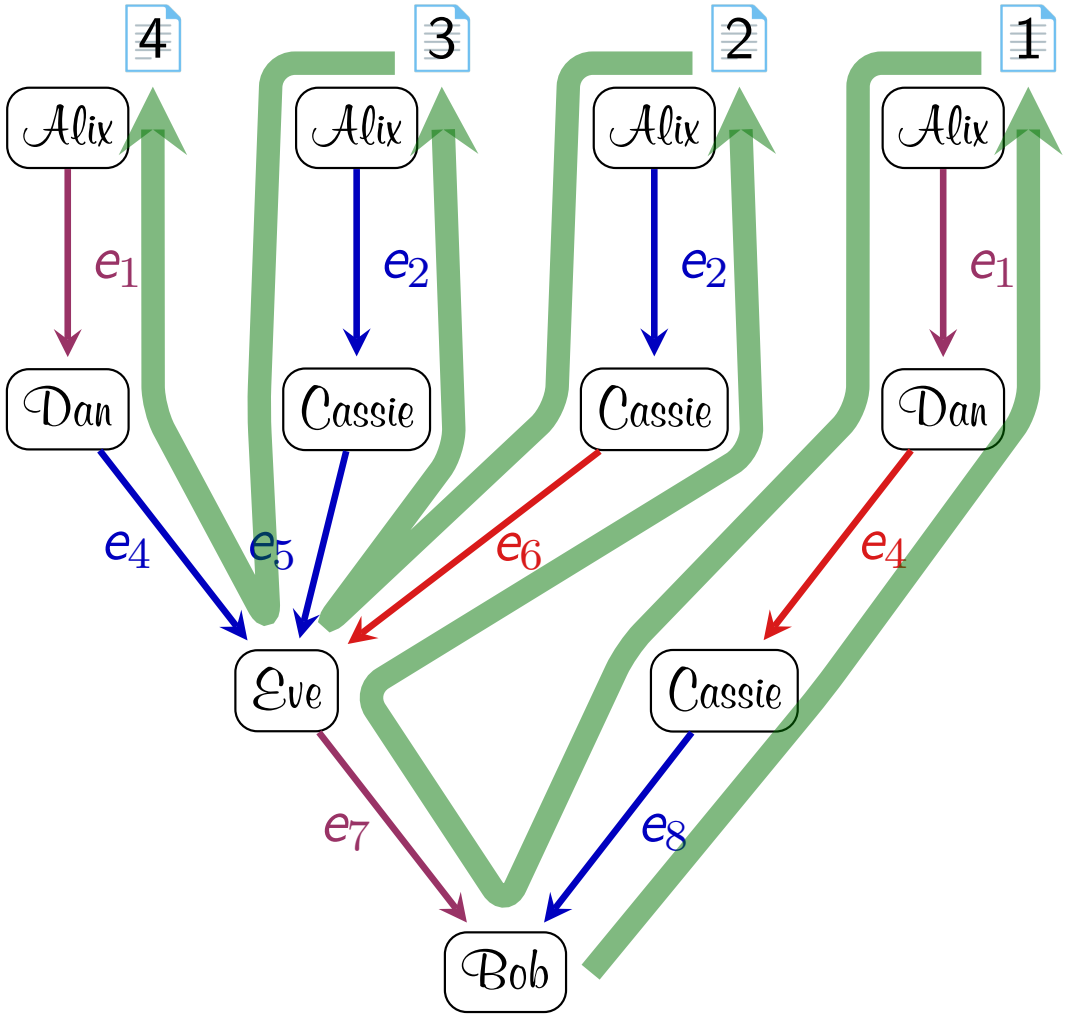
Enumeration strategy: DFS of the backward tree

Inputs:

$Q = b^* \cdot a \cdot (a + b)^*$

$s = \mathcal{A}lix \quad t = \mathcal{B}ob$





Backward tree of answers

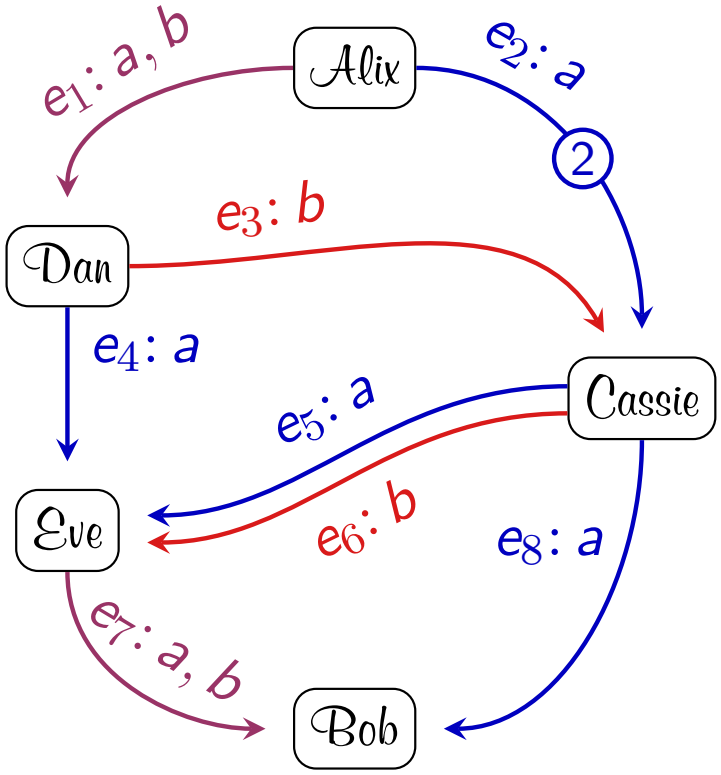
(up to exponential in $|D|$ and $|Q|$)

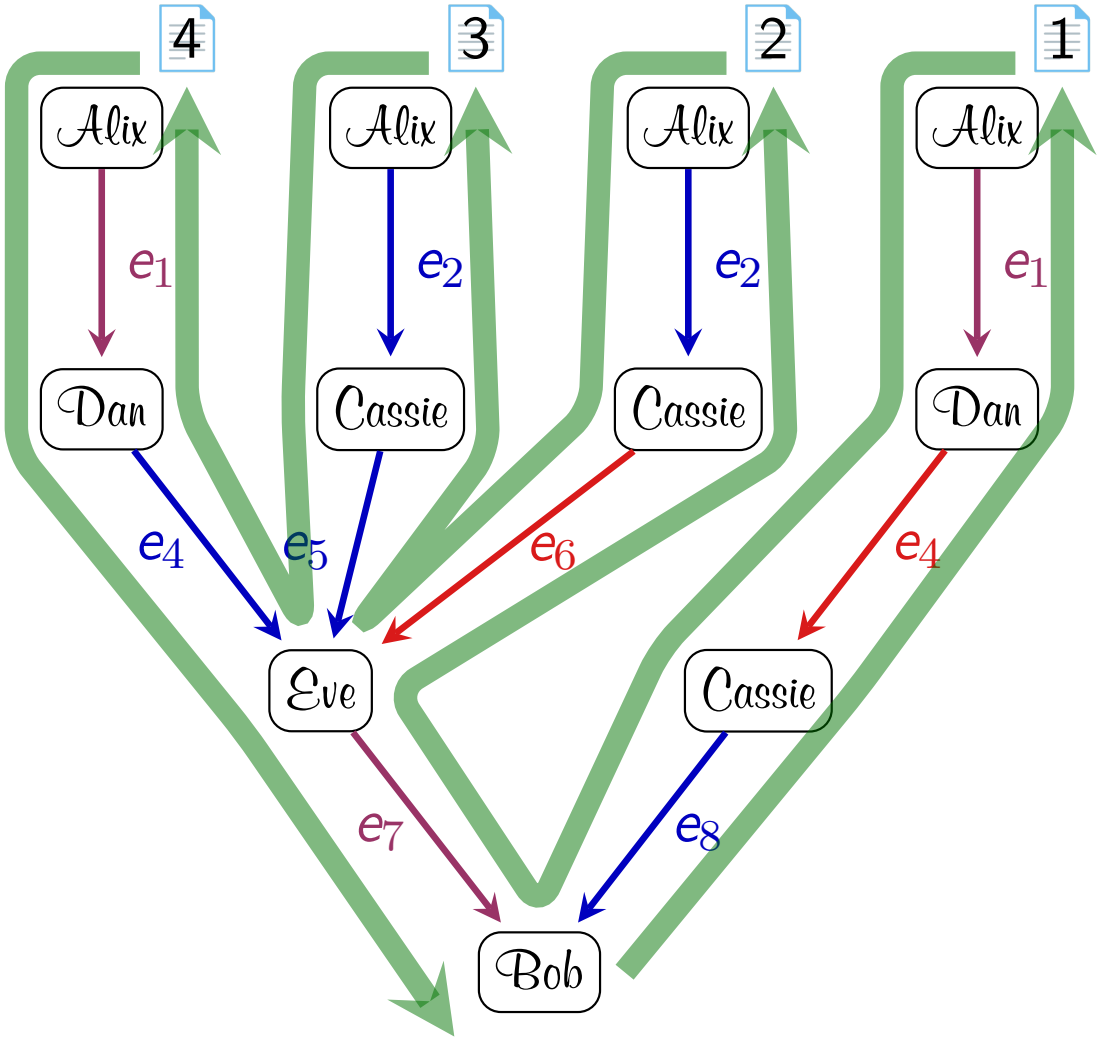
Enumeration strategy: DFS of the backward tree

Inputs:

$Q = b^* \cdot a \cdot (a + b)^*$

$s = \mathcal{A}lix \quad t = \mathcal{B}ob$





Backward tree of answers

(up to exponential in $|D|$ and $|Q|$)

Our algorithm in one slide

Preprocessing phase

- Annotate D using a BFS of product graph $D \times Q$
- Reindex the annotation to encode the backward tree

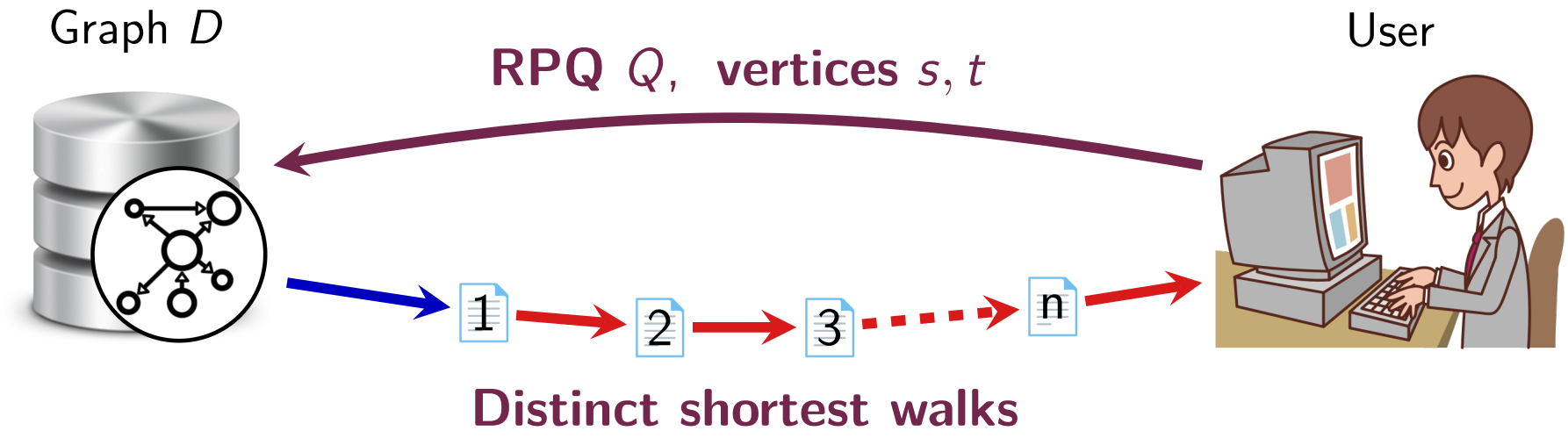
Enumeration phase

- DFS of the backward tree, computed on-the-fly
- No dead-ends
- Find next children independently of local topology
- Annotation is almost read-only

Carefull use of classical data structures

- "Sorting" in linear time via bucket-sort
- Shortcuts in the data structure via LinkedLists
- N-ary "zipper" to merge sorted lists

Our contribution

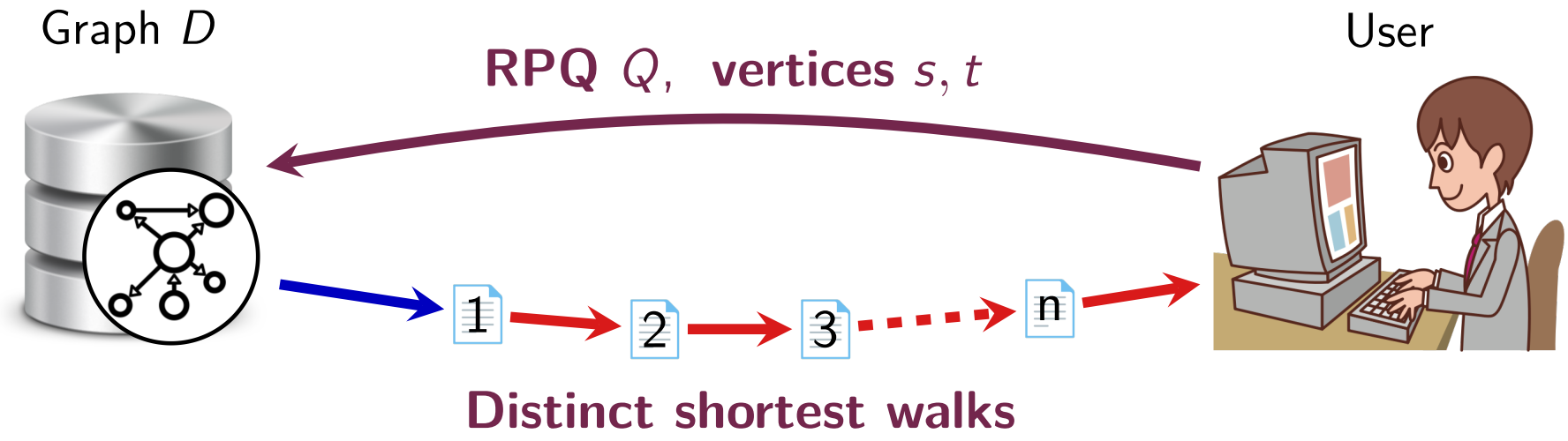


An algorithm to enumerate distinct shortest walks with

- $\mathbf{O}(|Q| \times |D|)$ preprocessing (time before first answer)
- $\mathbf{O}(|Q| \times |\lambda|)$ delay (time between two answers)
- $\mathbf{O}(|Q| \times |D|)$ memory usage

where λ is the length of one output.

Some perspectives



- Reducing overhead when nondeterminism does not occur
- Add other GQL features

- Are data structures well-behaved **in practice**?
- Gather information about nondeterminism in real-life settings