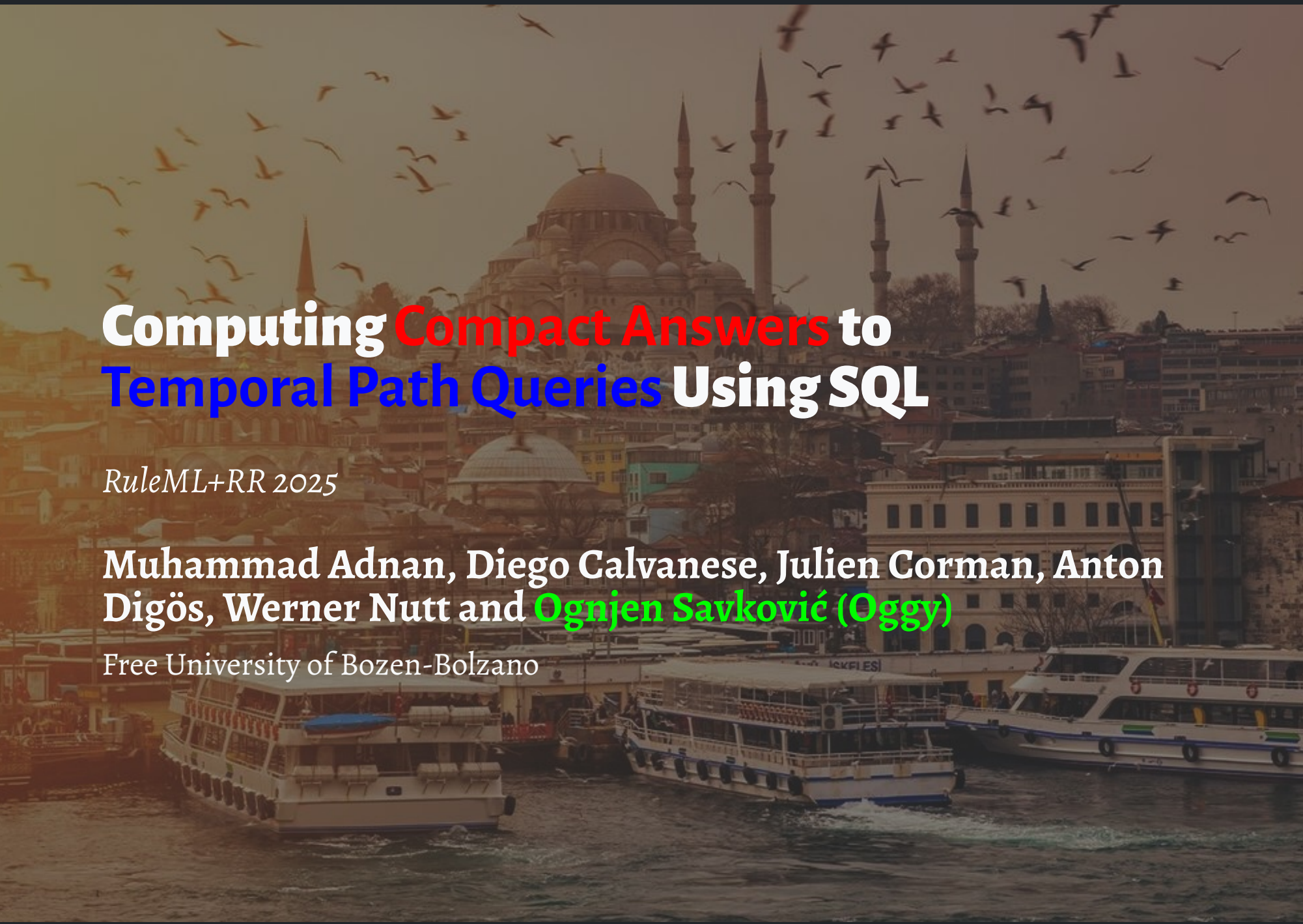


Computing Compact Answers to Temporal Path Queries Using SQL

RuleML+RR 2025

Muhammad Adnan, Diego Calvanese, Julien Corman, Anton Digös, Werner Nutt and Ognjen Savković (Oggy)

Free University of Bozen-Bolzano

General Motivation

Temporal Graphs = graphs with facts annotated by **time intervals**.

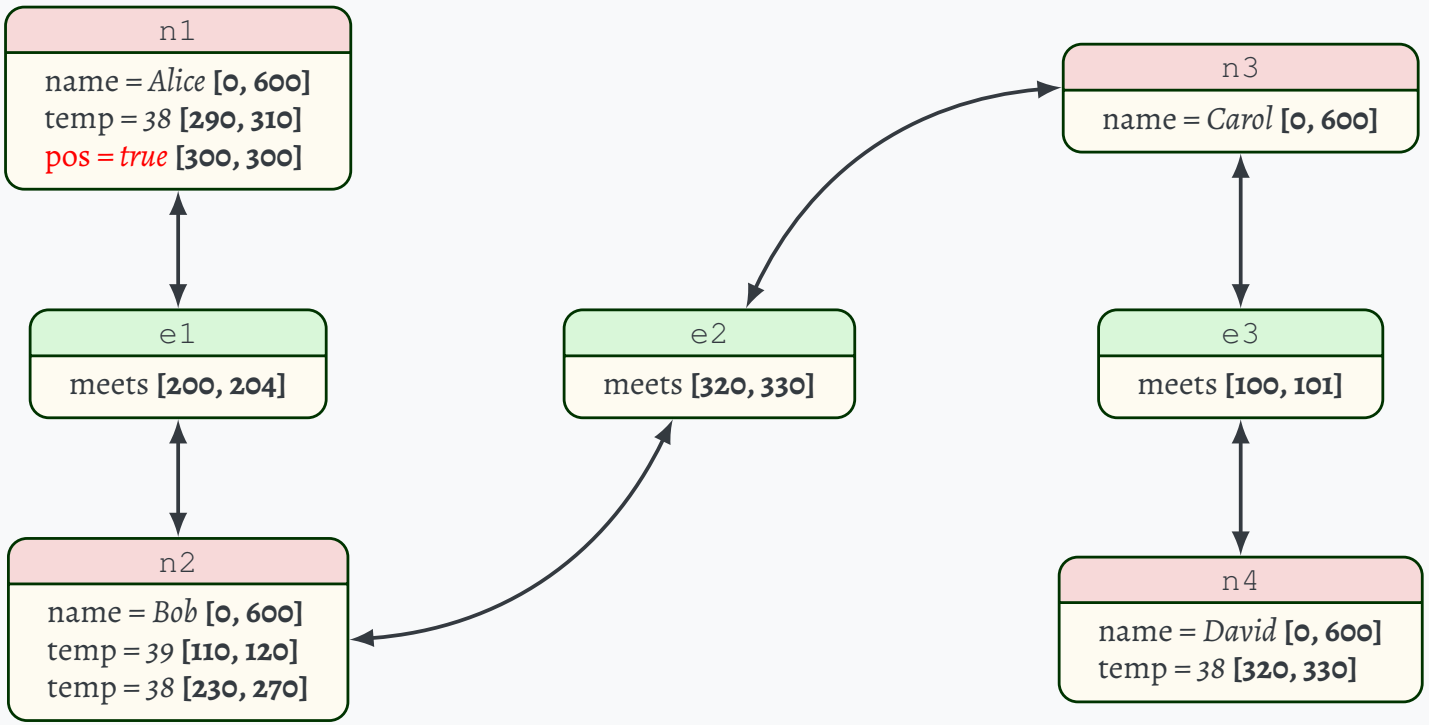
We want to **query phenomena over time** where there is a **relation** between objects/events in (possibly) **two different** time points:

- **Virus propagation**: *A patient has a temperature **now**, and he was visiting a sick friend **a week ago**.*
- **Uncertain travel times**: *If I start my trip **within two hours**, I can arrive at my destination in the **next five hours**.*
- **Error tracing**: ***Current** malfunction of a component in an airplane leads to the subsequent malfunction of another component **within an hour**.*

Goal: Query both **structure** and **time** simultaneously.

A Temporal Property Graph Example

- Here, $n_1, \ldots, n_4$ represent **nodes**, $e_1, \ldots, e_3$ represent direct **edges**
- Each *direct* and *labelled* edge is annotated with **time intervals**
- direct ~ *object properties*, labelled ~ *data properties*

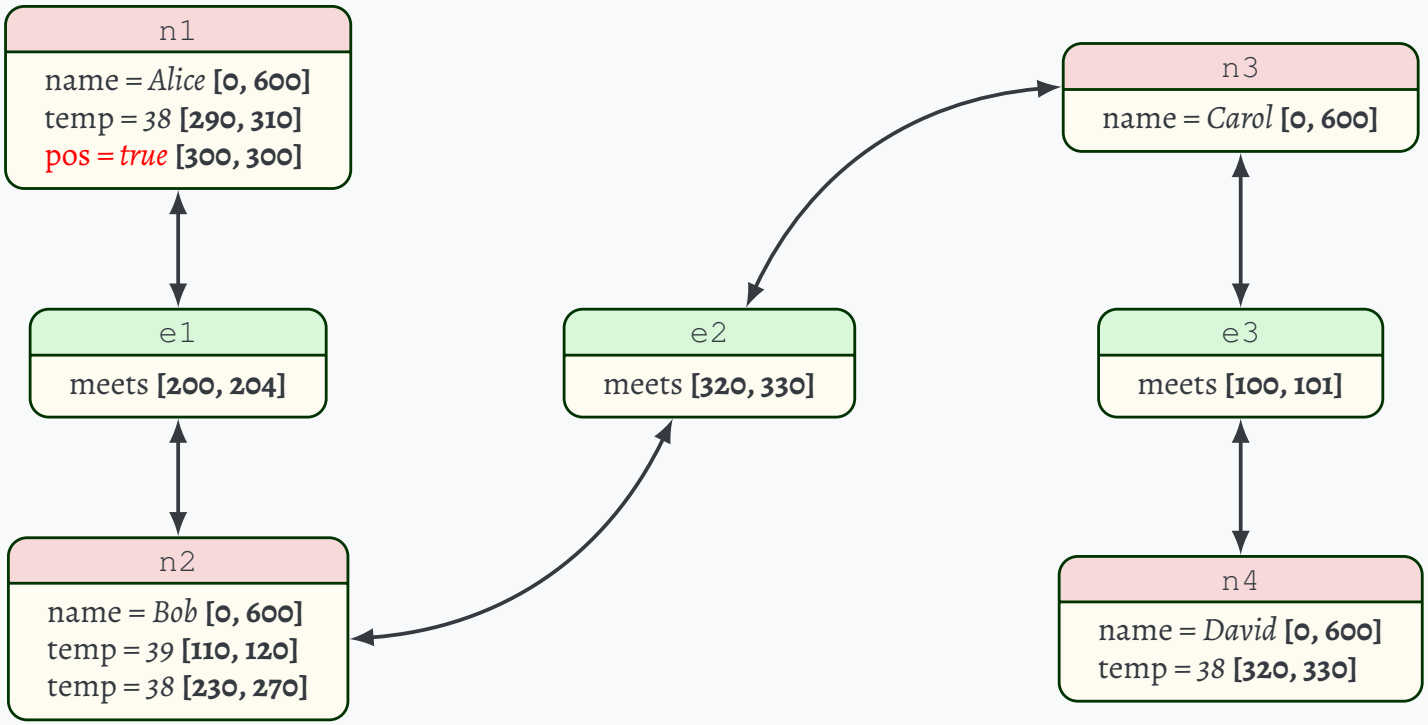


Time granularity is expressed in **hours**

Running Example: Temporal Regular Path Queries (TRPQs)

- **Example query** q_1: Find all pairs $\langle\langle o_1, t_1\rangle, \langle o_2, t_2\rangle\rangle$ such that
  - person o_1 tested positive at time t_1,
  - o_1 met o_2 within a week prior to t_1,
  - and o_2 had high temperature at time t_2, less than two days after the meeting

$$q_1 := (\texttt{pos} = \mathit{true})/\mathbf{T}_{[-168,0]}/\mathrm{F}/\texttt{meets}/\mathrm{F}/\mathbf{T}_{[0,48]}/(\texttt{temp} \geqslant 38)$$



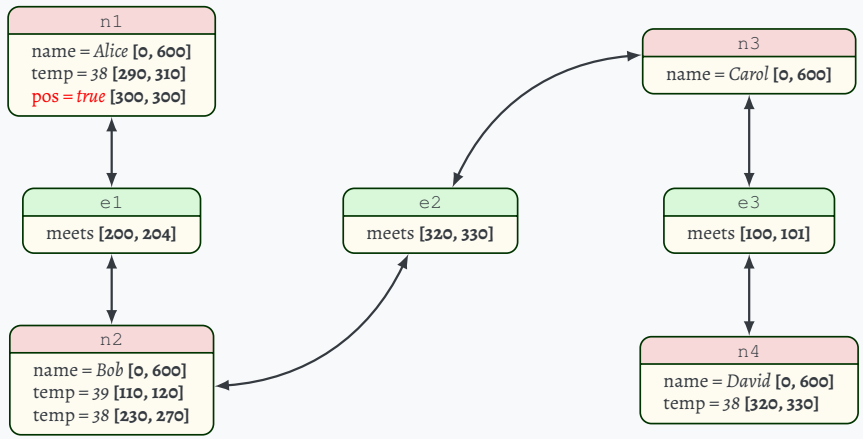
Here, operator F is needed since one can query an edge

Running Example: TRPQs /2

- **Example query q_2:** (A simpler query that we use to illustrate our problem)
  Return all Alices who have met someone in the last 2 to 3 hours

$$q_2 = (\texttt{name} = \textit{Alice})/\texttt{F}/\texttt{meets}/\texttt{T}_{[2,3]}/\texttt{F}$$

Start point		End point	
Object	Time	Object	Time
n_1	200	n_2	202
n_1	201	n_2	203
n_1	202	n_2	204
n_1	200	n_2	203
n_1	201	n_2	204



Running Example: Time per Distance representation

- Example query q_2: (A simpler query that we use to illustrate our problem)

$$q_2 = (\texttt{name} = \textit{Alice})/\texttt{F}/\texttt{meets}/\mathbf{T}_{[2,3]}/\texttt{F}$$

Start point		End point	
Object	Time	Object	Time
n_1	200	n_2	202
n_1	201	n_2	203
n_1	202	n_2	204
n_1	200	n_2	203
n_1	201	n_2	204

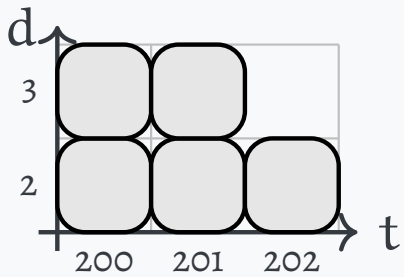
Start point		End point	
Object	Time	Object	Distance
n_1	200	n_2	2
n_1	201	n_2	2
n_1	202	n_2	2
n_1	200	n_2	3
n_1	201	n_2	3

Running Example: Time per Distance representation /2

- **Example query** q_2: (A simpler query that we use to illustrate our problem)

$$q_2 = (\texttt{name} = \textit{Alice})/\texttt{F}/\texttt{meets}/\texttt{T}_{[2,3]}/\texttt{F}$$

Start point		End point	
Object	Time	Object	Distance
n_1	200	n_2	2
n_1	201	n_2	2
n_1	202	n_2	2
n_1	200	n_2	3
n_1	201	n_2	3



TRPQ Language Recap

A **Temporal Regular Path Query** (TRPQ) is an expression:

$$\text{path} ::= \textit{pred} \mid \text{F} \mid \text{B} \mid \mathbf{T}_\delta \mid (\text{path}/\text{path}) \mid (\text{path} + \text{path}) \mid \text{path}[k, \_]$$

with $\delta \in \text{intv}(\mathbb{Z})$ and $k \in \mathbb{N}$.

Semantics of **time-shift** ($\mathbf{T}_\delta$) and **join** (/):

$$\begin{aligned}
[\![\mathbf{T}_\delta]\!]_G &= \{\langle o, o, t, d\rangle \mid o \in (N \cup E), t \in \mathcal{T}_G, d \in \delta\} \\
&\quad \textit{stay on the same object just move in time for distance } d \\
[\![\text{path}_1/\text{path}_2]\!]_G &= \{\langle o_1, o_3, t, d_1 + d_2\rangle \mid \langle o_1, o_2, t, d_1\rangle \in [\![\text{path}_1]\!]_G \\
&\quad \text{and } \langle o_2, o_3, t + d_1, d_2\rangle \in [\![\text{path}_2]\!]_G \text{ for some } o_2\} \\
&\quad \textit{join two paths where time points are “composable”}
\end{aligned}$$

Language originally defined by Arenas et. al. (ICDE 2022).
Here, we consider a core fragment important for time coalescing.

Challenge 1: How to compactly represents those answers

- How to **compactly** represents answers over such queries?
  - Left open in Arenas et. al. (ICDE 2022) that assumes **discrete time**
  - For dense time, **no finite representation** may exist

Observations:

- Traditional Temporal SQL (TSQL) can only represent **intervals**
- TSQL can **coalesce** intervals effectively
- TSQL does not have **time-shift** ($\mathbf{T}_\delta$)
- We want to coalesce a temporal **relation** (possible two dimensions)
  - Recall **Rectangle covering problem**:
    finding minimal coalescing of rectangles is **intractable**

We defined four Compact Representations

Non-compact:

- $\mathcal{U}$: all pairs of time per distance points

Two **compact** and **tractable**

- $\mathcal{U}^{\mathtt{t}}$: fold **time points** into intervals
- $\mathcal{U}^{\mathtt{d}}$: fold **distances** into intervals

Two **super-compact** but **intractable**

- $\mathcal{U}^{\mathtt{td}}$: fold **both** time points into intervals
- $\mathcal{U}^{\mathtt{s}}$: fold both and **cut** two *dents*

Non-compact and Four Compact Representations

Start point		End point	
Object	Time	Object	Time
n_1	200	n_2	202
n_1	201	n_2	203
n_1	202	n_2	204
n_1	200	n_2	203
n_1	201	n_2	204

Repr.	Start point		End point	
	Object	Time	Object	Dist.
$\mathcal{U}^{d}$	n_1	200	n_2	$[2, 3]$
	n_1	201	n_2	$[2, 3]$
	n_1	202	n_2	$[2, 2]$

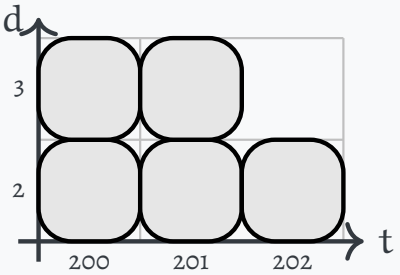
Repr.	Start point			End point
	Object	Time	Distance	Object
$\mathcal{U}^{t}$	n_1	$[200, 202]$	2	n_2
	n_1	$[200, 201]$	3	n_2

Repr.	Start point			End point
	Object	Time	Dist.	Object
$\mathcal{U}^{td}$	n_1	$[200, 201]$	$[2, 3]$	n_2
	n_1	$[202, 202]$	$[2, 2]$	n_2

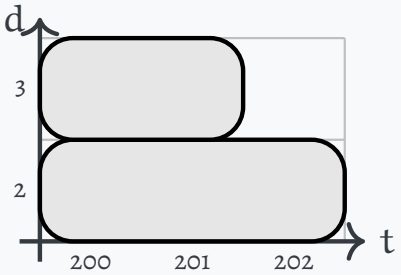
Repr.	Start point			End point	b	e
	Object	Time	Dist.	Object		
$\mathcal{U}^{c}$	n_1	$[200, 202]$	$[2, 3]$	n_2	200	201

Non-compact and Four Compact Representations

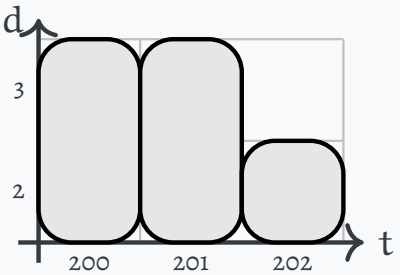
$\mathcal{U}$:



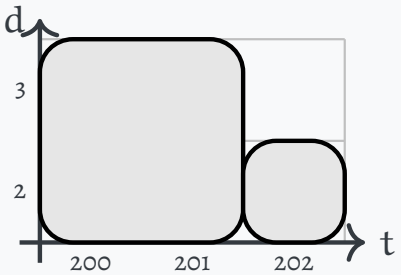
$\mathcal{U}^{t}$:



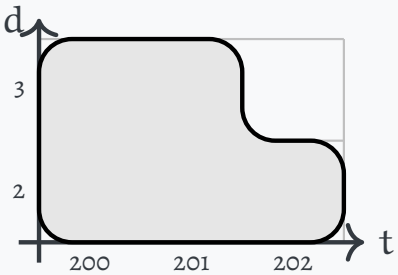
$\mathcal{U}^{d}$:

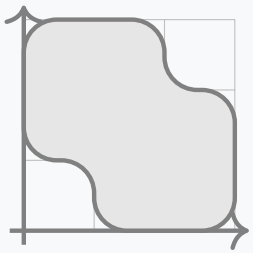


$\mathcal{U}^{td}$:

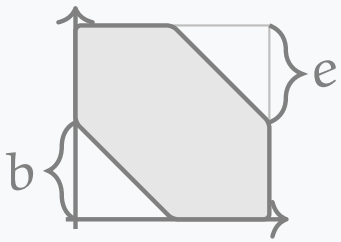


$\mathcal{U}^{c}$:





discrete



dense time

Main Challenge

- **Efficient implementation** for our **four representations**

In this work, we **show** two-step approach for implementing $\mathcal{U}^{\mathrm{t}}$ and $\mathcal{U}^{\mathrm{d}}$:

1. **Theory**: characterize TRPQ operators in two folded repr. $\mathcal{U}^{\mathrm{t}}$ and $\mathcal{U}^{\mathrm{d}}$
2. **Implementation**: translate to **Temporal SQL** such that
   - each operand is implemented following **characterizations**
   - do coalescing via Temporal SQL **Window** functions

Still **open**:

- How to implement coalescing **efficiently** for $\mathcal{U}^{\mathrm{td}}$ and $\mathcal{U}^{\mathrm{c}}$
- Implementing **Kleene star** `[k,_]` efficiently
  - one needs an implementation technique with **a fix-point operator**

For **full characterizations** of all four repr. see our **ISWC 2025** paper (**Julien** will present it at ISWC)

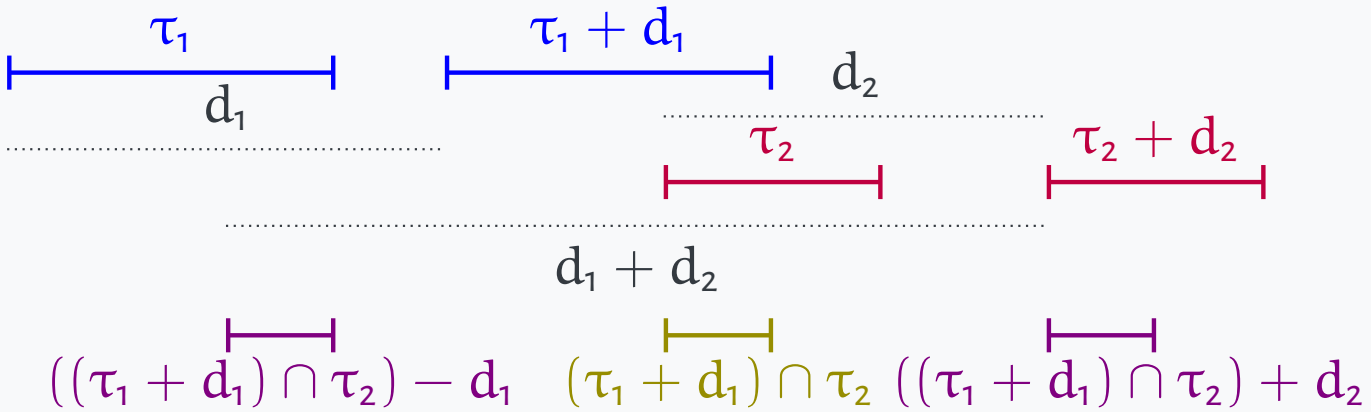
Two-step approach to implementation

Step 1: Characterization in $\mathcal{U}^t$

We (only) show **join**

$$(\!|\mathrm{path}_1/\mathrm{path}_2|\!)^t_G = \Big\{ \langle o_1, o_3, ((\tau_1 + d_1) \cap \tau_2) - d_1, d_1 + d_2 \rangle \mid \langle o_1, o_2, \tau_1, d_1 \rangle \in (\!|\mathrm{path}_1|\!)^t_G \text{ and } \langle o_2, o_3, \tau_2, d_2 \rangle \in (\!|\mathrm{path}_2|\!)^t_G \text{ and } (\tau_1 + d_1) \cap \tau_2 \neq \emptyset \text{ for some } o_2 \Big\}$$

Time calculation graphically:



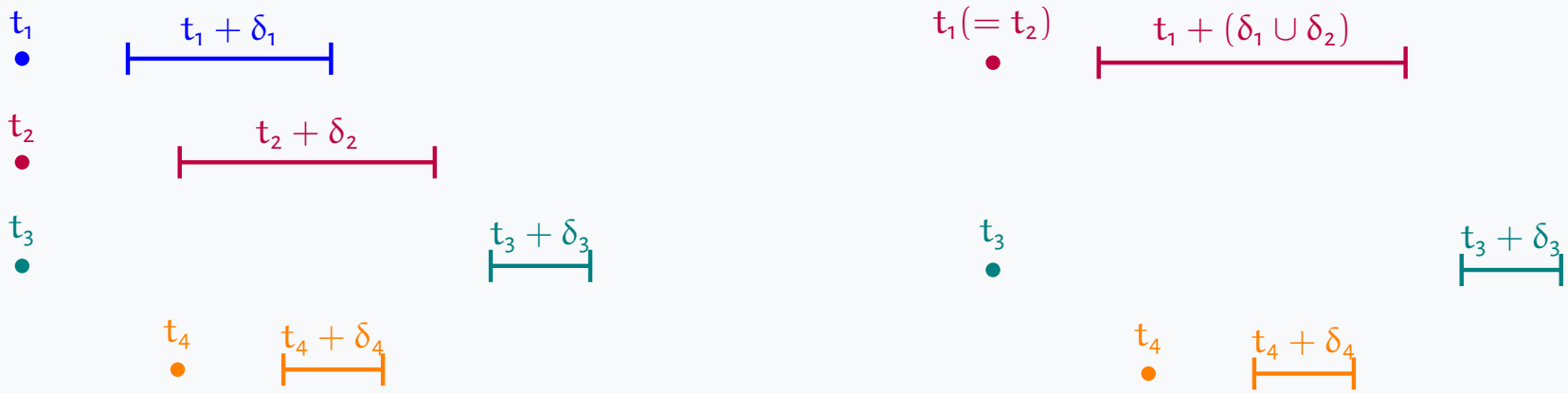
Other TRPQs operands we **characterize** similarly, and the same for $\mathcal{U}^d$.

Coalescing Challenge

After each step we apply **coalescing** of time for both U^t and U^d

- **Coalescing** = merge overlapping intervals to regain compactness
- **Essential** for performance

Simple example of before and after coalescing in U^d



Step2: SQL Implementation

1. Represent nodes/edges in **base tables**
   - For each **operation** we create an **intermediate** result table
   - For **local predicates** extract matching nodes/edges
   - For a **join** we create a join table as defined by the characterizations
2. After each operation we **coalesce** intervals
   - we use PostgreSQL **window** functions
   - based on the **state-of-the-art** techniques from the temporal databases
   - runs in $\mathcal{O}(n \log n)$

Experiments

Experimental Setup

We run a **parametrized** query:

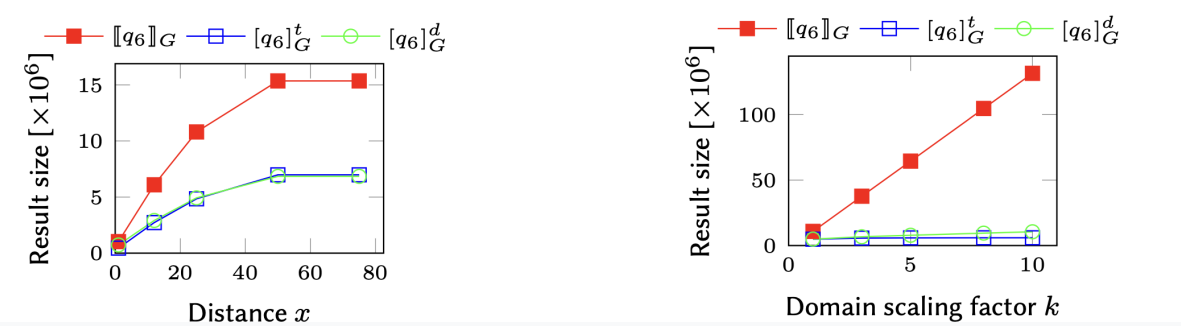
$$q_6 := (\texttt{pos} = \textit{true})/\mathbf{T}_{[-x,0]}/\texttt{F}/\texttt{meets}$$

- x increases the **distance** interval $[-x, 0]$ in q_6,
- Graph G consists of 24, 990 nodes and 2, 638, 623 edges
- and the **scaling factor** k that multiplies the size of intervals in G.

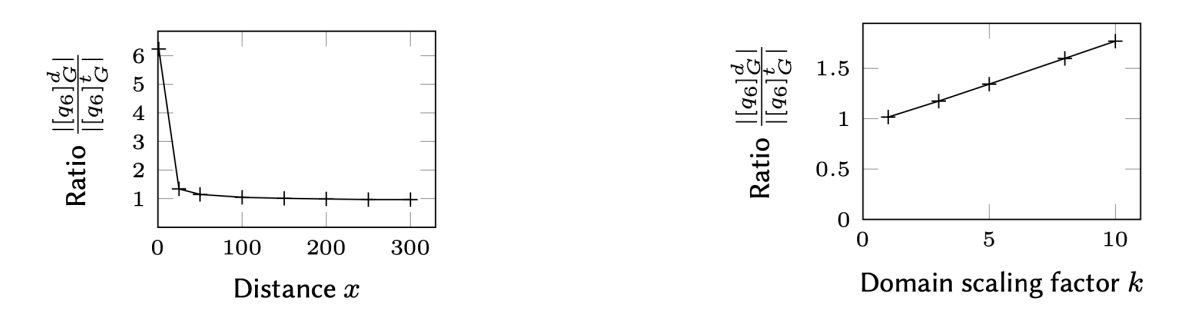
We tested:

- how **compact query answers** can be in $\mathcal{U}$, $\mathcal{U}^t$,
- how input interval sizes in graph and query impact **compact answer size**,
- how much **coalescing** $(\!|q|\!)^t_G$ and $(\!|q|\!)^d_G$ improves compactness.

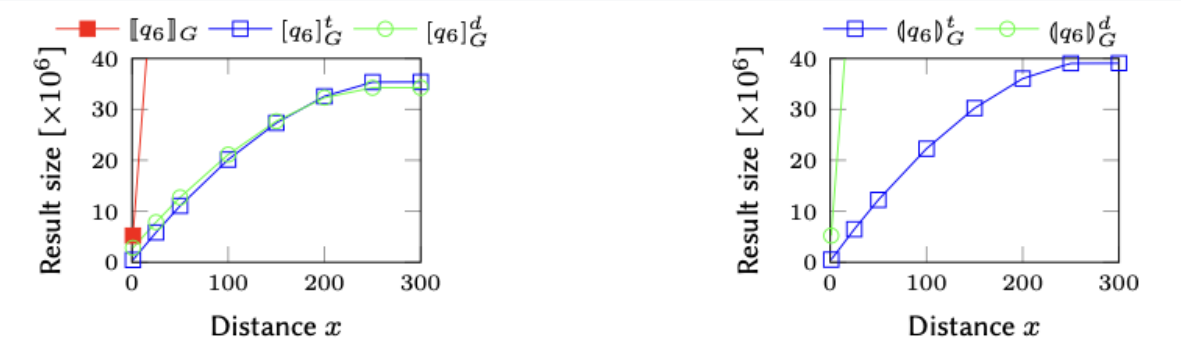
Experimental Results



Result sizes for varying x and fixed $k = 1$ (left), or varying k and fixed $x = 25$ (right)



Ratio $|[q_6]^d_G|$ over $|[q_6]^t_G|$ for varying x and fixed $k = 5$ (left), or varying k and fixed $x = 25$



Result sizes for varying x and fixed $k = 5$, with coalescing (left) and without (right)

Experimental Results/2

Conclusions:

- $\mathcal{U}_t$ more compact for large graph intervals
- $\mathcal{U}_d$ more compact for large query intervals
- **Coalescing** drastically reduces result size and runtime

Conclusions and Future Work

- **Compact answers** and **efficient evaluation** achieved for two representations
- **Coalescing** essential for performance
- Open problems: **Kleene star**, two more **complex representations**
- Future: query **optimization** techniques and test on **real-world datasets**

Code: `github.com/osavkovic/CompactTRPQ`

Teşekkür ederim!