🌰 Instance-Optimized String Fingerprints

Mihail Stoian*, Johannes Thürauf*, Andreas Zimmerer,
Alexander van Renen, Andreas Kipf

Data Systems Lab x Discrete Optimization Lab @UTN

@AIDB'25, September 1st, 2025

UTN Technische Universität Nürnberg

Data Pruning

	salary	location
min:	100K	Barcelona
max:	200K	Paris
min:	200K	Madrid
max:	300K	Redmond

Data Pruning

WHERE salary = 250K

	salary	location	
min:	100K	Barcelona	❌ (skip)
max:	200K	Paris	
min:	200K	Madrid	🔍 (maybe)
max:	300K	Redmond	

Data Pruning

❄️ WHERE location LIKE ‘B%’

		salary	location	
	min:	100K	Barcelona	✅ (match)
	max:	200K	Paris	
	min:	200K	Madrid	❌ (skip)
	max:	300K	Redmond	

Data Pruning

WHERE location LIKE '%ch%'

	salary	location	
min:	100K	Barcelona	🔍 (maybe)
max:	200K	Paris	
min:	200K	Madrid	🔍 (maybe)
max:	300K	Redmond	

Data Pruning

WHERE location LIKE ‘%ch%’

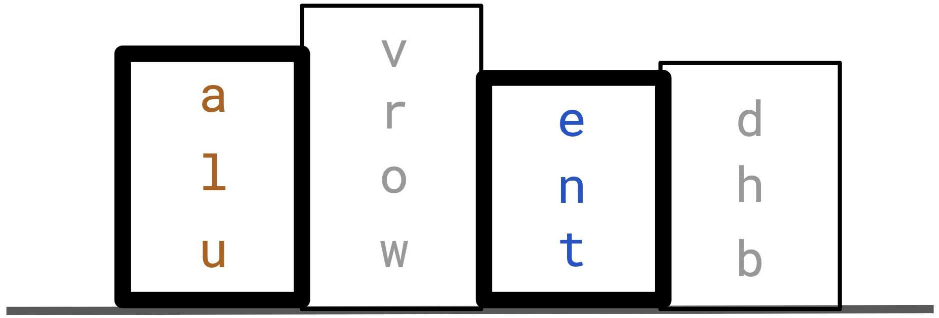
	salary	location	
min:	100K	Barcelona	🔍 (maybe)
max:	200K	Paris	
min:	200K	Madrid	🔍 (maybe)
max:	300K	Redmond	

String Fingerprints

1. **Partition** the alphabet in a fixed number of bins.
2. Compute a **bitmask** of bin indices.

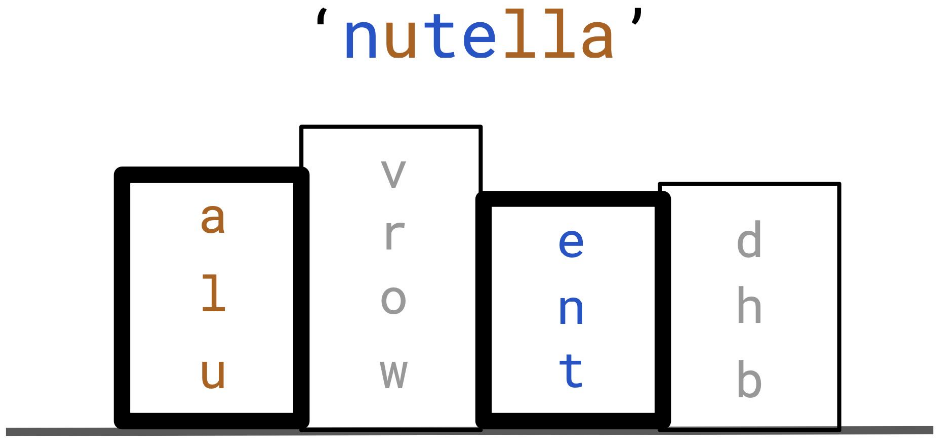
String Fingerprints

1. **Partition** the alphabet in a fixed number of bins.
2. Compute a **bitmask** of bin indices.



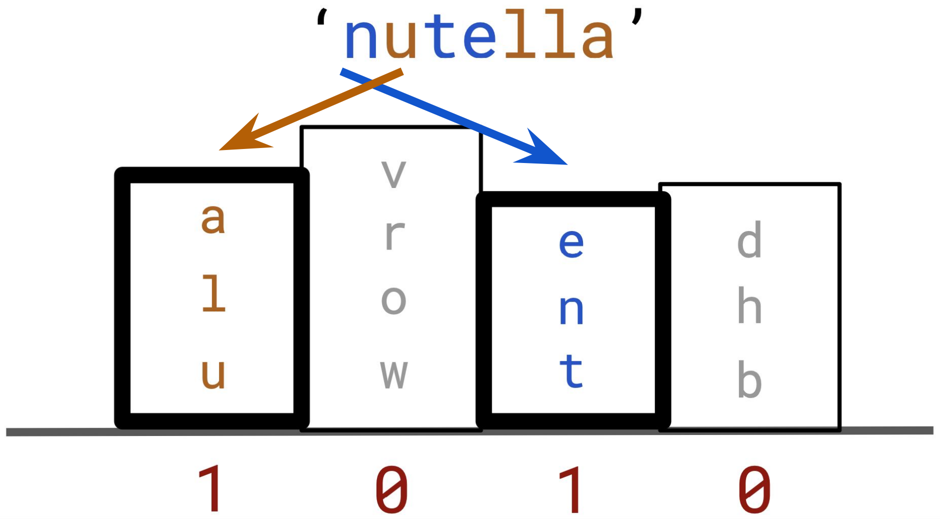
String Fingerprints

1. **Partition** the alphabet in a fixed number of bins.
2. Compute a **bitmask** of bin indices.



String Fingerprints

1. **Partition** the alphabet in a fixed number of bins.
2. Compute a **bitmask** of bin indices.



Application

- Lightweight secondary index for `LIKE` predicates *with false positives*.

Application

- Lightweight secondary index for `LIKE` predicates *with false positives*.

language	symbol	spelling
🇬🇧	🍫	nutella
🇷🇴	🧈	unt
🇫🇷	🐟	thon

Application

- Lightweight secondary index for `LIKE` predicates *with false positives*.

language	symbol	spelling	str_fp
🇬🇧	🍫	nutella	1010
🇷🇴	🧈	unt	1010
🇫🇷	🐟	thon	0111

Application

- Lightweight secondary index for `LIKE` predicates *with false positives*.

  Example: WHERE `spelling` LIKE ‘%utn%’.

language	symbol	spelling	str_fp
🇬🇧	🍫	nutella	1010
🇷🇴	🧈	unt	1010
🇫🇷	🐟	thon	0111

Application

- Lightweight secondary index for `LIKE` predicates *with false positives*.

  Example: WHERE `spelling` LIKE '%utn%' ⇒ `1010`.

language	symbol	spelling	str_fp
🇬🇧	🍫	nutella	1010
🇷🇴	🧈	unt	1010
🇫🇷	🐟	thon	0111

Application

- Lightweight secondary index for `LIKE` predicates *with false positives*.

  Example: WHERE `spelling` LIKE ‘%utn%’ ⇒ `1010`.

language	symbol	spelling	str_fp
🇬🇧	🍫	nutella	1010
🇷🇴	🧈	unt	1010
~~🇫🇷~~	~~🐟~~	~~thon~~	~~0111~~

$1010 \subsetneq 0111$

Application

- Lightweight secondary index for `LIKE` predicates *with false positives*.

  Example: WHERE `spelling` LIKE ‘%utn%’ ⇒ `1010`.

language	symbol	spelling	str_fp
🇬🇧	🍫	nutella	1010
🇷🇴	🧈	unt	1010
~~🇫🇷~~	~~🐟~~	~~thon~~	~~0111~~

$1010 \not\subseteq 0111$

Optimal Partitioning

- Intuition: Minimize the number of wasted LIKE evaluations.
- Example: WHERE `spelling` LIKE ‘%utn%’ (`1010`) ⇒ 2 false positives.

language	symbol	spelling	str_fp
🇬🇧	🍫	nutella	1010
🇷🇴	🧈	unt	1010

Optimal Partitioning

- Intuition: Minimize the number of wasted LIKE evaluations.
- Example: WHERE `spelling` LIKE '%utn%' (`1010`) ⇒ 2 false positives.

⇒ Objective: *Minimize the number of false positives.*

`language`	`symbol`	`spelling`	`str_fp`
🇬🇧	🍫	`nutella`	`1010`
🇷🇴	🧈	`unt`	`1010`

Mixed-Integer Program

- Words **W**: The string column.
- Queries **Q**: The patterns in the workload.
- Ground truth `f(q)`: The words that match query `q`.

Mixed-Integer Program

- Words **W**: The string column.
- Queries **Q**: The patterns in the workload.
- Ground truth `f(q)`: The words that match query `q`.
- Program:
  1. Encode the partitioning: $x_{a,i} = 1$, if letter `a` in bin `i`.

Mixed-Integer Program

- Words **W**: The string column.
- Queries **Q**: The patterns in the workload.
- Ground truth `f(q)`: The words that match query `q`.
- Program:
    1. Encode the partitioning: $x_{a,i}$ = 1, if letter **a** in bin **i**.
    2. Encode the fingerprint: $d_{s,i}$ = 1, if string **s** has a letter in bin **i**.

Mixed-Integer Program

- Words **W**: The string column.
- Queries **Q**: The patterns in the workload.
- Ground truth `f(q)`: The words that match query `q`.
- Program:
  1. Encode the partitioning: $x_{a,i}$ = 1, if letter **`a`** in bin **`i`**.
  2. Encode the fingerprint: $d_{s,i}$ = 1, if string **`s`** has a letter in bin **`i`**.
  3. Encode a false positive: $\eta_{w,q}$ = 1, if the partitioning correctly tells apart whether query **`q`** is *not* contained in word **`w`**.

Mixed-Integer Program

- Words **W**: The string column.
- Queries **Q**: The patterns in the workload.
- Ground truth `f(q)`: The words that match query `q`.
- Program:
  1. Encode the partitioning: $x_{a,i} = 1$, if letter `a` in bin `i`.
  2. Encode the fingerprint: $d_{s,i} = 1$, if string `s` has a letter in bin `i`.
  3. Encode a false positive: $\eta_{w,q} = 1$, if the partitioning correctly tells apart whether query `q` is *not* contained in word `w`.
- Objective: $\max \Sigma_{q \in Q} \Sigma_{w \in W \setminus f(q)} \eta_{w,q}$.
- Constraints: Details in the paper.

Evaluation

- Setup: Column `title.title` in IMDb dataset (2.37M tuples; no UTF-8).
- *Workload*:
  - 300 queries ⇒ 10 high-, mid-, low-frequency {1, …, 10}-grams from the column.
  - Split into:
    - 20 “**seen**” queries & 280 “**unseen**” queries.

Evaluation

- Setup: Column `title.title` in IMDb dataset (2.37M tuples; no UTF-8).
- *Workload*:
  - 300 queries ⇒ 10 high-, mid-, low-frequency {1, …, 10}-grams from the column.
  - Split into:
    - 20 “**seen**” queries & 280 “**unseen**” queries.
- *Data*:
  a. **Full** table.
  b. The **1st data block** (= 2^{16} tuples).
  c. 50-tuple **sample** from the 1st data block.

⇒ MIP is optimized on **seen** queries x 50-tuple **sample**.

Evaluation

- Setup: Column `title.title` in IMDb dataset (2.37M tuples; no UTF-8).
- *Workload*:
  - 300 queries ⇒ 10 high-, mid-, low-frequency {1, …, 10}-grams from the column.
  - Split into:
    - 20 "**seen**" queries & 280 "**unseen**" queries.
- *Data*:
  a. **Full** table.
  b. The **1st data block** (= 2^{16} tuples).
  c. 50-tuple **sample** from the 1st data block.

⇒ MIP is optimized on **seen** queries x 50-tuple **sample**.

- Bit-widths ∈ {4, 8, 16}-bit.

Evaluation

- Setup: Column `title.title` in IMDb dataset (2.37M tuples; no UTF-8).
- *Workload*:
  - 300 queries ⇒ 10 high-, mid-, low-frequency {1, …, 10}-grams from the column.
  - Split into:
    - 20 “**seen**” queries & 280 “**unseen**” queries.
- *Data*:
  a. **Full** table.
  b. The **1st data block** (= 2^{16} tuples).
  c. 50-tuple **sample** from the 1st data block.

⇒ MIP is optimized on **seen** queries x 50-tuple **sample**.

- Bit-widths ∈ {4, 8, 16}-bit.
- Baseline: Round-robin placement of letters into bins.

Evaluation: False Positive Rate

MIP Optimization

Queries • Data: ○ seen • 1st block sample □ seen • 1st block ✚ seen • table △ *unseen* • 1st block ✖ *unseen* • table

Evaluation: False Positive Rate

MIP Optimization

Queries • Data: ○ seen • 1st block sample □ seen • 1st block ✚ seen • table △ *unseen* • 1st block ✖ *unseen* • table

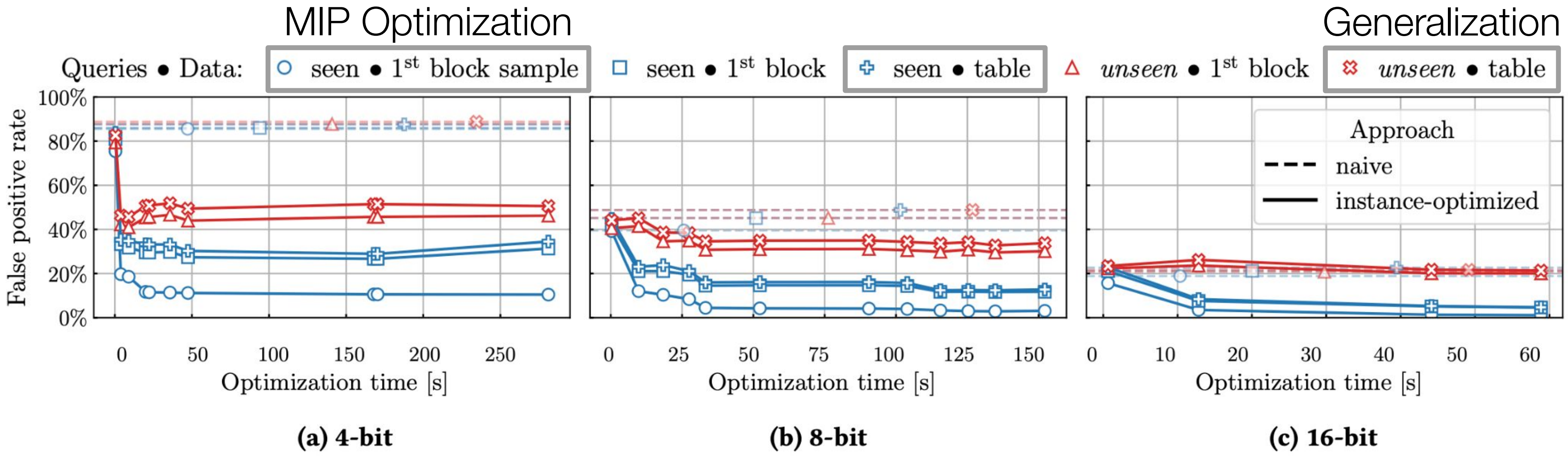
Evaluation: False Positive Rate

MIP Optimization

Generalization

Queries • Data: ○ seen • 1st block sample □ seen • 1st block ✚ seen • table △ *unseen* • 1st block ✖ *unseen* • table

Evaluation: False Positive Rate



(a) 4-bit

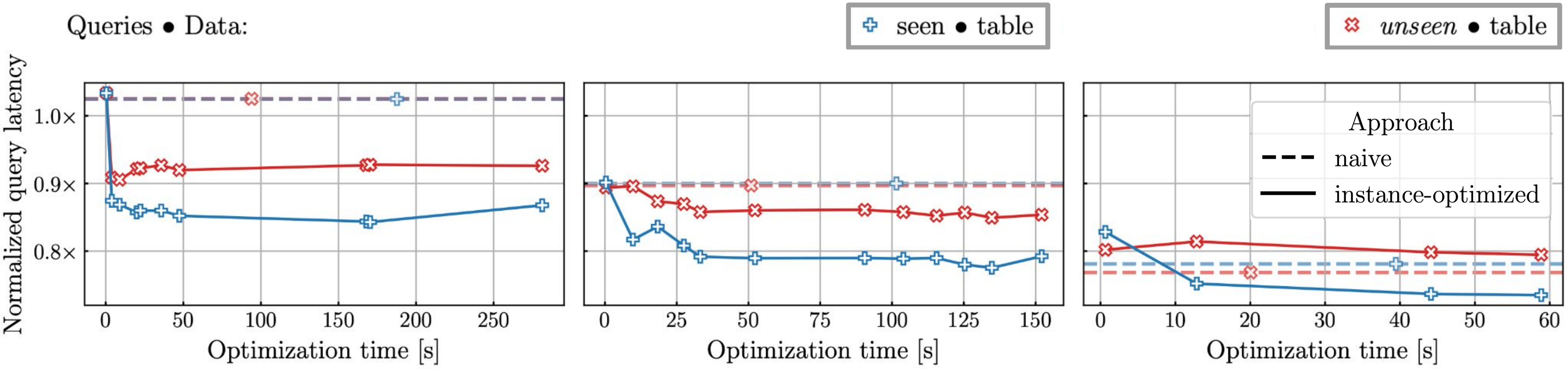
(b) 8-bit

(c) 16-bit

Evaluation: Normalized Query Latency

- Note: Run on the *full* table.

Generalization



(a) 4-bit

(b) 8-bit

(c) 16-bit

Evaluation: Takeaways

- <20% false positive rate on the full table.
- 🪄 Generalization to *unseen* queries (unlike predicate caching).
- 🚀 Up to 1.36x speedup for seen queries & 1.26x speedup for unseen queries.

(Many) Future Work Directions

- Instead of 1-grams, i.e., letters ⇒ Why not 2-/3-grams?
  - Intuition: We can capture the *order* of the letters.
- String zonemaps:
  - String fingerprints enable pruning for infix predicates 😲.
- String cardinality estimation.
  - Take the supersets and sum up their corresponding cardinality.
- Table clustering:
  - Sort by the fingerprint.

Wanna More Cool Research?

See you **tmrw** in *Research 8* (**Westminster 4F**), 1.45pm - 3.15pm!

“Parachute: Single-Pass Bi-Directional Information Passing”

🚀 **1.54x** speedup over `duckdb`