# FDB: A Query Engine for Factorised Relational Databases

Nurzhan Bakibayev and Dan Olteanu and Jakub Závodný

Department of Computer Science, University of Oxford, Oxford, OX1 3QD, UK

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF OXFORD

## Factorised Relational Databases

### Compact representations that reduce data redundancy and boost query performance

- Algebraic factorisations of relational data using distributivity of product over union
- Can be exponentially more succinct than the relations they encode
- Allow for constant-delay enumeration of tuples, unlike join decompositions and trivial representation (Q,D)

PlaysFor

| player | team |
|---|---|
| Messi | Barcelona |
| Villa | Barcelona |
| Cech | Chelsea |
| Torres | Chelsea |
| van Persie | Arsenal |

CompetesIn

| team | league |
|---|---|
| Barcelona | Primera |
| Barcelona | Champions |
| Chelsea | Premier |
| Chelsea | Champions |
| Arsenal | Premier |

LeagueStadium

| league | stadium |
|---|---|
| Primera | CampNou |
| Champions | CampNou |
| Champions | Wembley |
| Premier | Stamford |
| Premier | Wembley |

$Q_1$ = PlaysFor $\bowtie_{\text{team}}$ CompetesIn $\bowtie_{\text{league}}$ LeagueStadium

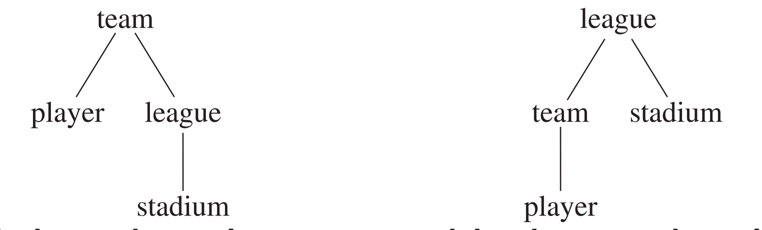
| player | team | league | stadium |
|---|---|---|---|
| Messi | Barcelona | Primera | CampNou |
| Messi | Barcelona | Champions | CampNou |
| Messi | Barcelona | Champions | Wembley |
| Villa | Barcelona | Primera | CampNou |
| Villa | Barcelona | Champions | CampNou |
| | | … | |

Two examples of **factorised representations** of the above query result:

$$\begin{aligned}&\langle\text{Barcelona}\rangle \times (\langle\text{Messi}\rangle \cup \langle\text{Villa}\rangle)\times\\ &\quad\times (\langle\text{Primera}\rangle \times \langle\text{CampNou}\rangle\cup\\ &\qquad\langle\text{Champions}\rangle \times (\langle\text{CampNou}\rangle \cup \langle\text{Wembley}\rangle))\cup\\ &\langle\text{Chelsea}\rangle \times (\langle\text{Cech}\rangle \cup \langle\text{Torres}\rangle)\times\\ &\quad\times ((\langle\text{Premier}\rangle \times (\langle\text{Stamford}\rangle \cup \langle\text{Wembley}\rangle))\cup\\ &\qquad\langle\text{Champions}\rangle \times (\langle\text{CampNou}\rangle \cup \langle\text{Wembley}\rangle))\cup\\ &\langle\text{Arsenal}\rangle \times \langle\text{van Persie}\rangle\times\\ &\quad\times ((\langle\text{Premier}\rangle \times (\langle\text{Stamford}\rangle \cup \langle\text{Wembley}\rangle)).\end{aligned}$$

$$\begin{aligned}&\langle\text{Primera}\rangle \times \langle\text{Barcelona}\rangle \times (\langle\text{Messi}\rangle \cup \langle\text{Villa}\rangle) \times \langle\text{CampNou}\rangle\cup\\ &\langle\text{Champions}\rangle \times (\langle\text{Barcelona}\rangle \times (\langle\text{Messi}\rangle \cup \langle\text{Villa}\rangle)\cup\\ &\qquad\langle\text{Chelsea}\rangle \times (\langle\text{Cech}\rangle \cup \langle\text{Torres}\rangle))\times\\ &\qquad\times (\langle\text{CampNou}\rangle \cup \langle\text{Wembley}\rangle)\cup\\ &\langle\text{Premier}\rangle \times (\langle\text{Chelsea}\rangle \times (\langle\text{Cech}\rangle \cup \langle\text{Torres}\rangle)\cup\\ &\qquad\langle\text{Arsenal}\rangle \times \langle\text{van Persie}\rangle)\times\\ &\qquad\times (\langle\text{Stamford}\rangle \cup \langle\text{Wembley}\rangle).\end{aligned}$$

**Factorisation trees** (f-trees) describe the nesting structure of the above factorisations:




Left: First group by team and then by players and independently by leagues with stadiums.
Right: First group by league and then by teams with players and independently by stadiums.

## Size of Factorised Representations of Query Results

For any conjunctive (aka select-project-join) query $Q$, there is a number $s(Q)$ such that

For any database **D**, there is a factorised representation of $Q(D)$ with size $O(|\mathbf{D}|^{s(Q)})$.

This is the **best possible bound** for factorisations inferred from $Q$, without looking at **D**.
How to compute the parameter $s(Q)$?

- Iterate over all f-trees inferred from $Q$
- For each f-tree $\mathcal{T}$, for each root-to-leaf path $p$ in $\mathcal{T}$, compute:
  - the **fractional edge cover number** of the hypergraph of the sub-query defined by $p$,
  - the maximum such number $s(\mathcal{T})$ over all paths in $\mathcal{T}$
- Take $s(Q)$ as the minimum $s(\mathcal{T})$ over all f-trees $\mathcal{T}$ of $Q$

## Publications


- **On Factorisation of Provenance Polynomials** D. Olteanu, J. Závodný. In *TaPP*, 2011.
- **Factorised Representations of Query Results.** D. Olteanu, J. Závodný. In *ICDT*, 2012.
- **FDB: A Query Engine for Factorised Relational Databases** N. Bakibayev, D. Olteanu, J. Závodný. In *PVLDB* 5(11):1232-1243, 2012.
- **Demonstration of the FDB Query Engine for Factorised Databases** N. Bakibayev, D. Olteanu, J. Závodný. In *PVLDB* 5(12), 2012.


## Query Evaluation

Any query can be evaluated by a sequential composition of operations called **factorisation plan** $f = \omega_1, \dots, \omega_k$ that performs the following sequence of transformations:

$$\mathcal{T}_{\text{initial}} = \mathcal{T}_0 \overset{\omega_1}{\mapsto} \mathcal{T}_1 \overset{\omega_2}{\mapsto} \dots \overset{\omega_k}{\mapsto} \mathcal{T}_k = \mathcal{T}_{\text{final}}$$

and is defined by the following operators on f-trees.

**Restructuring: Normalisation Operator**
- factors out expressions common to all terms of a union.
- all our operators preserve normalisation.

**Restructuring: Swap Operator** $\chi_{\mathcal{A},\mathcal{B}}$
- exchanges a node $\mathcal{B}$ with its parent node $\mathcal{A}$ in the input f-tree $\mathcal{T}$ while preserving normalisation of $\mathcal{T}$.

**Cartesian Product** $\times$
- simply concatenates the input representations.

**Merge (Absorb) Join Operator** $\mu_{\mathcal{A},\mathcal{B}}$ ($\alpha_{\mathcal{A},\mathcal{B}}$)
- executes selection condition $\mathcal{A} = \mathcal{B}$ if $\mathcal{A}$ and $\mathcal{B}$ are sibling nodes (respectively, $\mathcal{A}$ is an ancestor of $\mathcal{B}$) in $\mathcal{T}$.

**Projection Operator** $\pi_{-A}$
- projects away the attribute $A$, if $A$ is a leaf in $\mathcal{T}$.

The operators need **quasilinear time in the data input and output sizes**. The evaluation time for $f$ is $O(|\mathbf{D}|^{s(f)} \cdot \log|\mathbf{D}|)$, where $s(f) = \max(s(\mathcal{T}_0), s(\mathcal{T}_1), \dots, s(\mathcal{T}_k))$.

## Query Optimisation

Two optimisation objectives (in this order):

1. find a **factorisation plan with minimal cost**, and
2. find a **small factorisation of the query result**.

Cost based on asymptotic bounds (i.e., $s(f)$) or estimates.
Search space defined by the order of join, swap, and projection operators. Two optimisers:

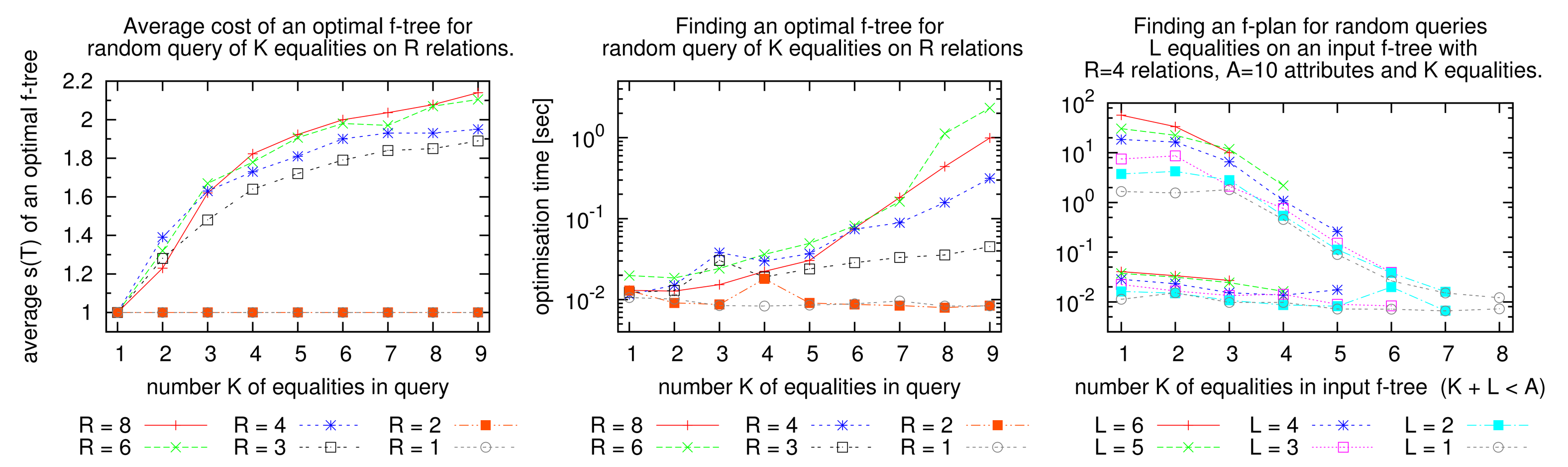
1. Exhaustive/full search
2. Greedy search: always choose the cheapest operator.

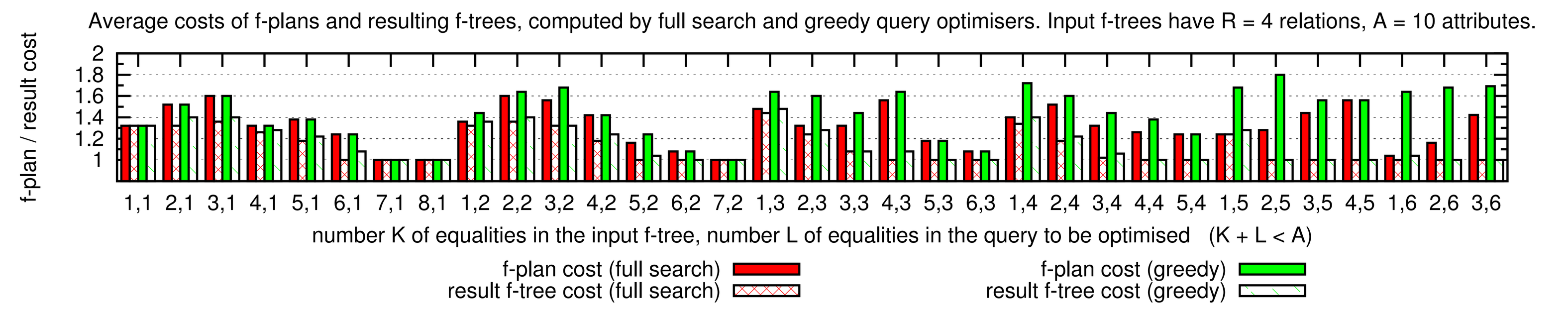
## Applications

- **Succinct representation of large query results**
- **Knowledge compilation in relational databases**
  - Compile data into compact factorised form
  - Speed up processing of many subsequent queries
- **Natural fit for large search spaces**
  - AND/OR trees used in design specification
  - World-set decompositions for incomplete data
  - Configuration problems in constraint satisfaction
- **Factorised provenance polynomials**
  - Compact encoding for provenance information
  - Efficient query evaluation in probabilistic databases

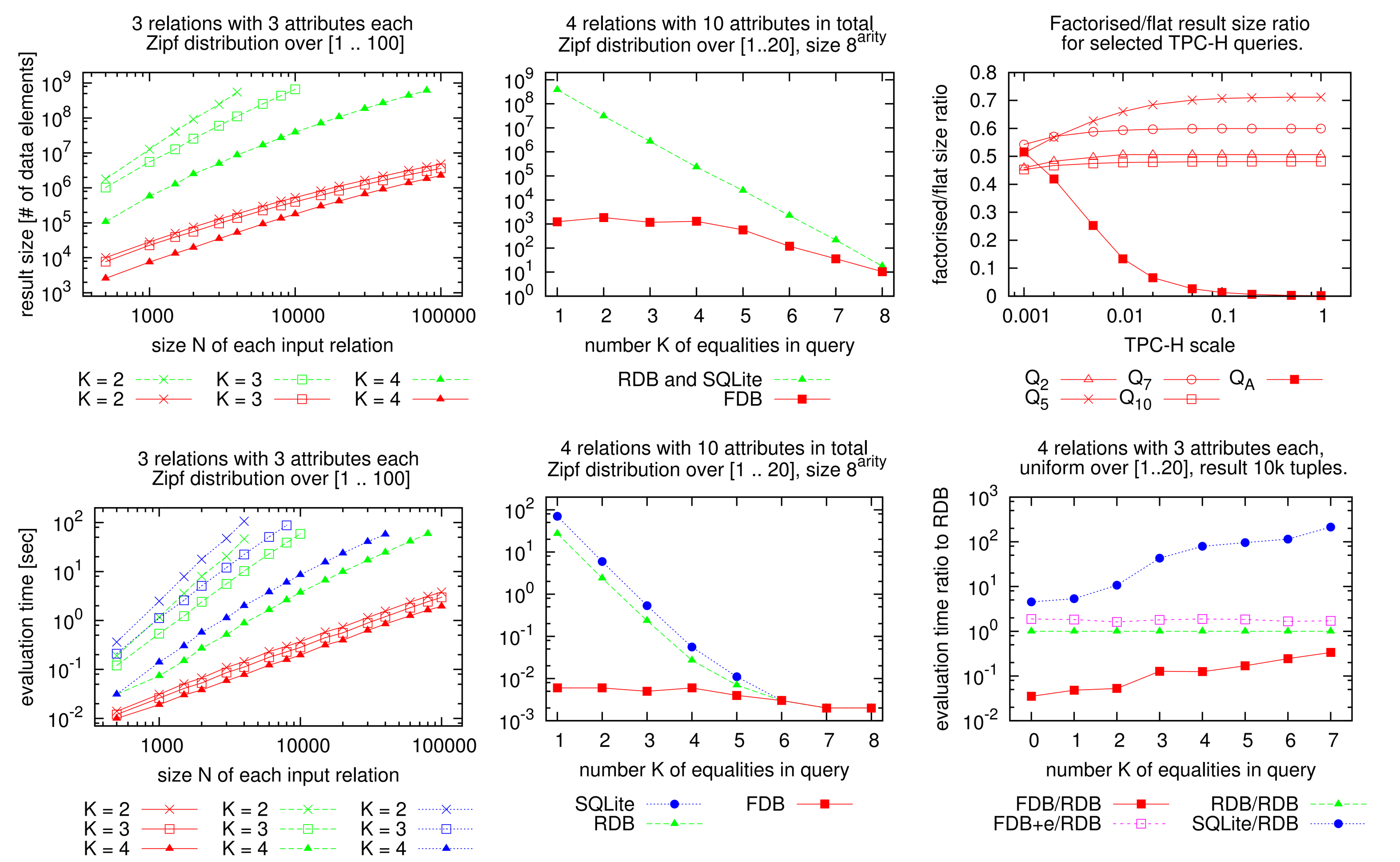
## Experiments: Query Optimisation

**Full search** (slower, top series) vs **greedy** (faster, bottom series):

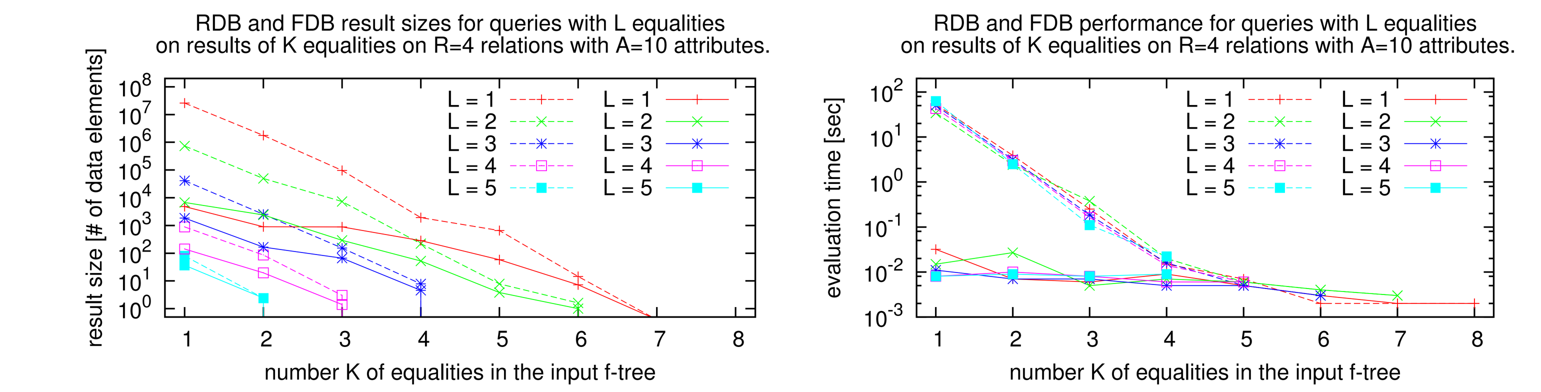



## Experiments: Query Evaluation on Flat Relational Data




## Experiments: Query Evaluation on Factorised Data




FDB/RDB: solid/dashed lines, bottom/top series in the right plot
RDB (lightweight purpose-built relational engine):

- is three times faster than SQLite.
- works on flat data equivalent to the input factorised data.
- needs one scan over the input, while FDB needs restructuring.