Compact encoding for provenance information

◮ Factorised provenance polynomials

PVLDB
Compact encoding for provenance information
◮ Factorised provenance polynomials

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

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)

Player team

(left) Player team

(team) League/stadium

(right) League/stadium

Two examples of factorised representations of the above query result:

(Barcelona) × (CampNou) × (Wembley)
(Chelsea) × (Wembley)
(Barcelona) × (CampNou)
(Villa Barcelona)
(Czech)
(Chelsea)
(Anfield)

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

FDB: A Query Engine for Factorised Relational Databases

Query Evaluation

Any query can be evaluated by a sequential composition of operations called factorisation plan \( f = \cup_{k=1}^{\infty} f_k \) that performs the following sequence of transformations:

\[
T_{\text{init}} = T_1 \cap T_2 \cap \ldots \cap T_n = T_{\text{final}}
\]

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 \( s_{A,B}(f) \)
◮ exchanges a node \( s \) with its parent node \( \mu \) in the input f-tree \( \tau \) while preserving normalisation of \( \tau \).

Cartesian Product
◮ simply concatenates the input representations.

Merge (Absorb) Join Operator \( j_{A,B}(f) \)
◮ executes selection condition \( A = B \) if \( A \) and \( B \) are sibling nodes (respectively, \( \mu \) is an ancestor of \( s \) in \( \tau \)).

Projection Operator \( \pi_A(f) \)
◮ projects away the attribute \( A \) if \( A \) is a leaf in \( \tau \).

The operators need quasilinear time in the data input and output sizes. The evaluation time for \( f \) is:

\[
O(D^{k+1} \log D_{\mu}), \quad \text{where} \quad s(f) = \max\{s(T_1), s(T_2), \ldots, s(T_n)\}
\]

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

FDB: A Query Engine for Factorised Relational Databases

WWW: http://www.cs.ox.ac.uk/projects/FDB/