Factorized databases I

Jakub Zavodny (University of Oxford, UK)

DATA ANALYSIS AND MODELING LAB
Palacky University, Olomouc, Czech Republic

INVESTMENTS IN EDUCATION DEVELOPMENT

Factorised Databases 1.

Factorisation of Relations

Jakub Závodný, University of Oxford

based on joint work with PhD supervisor Dan Olteanu

Relational Databases

Data is arranged in a table.

chain	country
Tesco	Czech Republic
Tesco	Slovakia
Tesco	Hungary
Tesco	Poland
Tesco	UK
Tesco	Spain
Lidl	Czech Republic
Lidl	Slovakia
Lidl	Hungary
Lidl	Poland
Lidl	UK
Aldi	Hungary
Aldi	Poland
Aldi	UK
Aldi	Spain

Relational Databases

A table is modelled by a relation $\quad HasShop \subseteq Chains \times Countries$

chain	country
Tesco	Czech Republic
Tesco	Slovakia
Tesco	Hungary
Tesco	Poland
Tesco	UK
Tesco	Spain
Lidl	Czech Republic
Lidl	Slovakia
Lidl	Hungary
Lidl	Poland
Lidl	UK
Aldi	Hungary
Aldi	Poland
Aldi	UK
Aldi	Spain

Relation = a set of tuples

Relational Databases

Relation = a set of tuples = union of products of singletons.

$$\begin{aligned}\mathrm{HasShop} = {} & \langle \mathit{Tesco} \rangle \times \langle \mathit{CzechRep} \rangle \cup \\ & \langle \mathit{Tesco} \rangle \times \langle \mathit{Slovakia} \rangle \cup \\ & \langle \mathit{Tesco} \rangle \times \langle \mathit{Hungary} \rangle \cup \\ & \langle \mathit{Tesco} \rangle \times \langle \mathit{Poland} \rangle \cup \\ & \langle \mathit{Tesco} \rangle \times \langle \mathit{UK} \rangle \cup \\ & \langle \mathit{Tesco} \rangle \times \langle \mathit{Spain} \rangle \cup \\ & \langle \mathit{Lidl} \rangle \times \langle \mathit{CzechRep} \rangle \cup \\ & \langle \mathit{Lidl} \rangle \times \langle \mathit{Slovakia} \rangle \cup \\ & \langle \mathit{Lidl} \rangle \times \langle \mathit{Hungary} \rangle \cup \\ & \langle \mathit{Lidl} \rangle \times \langle \mathit{Poland} \rangle \cup \\ & \langle \mathit{Lidl} \rangle \times \langle \mathit{UK} \rangle \cup \\ & \langle \mathit{Aldi} \rangle \times \langle \mathit{Hungary} \rangle \cup \\ & \langle \mathit{Aldi} \rangle \times \langle \mathit{Poland} \rangle \cup \\ & \langle \mathit{Aldi} \rangle \times \langle \mathit{UK} \rangle \cup \\ & \langle \mathit{Aldi} \rangle \times \langle \mathit{Spain} \rangle \end{aligned}$$

$\langle \mathit{value} \rangle$ denotes a singleton relation $\{(\mathit{value})\}$

Factorisation in Relational Databases

A union of products can be factorised using distributivity:

$$\begin{aligned} \mathit{HasShops} = {} & (\langle \mathit{Tesco} \rangle \cup \langle \mathit{Lidl} \rangle) \times (\langle \mathit{CzechRep} \rangle \cup \langle \mathit{Slovakia} \rangle) \cup \\ & (\langle \mathit{Tesco} \rangle \cup \langle \mathit{Lidl} \rangle \cup \langle \mathit{Aldi} \rangle) \times (\langle \mathit{Hungary} \rangle \cup \langle \mathit{Poland} \rangle \cup \langle \mathit{UK} \rangle) \cup \\ & (\langle \mathit{Tesco} \rangle \cup \langle \mathit{Aldi} \rangle) \times \langle \mathit{Spain} \rangle \end{aligned}$$

Factorisation = expression using singletons, $\cup$ and $\times$.

Factorisation in Relational Databases

A union of products can be factorised using distributivity:

$$\begin{aligned} \mathit{HasShops} = {} & (\langle \mathit{Tesco} \rangle \cup \langle \mathit{Lidl} \rangle) \times (\langle \mathit{CzechRep} \rangle \cup \langle \mathit{Slovakia} \rangle) \cup \\ & (\langle \mathit{Tesco} \rangle \cup \langle \mathit{Lidl} \rangle \cup \langle \mathit{Aldi} \rangle) \times (\langle \mathit{Hungary} \rangle \cup \langle \mathit{Poland} \rangle \cup \langle \mathit{UK} \rangle) \cup \\ & (\langle \mathit{Tesco} \rangle \cup \langle \mathit{Aldi} \rangle) \times \langle \mathit{Spain} \rangle \end{aligned}$$

$$\begin{aligned} = {} & (\langle \mathit{Tesco} \rangle \cup \langle \mathit{Lidl} \rangle) \times (\langle \mathit{CzechRep} \rangle \cup \langle \mathit{Slovakia} \rangle \cup \langle \mathit{Hungary} \rangle \cup \langle \mathit{Poland} \rangle \cup \langle \mathit{UK} \rangle) \cup \\ & \langle \mathit{Tesco} \rangle \times \langle \mathit{Spain} \rangle \cup \\ & \langle \mathit{Aldi} \rangle \times (\langle \mathit{Hungary} \rangle \cup \langle \mathit{Poland} \rangle \cup \langle \mathit{UK} \rangle \cup \langle \mathit{Spain} \rangle) \end{aligned}$$

A given relation can have many factorisations, algebraically equivalent by commutativity of $\cup$ and $\times$ and by distributivity of $\times$ over $\cup$.

Connection: Formal Concepts

chain	country
Tesco	Czech Republic
Tesco	Slovakia
Tesco	Hungary
Tesco	Poland
Tesco	UK
Tesco	Spain
Lidl	Czech Republic
Lidl	Slovakia
Lidl	Hungary
Lidl	Poland
Lidl	UK
Aldi	Hungary
Aldi	Poland
Aldi	UK
Aldi	Spain

	CzechRep	Slovakia	Hungary	Poland	Spain	UK
Tesco	✓	✓	✓	✓	✓	✓
Lidl	✓	✓	✓	✓		✓
Aldi			✓	✓	✓	✓

Connection: Formal Concepts

$$(\langle Tesco \rangle \cup \langle Lidl \rangle) \times (\langle CzechRep \rangle \cup \langle Slovakia \rangle) \cup \quad (1)$$

$$(\langle Tesco \rangle \cup \langle Lidl \rangle \cup \langle Aldi \rangle) \times (\langle Hungary \rangle \cup \langle Poland \rangle \cup \langle UK \rangle) \cup \quad (2)$$

$$(\langle Tesco \rangle \cup \langle Aldi \rangle) \times \langle Spain \rangle \quad (3)$$

	CzechRep	Slovakia	Hungary	Poland	Spain	UK
Tesco	(1)	(1)	(2)	(2)	(3)	(2)
Lidl	(1)	(1)	(2)	(2)		(2)
Aldi			(2)	(2)	(3)	(2)

- Formal concept, closed n-set = maximal product of unions

  But:

- Factorisations allow non-maximal products of unions.
- Factorisations for n-ary relations allow deeper nesting.

Factorisation in Relational Databases

More columns, more complicated factorisations.

$$R_2 \subseteq \mathit{Chains} \times \mathit{Products} \times \mathit{Countries}$$

$$\Big(\langle \mathit{Tesco}\rangle \times (\langle \mathit{food}\rangle \cup \langle \mathit{clothes}\rangle) \cup \langle \mathit{Lidl}\rangle \times \langle \mathit{food}\rangle\Big) \times (\langle \mathit{CzechRep}\rangle \cup \langle \mathit{Slovakia}\rangle) \cup$$
$$(\langle \mathit{Tesco}\rangle \cup \langle \mathit{Lidl}\rangle \cup \langle \mathit{Aldi}\rangle) \times \langle \mathit{food}\rangle \times (\langle \mathit{Poland}\rangle \cup \langle \mathit{UK}\rangle) \cup$$
$$(\langle \mathit{Tesco}\rangle \cup \langle \mathit{Aldi}\rangle) \times \langle \mathit{books}\rangle \times (\langle \mathit{UK}\rangle \cup \langle \mathit{Spain}\rangle)$$

chain	product	country
Tesco	food	CzechRep
Tesco	clothes	CzechRep
Lidl	food	CzechRep
Tesco	food	Slovakia
Tesco	clothes	Slovakia
Lidl	food	Slovakia
Tesco	food	Poland
...		

Why Factorisation?

$$\Big(\langle \mathit{Tesco}\rangle \times (\langle \mathit{food}\rangle \cup \langle \mathit{clothes}\rangle) \cup \langle \mathit{Lidl}\rangle \times \langle \mathit{food}\rangle\Big) \times (\langle \mathit{CzechRep}\rangle \cup \langle \mathit{Slovakia}\rangle) \cup$$
$$(\langle \mathit{Tesco}\rangle \cup \langle \mathit{Lidl}\rangle \cup \langle \mathit{Aldi}\rangle) \times \langle \mathit{food}\rangle \times (\langle \mathit{Poland}\rangle \cup \langle \mathit{UK}\rangle) \cup$$
$$(\langle \mathit{Tesco}\rangle \cup \langle \mathit{Aldi}\rangle) \times \langle \mathit{books}\rangle \times (\langle \mathit{UK}\rangle \cup \langle \mathit{Spain}\rangle)$$

- Data Analysis people say:
  Factorisations reveal deeper structure of the data!
- Database people say:
  Factorisations are useful for data storage!

Why Factorisation?

$$\Big(\langle \mathit{Tesco}\rangle \times (\langle \mathit{food}\rangle \cup \langle \mathit{clothes}\rangle) \cup \langle \mathit{Lidl}\rangle \times \langle \mathit{food}\rangle\Big) \times (\langle \mathit{CzechRep}\rangle \cup \langle \mathit{Slovakia}\rangle) \cup$$
$$(\langle \mathit{Tesco}\rangle \cup \langle \mathit{Lidl}\rangle \cup \langle \mathit{Aldi}\rangle) \times \langle \mathit{food}\rangle \times (\langle \mathit{Poland}\rangle \cup \langle \mathit{UK}\rangle) \cup$$
$$(\langle \mathit{Tesco}\rangle \cup \langle \mathit{Aldi}\rangle) \times \langle \mathit{books}\rangle \times (\langle \mathit{UK}\rangle \cup \langle \mathit{Spain}\rangle)$$

- Factorisations compactly encode combinations of groups of values.
  - Can be exponentially smaller than flat relations.

$$(\langle a\rangle \cup \langle b\rangle) \times (\langle a\rangle \cup \langle b\rangle) \times \cdots \times (\langle a\rangle \cup \langle b\rangle)$$

  - Results of conjunctive queries have succinct factorisations!

Why Factorisation?

$$\Big(\langle \overline{Tesco} \rangle \times (\langle \overline{food} \rangle \cup \langle clothes \rangle) \cup \langle Lidl \rangle \times \langle food \rangle\Big) \times (\langle \overline{CzechRep} \rangle \cup \langle Slovakia \rangle) \cup$$
$$(\langle Tesco \rangle \cup \langle Lidl \rangle \cup \langle Aldi \rangle) \times \langle food \rangle \times (\langle Poland \rangle \cup \langle UK \rangle) \cup$$
$$(\langle Tesco \rangle \cup \langle Aldi \rangle) \times \langle books \rangle \times (\langle UK \rangle \cup \langle Spain \rangle)$$

- Factorisations compactly encode combinations of groups of values.
- Factorisations allow for enumerating tuples with constant delay.

Tesco food CzechRep

Why Factorisation?

$$\Big(\langle \overline{Tesco} \rangle \times (\langle \overline{food} \rangle \cup \langle clothes \rangle) \cup \langle Lidl \rangle \times \langle food \rangle\Big) \times (\langle CzechRep \rangle \cup \langle \overline{Slovakia} \rangle) \cup$$
$$(\langle Tesco \rangle \cup \langle Lidl \rangle \cup \langle Aldi \rangle) \times \langle food \rangle \times (\langle Poland \rangle \cup \langle UK \rangle) \cup$$
$$(\langle Tesco \rangle \cup \langle Aldi \rangle) \times \langle books \rangle \times (\langle UK \rangle \cup \langle Spain \rangle)$$

- Factorisations compactly encode combinations of groups of values.
- Factorisations allow for enumerating tuples with constant delay.

Tesco food CzechRep
Tesco food Slovakia

Why Factorisation?

$$\Big(\langle \overline{Tesco} \rangle \times (\langle food \rangle \cup \langle \overline{clothes} \rangle) \cup \langle Lidl \rangle \times \langle food \rangle\Big) \times (\langle \overline{CzechRep} \rangle \cup \langle Slovakia \rangle) \cup$$
$$(\langle Tesco \rangle \cup \langle Lidl \rangle \cup \langle Aldi \rangle) \times \langle food \rangle \times (\langle Poland \rangle \cup \langle UK \rangle) \cup$$
$$(\langle Tesco \rangle \cup \langle Aldi \rangle) \times \langle books \rangle \times (\langle UK \rangle \cup \langle Spain \rangle)$$

- Factorisations compactly encode combinations of groups of values.
- Factorisations allow for enumerating tuples with constant delay.

Tesco	food	CzechRep
Tesco	food	Slovakia
Tesco	clothes	CzechRep

Why Factorisation?

$$\Big(\langle \overline{Tesco} \rangle \times (\langle food \rangle \cup \langle \overline{clothes} \rangle) \cup \langle Lidl \rangle \times \langle food \rangle\Big) \times (\langle CzechRep \rangle \cup \langle \overline{Slovakia} \rangle) \cup$$
$$(\langle Tesco \rangle \cup \langle Lidl \rangle \cup \langle Aldi \rangle) \times \langle food \rangle \times (\langle Poland \rangle \cup \langle UK \rangle) \cup$$
$$(\langle Tesco \rangle \cup \langle Aldi \rangle) \times \langle books \rangle \times (\langle UK \rangle \cup \langle Spain \rangle)$$

- Factorisations compactly encode combinations of groups of values.
- Factorisations allow for enumerating tuples with constant delay.

Tesco	food	CzechRep
Tesco	food	Slovakia
Tesco	clothes	CzechRep
Tesco	clothes	Slovakia

Why Factorisation?

$$\Big(\langle Tesco\rangle \times (\langle food\rangle \cup \langle clothes\rangle) \cup \langle \overline{Lidl}\rangle \times \langle \overline{food}\rangle\Big) \times (\langle \overline{CzechRep}\rangle \cup \langle Slovakia\rangle) \cup$$
$$(\langle Tesco\rangle \cup \langle Lidl\rangle \cup \langle Aldi\rangle) \times \langle food\rangle \times (\langle Poland\rangle \cup \langle UK\rangle) \cup$$
$$(\langle Tesco\rangle \cup \langle Aldi\rangle) \times \langle books\rangle \times (\langle UK\rangle \cup \langle Spain\rangle)$$

- Factorisations compactly encode combinations of groups of values.
- Factorisations allow for enumerating tuples with constant delay.

Tesco	food	CzechRep
Tesco	food	Slovakia
Tesco	clothes	CzechRep
Tesco	clothes	Slovakia
Lidl	food	CzechRep

Why Factorisation?

$$\Big(\langle \mathit{Tesco}\rangle \times (\langle \mathit{food}\rangle \cup \langle \mathit{clothes}\rangle) \cup \langle \overline{\mathit{Lidl}}\rangle \times \langle \overline{\mathit{food}}\rangle\Big) \times (\langle \mathit{CzechRep}\rangle \cup \langle \overline{\mathit{Slovakia}}\rangle) \cup$$
$$(\langle \mathit{Tesco}\rangle \cup \langle \mathit{Lidl}\rangle \cup \langle \mathit{Aldi}\rangle) \times \langle \mathit{food}\rangle \times (\langle \mathit{Poland}\rangle \cup \langle \mathit{UK}\rangle) \cup$$
$$(\langle \mathit{Tesco}\rangle \cup \langle \mathit{Aldi}\rangle) \times \langle \mathit{books}\rangle \times (\langle \mathit{UK}\rangle \cup \langle \mathit{Spain}\rangle)$$

- Factorisations compactly encode combinations of groups of values.
- Factorisations allow for enumerating tuples with constant delay.

Tesco	food	CzechRep
Tesco	food	Slovakia
Tesco	clothes	CzechRep
Tesco	clothes	Slovakia
Lidl	food	CzechRep
Lidl	food	Slovakia

Why Factorisation?

$$\Big(\langle \mathit{Tesco}\rangle \times (\langle \mathit{food}\rangle \cup \langle \mathit{clothes}\rangle) \cup \langle \mathit{Lidl}\rangle \times \langle \mathit{food}\rangle\Big) \times (\langle \mathit{CzechRep}\rangle \cup \langle \mathit{Slovakia}\rangle) \cup$$
$$(\langle \overline{\mathit{Tesco}}\rangle \cup \langle \mathit{Lidl}\rangle \cup \langle \mathit{Aldi}\rangle) \times \langle \overline{\mathit{food}}\rangle \times (\langle \overline{\mathit{Poland}}\rangle \cup \langle \mathit{UK}\rangle) \cup$$
$$(\langle \mathit{Tesco}\rangle \cup \langle \mathit{Aldi}\rangle) \times \langle \mathit{books}\rangle \times (\langle \mathit{UK}\rangle \cup \langle \mathit{Spain}\rangle)$$

- Factorisations compactly encode combinations of groups of values.
- Factorisations allow for enumerating tuples with constant delay.

Tesco	food	CzechRep
Tesco	food	Slovakia
Tesco	clothes	CzechRep
Tesco	clothes	Slovakia
Lidl	food	CzechRep
Lidl	food	Slovakia
Tesco	food	Poland
...		

Why Factorisation?

$$\Big(\langle \mathit{Tesco}\rangle \times (\langle \mathit{food}\rangle \cup \langle \mathit{clothes}\rangle) \cup \langle \mathit{Lidl}\rangle \times \langle \mathit{food}\rangle\Big) \times (\langle \mathit{CzechRep}\rangle \cup \langle \mathit{Slovakia}\rangle) \cup$$
$$(\langle \mathit{Tesco}\rangle \cup \langle \mathit{Lidl}\rangle \cup \langle \mathit{Aldi}\rangle) \times \langle \mathit{food}\rangle \times (\langle \mathit{Poland}\rangle \cup \langle \mathit{UK}\rangle) \cup$$
$$(\langle \mathit{Tesco}\rangle \cup \langle \mathit{Aldi}\rangle) \times \langle \mathit{books}\rangle \times (\langle \mathit{UK}\rangle \cup \langle \mathit{Spain}\rangle)$$

- Factorisations compactly encode combinations of groups of values.
- Factorisations allow for enumerating tuples with constant delay.
  - Unlike general join decompositions or trivial representation (Q, D) of query results.

Why Factorisation?

$$\Big(\langle \mathit{Tesco}\rangle \times (\langle \mathit{food}\rangle \cup \langle \mathit{clothes}\rangle) \cup \langle \mathit{Lidl}\rangle \times \langle \mathit{food}\rangle\Big) \times (\langle \mathit{CzechRep}\rangle \cup \langle \mathit{Slovakia}\rangle) \cup$$
$$(\langle \mathit{Tesco}\rangle \cup \langle \mathit{Lidl}\rangle \cup \langle \mathit{Aldi}\rangle) \times \langle \mathit{food}\rangle \times (\langle \mathit{Poland}\rangle \cup \langle \mathit{UK}\rangle) \cup$$
$$(\langle \mathit{Tesco}\rangle \cup \langle \mathit{Aldi}\rangle) \times \langle \mathit{books}\rangle \times (\langle \mathit{UK}\rangle \cup \langle \mathit{Spain}\rangle)$$

- Factorisations compactly encode combinations of groups of values.
- Factorisations allow for enumerating tuples with constant delay.
  - Unlike general join decompositions or trivial representation (Q, D) of query results.
- Factorisations allow query evaluation *without expanding* to flat relations
  - Using factorisations can boost query evaluation (processing a smaller representation).
  - Unlike general compression methods.

Why Factorisation?

$$\Big(\langle Tesco\rangle \times (\langle food\rangle \cup \langle clothes\rangle) \cup \langle Lidl\rangle \times \langle food\rangle\Big) \times (\langle CzechRep\rangle \cup \langle Slovakia\rangle) \cup$$
$$(\langle Tesco\rangle \cup \langle Lidl\rangle \cup \langle Aldi\rangle) \times \langle food\rangle \times (\langle Poland\rangle \cup \langle UK\rangle) \cup$$
$$(\langle Tesco\rangle \cup \langle Aldi\rangle) \times \langle books\rangle \times (\langle UK\rangle \cup \langle Spain\rangle)$$

- Factorisations compactly encode combinations of groups of values.
- Factorisations allow for enumerating tuples with constant delay.
  - Unlike general join decompositions or trivial representation (Q, D) of query results.
- Factorisations allow query evaluation *without expanding* to flat relations
  - Using factorisations can boost query evaluation (processing a smaller representation).
  - Unlike general compression methods.

Applications of Factorised Representations

Relational data storage and management:

- Storing & querying relationship (many-to-many) data
- Succinct representation of large query results
  - Storing intermediate results in query evaluation
  - Knowledge compilation (compile once, speed up all subsequent queries)
  - Parallel computation (compress intermediate results sent b/w machines)
- Managing a large set of possibilities or choices
  - Configuration problems (space of valid solutions)
  - Incomplete information (space of possible worlds)
- Provenance and probabilistic data
  - Compact encoding for large provenance
  - Factorisation of provenance is used for query evaluation in probabilistic databases.

Key Challenges

1. How compact can factorised query results be?

2. Can such factorisations speed up query evaluation?

Key Results

1. How compact can factorised query results be?

- Asymptotic size bounds for factorisations of query results.
- Characterisation of conjunctive queries based on factorisability of their results.

2. Can such factorisations speed up query evaluation?

- Compute factorisations directly from query and input data.
- Optimisation and evaluation techniques for queries with selections, projections, joins, aggregates and ordering on factorisations.
- FDB: a main-memory engine using factorised relations.

Key Challenges

1. How compact can factorised query results be?

Factorisation of Joins

Products	
chain	product
Tesco	food
Tesco	clothes
Tesco	books
Aldi	food

Locations	
chain	country
Tesco	Slovakia
Tesco	UK
Aldi	UK
Aldi	Spain

Join two tables on *chain*:

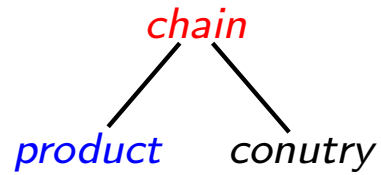
Products $\bowtie_{chain}$ Locations		
shop	product	country
Tesco	food	Slovakia
Tesco	clothes	Slovakia
Tesco	books	Slovakia
Tesco	food	UK
Tesco	clothes	UK
Tesco	books	UK
Aldi	food	UK
Aldi	food	Spain

$$= \langle Tesco \rangle \times (\langle food \rangle \cup \langle clothes \rangle \cup \langle books \rangle) \times (\langle Slovakia \rangle \cup \langle UK \rangle) \cup$$
$$\langle Aldi \rangle \times \langle food \rangle \times (\langle UK \rangle \cup \langle Spain \rangle)$$

Factorisation Trees

Uniform nesting structures $\longleftrightarrow$ defined by *factorisation trees* (f-trees).

$$\bigcup_{Chains} \Big(\langle chain \rangle \times \Big(\bigcup_{Products} \langle product \rangle \Big) \times \Big(\bigcup_{Countries} \langle country \rangle \Big) \Big) \longleftrightarrow$$

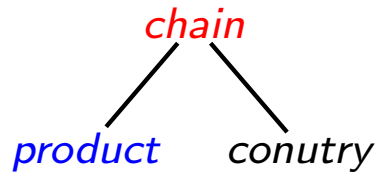


$$\langle Tesco \rangle \times (\langle food \rangle \cup \langle clothes \rangle \cup \langle books \rangle) \times (\langle Slovakia \rangle \cup \langle UK \rangle)$$
$$\cup \langle Aldi \rangle \times \langle food \rangle \times (\langle UK \rangle \cup \langle Spain \rangle)$$

Factorisation Trees

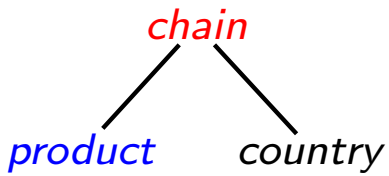
Uniform nesting structures $\longleftrightarrow$ defined by *factorisation trees* (f-trees).

$$\bigcup_{Chains} \Big(\langle chain \rangle \times \Big(\bigcup_{Products} \langle product \rangle \Big) \times \Big(\bigcup_{Countries} \langle country \rangle \Big) \Big) \longleftrightarrow$$



$$\bigcup_{Chains} \Big(\langle chain \rangle \times \Big(\bigcup_{Products} \langle product \rangle \times \Big(\bigcup_{Countries} \langle country \rangle \Big) \Big) \Big) \longleftrightarrow$$

Factorisation Trees for Relations



$$\longleftrightarrow \quad \bigcup\Big(\langle chain\rangle \times \Big(\bigcup\langle product\rangle\Big) \times \Big(\bigcup\langle country\rangle\Big)\Big).$$

Some relations are not factorisable using this f-tree

ch	p	c
Tesco	food	UK
Tesco	clothes	CzechRep

$$\neq \quad \bigcup\Big(\langle ch\rangle \times (\bigcup\langle p\rangle) \times (\bigcup\langle c\rangle)\Big)$$

but the join is *always* factorisable using this f-tree

$$Products(ch, p) \bowtie_s Locations(ch, c) \quad = \quad \bigcup\Big(\langle ch\rangle \times (\bigcup\langle p\rangle) \times (\bigcup\langle c\rangle)\Big)$$

since p and c are *conditionally independent*.

Factorisation Trees for Query Results

For any join query Q,

we characterise **f-trees that always factorise** the result of Q.

If Q is an equi-join query and $\mathcal{T}$ any f-tree, then

the result $Q(\mathbf{D})$ can be factorised according to $\mathcal{T}$ for *any* database $\mathbf{D}$

iff

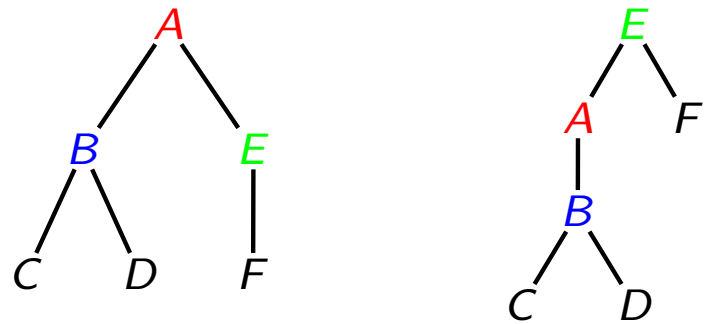
for each relation of Q, all its attributes are on a root-to-leaf path.

If Q has projections, a similar but more complicated result holds.

Factorisation Trees for Query Results

Consider the query:

$$Q(A, B, C, D, E, F) \leftarrow R(A, B, C), S(A, B, D), T(A, E), U(E, F).$$



Left f-tree induces the factorisation structure:

$$\bigcup_{a \in A} \Big(\langle a \rangle \times \bigcup_{b \in B} \Big(\langle b \rangle \times \big(\bigcup_{c \in C} \langle c \rangle \Big) \times \big(\bigcup_{d \in D} \langle d \rangle \big) \big) \times \bigcup_{e \in E} \Big(\langle e \rangle \times \big(\bigcup_{f \in F} \langle f \rangle \big) \Big) \Big)$$

Key Challenges

1. How compact can factorised query results be?

Size of Factorisations

The *size* of a factorisation is the number of its singleton data elements.

$$\left|\left(\langle a\rangle\langle 1\rangle \cup \langle a\rangle\langle 2\rangle \cup \langle b\rangle\langle 1\rangle \cup \langle b\rangle\langle 2\rangle \cup \langle c\rangle\langle 1\rangle \cup \langle c\rangle\langle 2\rangle\right)\right| = 12,$$

$$\left|\left(\langle a\rangle \cup \langle b\rangle \cup \langle c\rangle\right) \times \left(\langle 1\rangle \cup \langle 2\rangle\right)\right| = 5.$$

How much space do we save by factorisation?

Size of Factorisations: Main Result

For any conjunctive query Q there is a number $s(Q)$ such that

For any database $\mathbf{D}$, $Q(\mathbf{D})$ admits a factorisation of size $O(|\mathbf{D}|^{s(Q)})$.

- **Best possible bound** for a general result of Q using f-trees.

  There exists $\mathbf{D}$ such that all factorisations over f-trees are $\Omega(|\mathbf{D}|^{s(Q)})$.

- Instance-optimal factorisation of relations (i.e., dependent on $\mathbf{D}$) is hard.

Flat vs. Factorised Size

- For any database $\mathbf{D}$, $|Q(\mathbf{D})|$ is $O(|\mathbf{D}|^{\rho^*(Q)})$. [AGM'08]
- For any database $\mathbf{D}$, $Q(\mathbf{D})$ admits factorisation of size $O(|\mathbf{D}|^{s(Q)})$. [OZ'11]

$$1 \leq s(Q) \leq \rho^*(Q) \leq |Q|$$

There are classes of queries with $s(Q) \ll \rho^*(Q)$.

Appendix: Intuition for Asymptotic Bounds

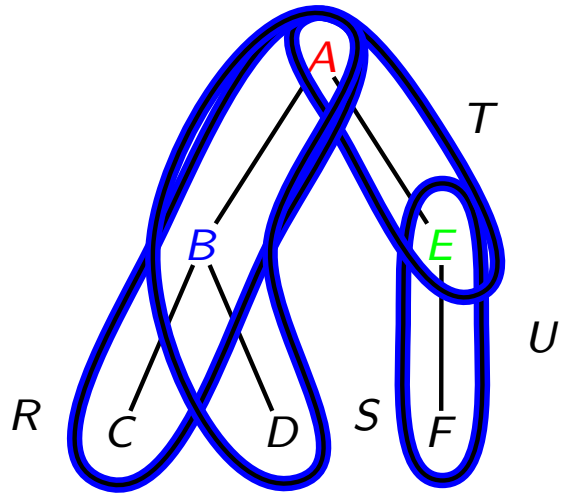
- For any database $\mathbf{D}$, $|Q(\mathbf{D})|$ is $O(|\mathbf{D}|^{\rho^*(Q)})$. [AGM'08]
- For any database $\mathbf{D}$, $Q(\mathbf{D})$ admits factorisation of size $O(|\mathbf{D}|^{s(Q)})$. [OZ'11]

What is $\rho^*(Q)$ and $s(Q)$?

Intuition – Edge Covers

$$Q(A, B, C, D, E, F) \leftarrow R(A, B, C), S(A, B, D), T(A, E), U(E, F).$$

The hypergraph of Q:



First observation:

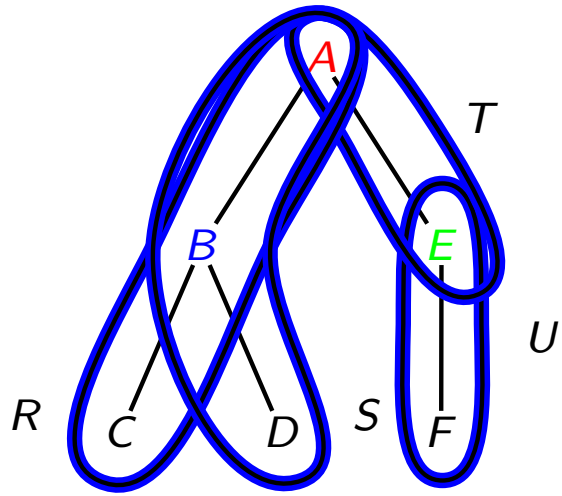
- Cover all attributes by k relations $\Rightarrow |Q(\mathbf{D})| \leq |\mathbf{D}|^k$.
- Set of m independent attributes $\Rightarrow$ construct $\mathbf{D}$ with $|Q(\mathbf{D})| \sim |\mathbf{D}|^m$.

$$\max_m = \text{IndependentSet}(Q) \leq \text{EdgeCover}(Q) = \min_k$$

Intuition – Fractional Edge Covers

$$Q(A, B, C, D, E, F) \leftarrow R(A, B, C), S(A, B, D), T(A, E), U(E, F).$$

The hypergraph of Q:



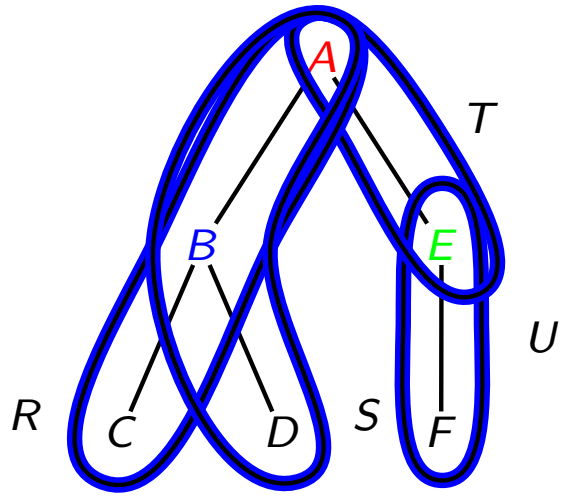
[AGM'08]:

- *Fractional* edge cover of Q with weight k $\Rightarrow |Q(\mathbf{D})| \leq |\mathbf{D}|^k$.
- *Fractional* independent set of weight $m \Rightarrow$ construct $\mathbf{D}$ with $|Q(\mathbf{D})| \sim |\mathbf{D}|^m$.

By linear programming duality:

$$\max_m = \text{FractionalIndependentSet}(Q) = \text{FractionalEdgeCover}(Q) = \min_k$$

Intuition – Fractional Edge Covers



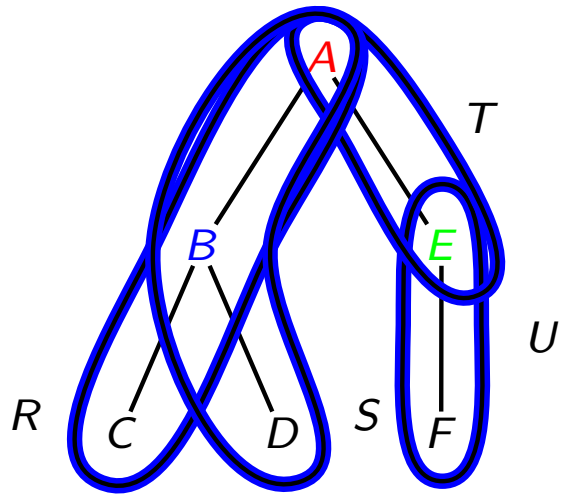
For a query $Q = R_1 \bowtie \cdots \bowtie R_n$, the *fractional edge cover number* $\rho^*(Q)$ is the cost of an optimal solution to the linear program

$$\begin{aligned} \text{minimising} \quad & \textstyle\sum_i x_{R_i} \\ \text{subject to} \quad & \textstyle\sum_{i:R_i \text{ has attribute } A} x_{R_i} \geq 1 \quad \text{for all attributes } A, \\ & x_{R_i} \geq 0 \qquad\qquad\qquad\quad \text{for all } R_i. \end{aligned}$$

- x_{R_i} is the weight of relation R_i.
- Each attribute has to be covered by relations with sum of weights ≥ 1.
- In the non-weighted edge cover, the variables $x_{R_i} \in \{0, 1\}$

Example

$$Q(A, B, C, D, E, F) \leftarrow R(A, B, C), S(A, B, D), T(A, E), U(E, F).$$

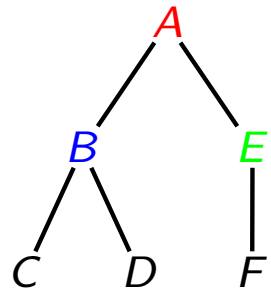


- Relations R, S, U cover the whole query.
  $\mathrm{FractionalEdgeCover}(Q) \leq 3$
- Each of the nodes C, D, and F must be covered by separate relations.
  $\mathrm{FractionalIndependentSet}(Q) \geq 3$

$\Rightarrow \rho^*(Q) = 3$

$\Rightarrow |Q(\mathbf{D})| = O(|\mathbf{D}|^3)$ and for some inputs $|Q(\mathbf{D})| = \Theta(|\mathbf{D}|^3)$.

Intuition – Size of Factorisations



$$\bigcup_{a\in A}\Big(\langle a\rangle \times \bigcup_{b\in B}\Big(\langle b\rangle \times \big(\bigcup_{c\in C}\langle c\rangle\big) \times \big(\bigcup_{d\in D}\langle d\rangle\big)\Big) \times \bigcup_{e\in E}\Big(\langle e\rangle \times \big(\bigcup_{f\in F}\langle f\rangle\big)\Big)\Big)$$

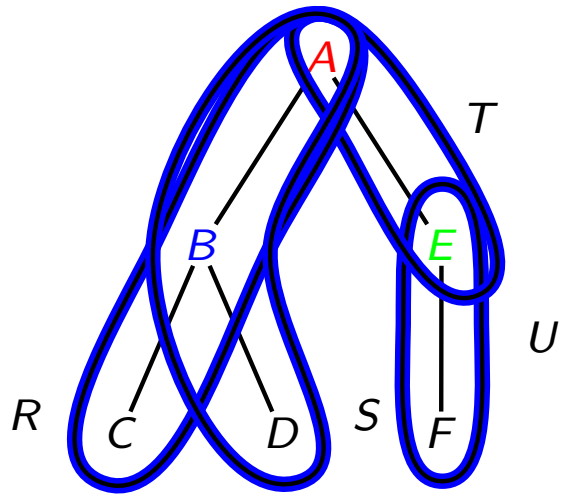
Attributes only depend on ancestor attributes

- F only depends on E and A.
- One $\langle f\rangle$ for each $(a, e, f) \in Q(\mathbf{D})$.
- The number of F-singletons is $|\pi_{A,E,F}(Q(\mathbf{D}))|$.

Size of f-representation = sum of sizes of **subqueries along paths**.

Example

$$Q(A, B, C, D, E, F) \leftarrow R(A, B, C), S(A, B, D), T(A, E), U(E, F).$$



- Path A, E, F has fractional edge cover 2.
  $\Rightarrow$ The number of F-singletons is $\leq |\mathbf{D}|^2$, but can be $\sim |\mathbf{D}|^2$.
- All other root-to-leaf paths have fractional edge cover 1.
  $\Rightarrow$ The number of other singletons is $\leq |\mathbf{D}|$.

$s(Q) = 2$ $\Rightarrow$ Factorisation size $\sim |\mathbf{D}|^2$

Recall that $\rho^*(Q) = 3$ $\Rightarrow$ Flat size $\sim |\mathbf{D}|^3$

Flat vs. Factorised Size

- For any database $\mathbf{D}$, $|Q(\mathbf{D})|$ is $O(|\mathbf{D}|^{\rho^*(Q)})$. [AGM'08]
- For any database $\mathbf{D}$, $Q(\mathbf{D})$ admits factorisation of size $O(|\mathbf{D}|^{s(Q)})$. [OZ'11]

- $\rho^*(Q)$ = fractional edge cover number of the **entire query**.
- $s(Q)$ = fractional edge cover number of **root-to-leaf paths in best f-tree**.

$$1 \leq s(Q) \leq \rho^*(Q) \leq |Q|$$

There are classes of queries with $s(Q) \ll \rho^*(Q)$.
(There are classes of queries with $s(Q) = 1$ and $\rho^*(Q) = |Q|$.)

Thank you!