

α -Acyclic Joins

Jef Wijsen

August 16, 2024

Outline

Motivation

Full Reducer: Definition

Full Reducer: Existence and Construction

Computing an α -Acyclic Join

Join Trees

Computing a Projection of an α -Acyclic Join

New Join Algorithms

Distributed Join

- ▶ $M[NN, \textit{Field_of_Study}, \textit{Year}]$ stores data about 5000 students of UMONS. The relation M is stored in Mons.
- ▶ $B[NN, \textit{Street}, \textit{Number}, \textit{City}]$ stores the addresses of 10.000.000 Belgian citizens. The relation B is stored in Brussels.
- ▶ **Question:** Get $M \bowtie B$ in Mons.

Distributed Join

- ▶ $M[NN, \textit{Field_of_Study}, \textit{Year}]$ stores data about 5000 students of UMONS. The relation M is stored in Mons.
- ▶ $B[NN, \textit{Street}, \textit{Number}, \textit{City}]$ stores the addresses of 10.000.000 Belgian citizens. The relation B is stored in Brussels.
- ▶ **Question:** Get $M \bowtie B$ in Mons.
- ▶ **Solution:**
 - ▶ Compute $\pi_{NN}(M)$ in Mons and ship the result (5000 national numbers) to Brussels.
 - ▶ Compute $B \bowtie (\pi_{NN}(M))$ in Brussels and ship the result (5000 tuples of B) to Mons.
 - ▶ Compute $M \bowtie (B \bowtie (\pi_{NN}(M)))$ in Mons.

Terminology

- ▶ New operator called **semijoin**: $B \bowtie M$ gets the set of tuples in B that join with some tuple of M (on the common attribute NN).
- ▶ The tuples of B not in $B \bowtie M$ are said to be **dangling** with respect to the join of the relations in $\{B, M\}$.

Note that \bowtie is not a primitive operator in SPJRUD:

- ▶ $R \bowtie S \equiv \pi_{\text{sort}(R)}(R \bowtie S)$
- ▶ $R \bowtie S \equiv R \bowtie \pi_{\text{sort}(R) \cap \text{sort}(S)}(S)$

Remarkable Result (FYI only)

Let $SP \times RUD$ denote relational algebra with semijoin but without join.

Theorem

The following problem is decidable: Given two expressions E and F in $SP \times RUD$, decide whether $E \equiv F$.

Joining Three or More Relations

R	A	B		S	B	C		T	C	D	
	1	2			1	2	*		1	2	*
	2	4	*		2	4			2	4	*
	3	6	*		3	6	*		3	6	*
	4	8	*		4	8	*		4	8	

$R \bowtie S \bowtie T$	A	B	C	D
	1	2	4	8

- ▶ As it turns out, all *-tuples are dangling with respect to $\{R, S, T\}$.
- ▶ It would be interesting to remove dangling tuples **prior** to the join.
- ▶ Can we recognize (and remove) dangling tuples **by using only semijoins**?

Outline

Motivation

Full Reducer: Definition

Full Reducer: Existence and Construction

Computing an α -Acyclic Join

Join Trees

Computing a Projection of an α -Acyclic Join

New Join Algorithms

Simplifying Notations

- ▶ $R[A, B]$ means “relation name R with $\text{sort}(R) = \{A, B\}$.”
- ▶ We use R where $\text{sort}(R)$ is expected. For example,
$$R \bowtie S = \pi_R (R \bowtie S).$$
- ▶ We use R where $R^{\mathcal{I}}$ is expected (where \mathcal{I} is a database).
- ▶ A **schema** \mathbf{S} is a finite set of relation names. We will assume that $\text{sort}(\cdot)$ does not map distinct relation names to the same set of attributes.
 \implies We can use $\text{sort}(R)$ where R is expected.
- ▶ If $\mathbf{S} = \{R_1, R_2, \dots, R_n\}$, then $\bowtie \mathbf{S}$ is a shorthand for
$$R_1 \bowtie R_2 \bowtie \dots \bowtie R_n.$$

For example, we can write $AB \bowtie BC \bowtie CD$.

Removing Dangling Tuples by Means of Semijoins

- ▶ A tuple $t \in R_i$ is **dangling** with respect to $\mathbf{S} := \{R_1, \dots, R_n\}$ if $t \notin \pi_{R_i}(R_1 \bowtie R_2 \bowtie \dots \bowtie R_n) \equiv R_i \bowtie (\bowtie (\mathbf{S} \setminus \{R_i\}))$.
- ▶ A **semijoin program** is a finite sequence of instructions of the form

$$R := R \bowtie S$$

The semantics of this instruction is: remove from R all tuples that are dangling with respect to $\{R, S\}$.

- ▶ A semijoin program is a **full reducer** for \mathbf{S} if it removes all tuples that are dangling with respect to \mathbf{S} (for every database).

Counterexample

The semijoin program

$$AB := AB \bowtie BC$$

$$BC := BC \bowtie CD$$

$$CD := CD \bowtie BC$$

is not a full reducer for $\{AB, BC, CD\}$, as shown by:

$$R \left| \begin{array}{cc} A & B \\ \hline 1 & 2 \end{array} \right. \quad S \left| \begin{array}{cc} B & C \\ \hline 2 & 4 \end{array} \right. \quad T \left| \begin{array}{cc} C & D \\ \hline & \end{array} \right.$$

Example of a Full Reducer

The semijoin program

$$BC := BC \times AB \quad (1)$$

$$CD := CD \times BC \quad (2)$$

$$BC := BC \times CD \quad (3)$$

$$AB := AB \times BC \quad (4)$$

is a full reducer for $\{AB, BC, CD\}$. Why?

Lines (2) and (3) remove from $BC \cup CD$ all tuples that are dangling with respect to $\{BC, CD\}$. But before that, (1) has removed from BC all tuples that do not join with AB . Thus, after (3), no tuples of $BC \cup CD$ are dangling with respect to $\{AB, BC, CD\}$. Finally, (4) removes from AB all tuples that are dangling with respect to $\{AB, BC, CD\}$.

Schema Without Full Reducer

There is no full reducer for $\{AB, BC, AC\}$, as shown by:

R	A	B	S	B	C	U	A	C
	a	b		b	c		d	c
	d	e		e	f		a	f

Indeed, no tuples are removed by:

$$AB := AB \times BC$$

$$AB := AB \times AC$$

$$BC := BC \times AB$$

$$BC := BC \times AC$$

$$AC := AC \times AB$$

$$AC := AC \times BC$$

Outline

Motivation

Full Reducer: Definition

Full Reducer: Existence and Construction

Computing an α -Acyclic Join

Join Trees

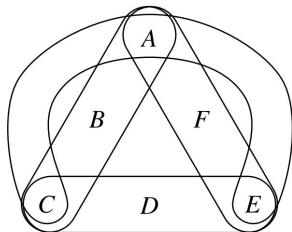
Computing a Projection of an α -Acyclic Join

New Join Algorithms

Which Schemas Have a Full Reducer?

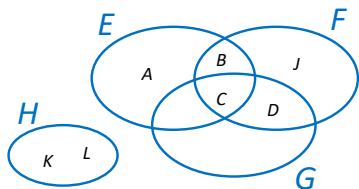
The **hypergraph** of \mathbf{S} has vertex set $\bigcup_{R \in \mathbf{S}} \text{sort}(R)$ and hyperedge set $\{\text{sort}(R) \mid R \in \mathbf{S}\}$. Hypergraphs generalize undirected graphs by allowing edges with three or more vertices.

For example, the hypergraph of $\mathbf{S} := \{ABC, ACE, AEF, CDE\}$:



Ears

- ▶ A hyperedge E is an **ear** of another hyperedge F if the attributes in $E \setminus F$ belong to no other hyperedge than E .
- ▶ A hyperedge E is also an **ear** if its attributes belong to no other hyperedge than E .



H is an ear.

E is an ear of F .

E is not an ear of G , nor of H .

G is an ear of F .

G is not an ear of E , nor of H .

F is not an ear of some other hyperedge.

Ear Removal or GYO Reduction

GYO=Graham-Yu-Özsoyoglu

The **GYO-reduction** of a hypergraph is obtained by applying ear removal until no more removals are possible. A hypergraph is **α -acyclic** if its GYO-reduction is the empty hypergraph; otherwise it is **α -cyclic**.

A schema is α -acyclic if its hypergraph is α -acyclic.

Exercise. Show the following:

- ▶ E ear of F & F ear of $G \implies E$ ear of G ;
- ▶ the GYO-reduction of a hypergraph is unique, independent of the sequence of ear removals chosen;
- ▶ an undirected graph is acyclic iff it is α -acyclic.

Theorem

A schema has a full reducer iff it is α -acyclic.

Observation

The extension of an α -cyclic schema can be α -acyclic:

- ▶ $\{AB, BC, AC\}$ is α -cyclic;
- ▶ $\{AB, BC, AC, ABC\}$ is α -acyclic.

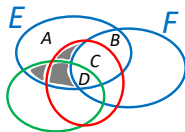
Proof of \Leftarrow : An α -Acyclic Schema Has a Full Reducer

Assume the GYO reduction of α -acyclic schema \mathbf{S} first removes E , ear of F . Construct the full reducer:

$$F := F \times E \quad (5)$$

$$\boxed{\text{full reducer of } \mathbf{S} \setminus \{E\}} \quad (6)$$

$$E := E \times F \quad (7)$$



After line (5), all tuples in F join with [some tuple of] E . After (6), if g is a tuple in some relation $G \in \mathbf{S} \setminus \{E\}$, then g is not dangling with respect to $\mathbf{S} \setminus \{E\}$. That is, $\bowtie (\mathbf{S} \setminus \{E\})$ will contain a tuple t such that $t[G] = g$. Since $t[F] \in F$, $t[F]$ will join with E . But then the entire tuple t will join with E (because all attributes common to E and $\mathbf{S} \setminus \{E\}$ belong to F). Consequently, g will not be dangling with respect to \mathbf{S} . Finally, line (7) makes sure that no tuple of E will be dangling with respect to \mathbf{S} .

Outline

Motivation

Full Reducer: Definition

Full Reducer: Existence and Construction

Computing an α -Acyclic Join

Join Trees

Computing a Projection of an α -Acyclic Join

New Join Algorithms

Join Order

Assume we have executed the full reducer:

$$F := F \bowtie E$$

full reducer of $\mathbf{S} \setminus \{E\}$

$$E := E \bowtie F$$

In what order will we join the relations of \mathbf{S} ?

Compute $\bowtie (\mathbf{S} \setminus \{E\})$ in *Result*

$$Result := E \bowtie Result$$

The computation in the box is applied recursively until $\mathbf{S} \setminus \{E\}$ is a single relation.

Exercise. Show that the size of *Result* will not decrease.

Example $AB \bowtie BC \bowtie CD$

Remove AB , ear of BC . Then, in $\{BC, CD\}$, remove BC , ear of CD . Finally, remove CD .

$$BC := BC \bowtie AB$$

$$CD := CD \bowtie BC$$

$$BC := BC \bowtie CD$$

$$AB := AB \bowtie BC$$

$$\text{Result} := CD$$

$$\text{Result} := BC \bowtie \text{Result}$$

$$\text{Result} := AB \bowtie \text{Result}$$

Thus, $AB \bowtie (BC \bowtie CD)$, after removal of dangling tuples.

Exercise. Show that $BC \bowtie (AB \bowtie CD)$ is a bad join order, even if there are no dangling tuples.

Exercises

Exercise. Show that a schema \mathbf{S} is α -acyclic if and only if

1. $\mathbf{S} = \emptyset$, or
2. \mathbf{S} has an ear E and $\mathbf{S} \setminus \{E\}$ is α -acyclic.

Exercise. Show that \mathbf{S} is α -acyclic if and only if all the attributes of \mathbf{S} can be deleted by repeatedly applying the following two operations:

1. delete an attribute that occurs in only one hyperedge;
2. delete a hyperedge that is contained in another hyperedge.

Exercise. Let *Schema* be a binary relation that stores a schema \mathbf{S} as follows: a fact *Schema*(A, R) means that A is an attribute and R a relation name such that $A \in \text{sort}(R)$. Is there a program in Stratified Datalog that checks whether \mathbf{S} is α -acyclic?

Outline

Motivation

Full Reducer: Definition

Full Reducer: Existence and Construction

Computing an α -Acyclic Join

Join Trees

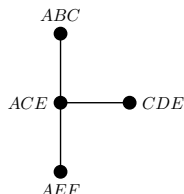
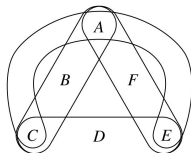
Computing a Projection of an α -Acyclic Join

New Join Algorithms

Join Trees

A **join tree** of a hypergraph¹ \mathbf{S} is a tree (i.e., a connected acyclic undirected graph) whose vertices are the hyperedges of \mathbf{S} such that the following condition holds:

Connectedness Condition: For every attribute A , the subgraph of the tree induced by the vertices that contain A is connected.



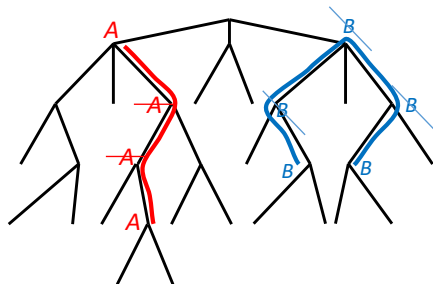
Theorem

A schema has a join tree iff it is α -acyclic.

¹We blur the distinction between schemas and hypergraphs.

Proof that an α -Acyclic Schema Has a Join Tree

Assume an α -acyclic schema. Construct a tree as follows: if E is removed as an ear of F , then E is made a child of F in the tree. Assume, toward a contradiction, that the resulting tree is not a join tree. Then, there must be a path that is like the red path or the blue path in the figure below. Explain why the existence of either path leads to a contradiction.



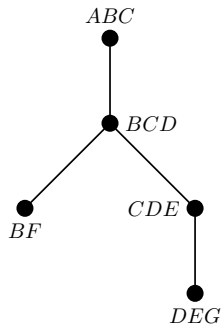
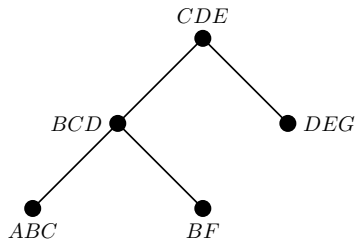
Proof that a Schema with a Join Tree is α -Acyclic

Assume a schema \mathbf{S} with a join tree τ . We will show that \mathbf{S} has a GYO reduction that removes all hyperedges. The proof is by induction on the number n of hyperedges in the schema (which is equal to the number of vertices in τ).

Induction basis $n = 1$. The single hyperedge is an ear and can be removed.

Induction step $n \geq 2$. Construct a **rooted tree** by designating some vertex of τ as the root. In the rooted tree, let E be a leaf node whose parent is F . Then, E is an ear of F , because if $A \in E \setminus F$, then the **Connectedness Condition** implies that A occurs nowhere else in the tree. If we remove E from τ , we obtain a join tree of $\mathbf{S} \setminus \{E\}$ (Why?). By the induction hypothesis, $\mathbf{S} \setminus \{E\}$ has a GYO reduction that removes all hyperedges.

Any Vertex Can be Picked as the Root



Outline

Motivation

Full Reducer: Definition

Full Reducer: Existence and Construction

Computing an α -Acyclic Join

Join Trees

Computing a Projection of an α -Acyclic Join

New Join Algorithms

Computing a Projection on X of an α -Acyclic Join

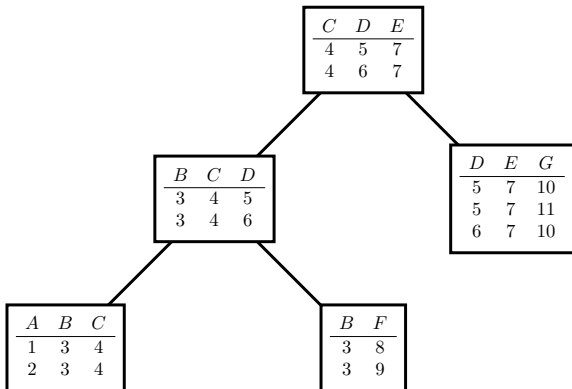
Yannakakis' Algorithm for computing $\pi_X (\bowtie \mathbf{S})$ with α -acyclic \mathbf{S} :

- (i) Apply a full reducer.
- (ii) Construct a rooted join tree for \mathbf{S} .
- (iii) Visit each node of the rooted join tree, other than the root, in some bottom-up order. When we visit E whose parent is F (where $E, F \in \mathbf{S}$), we execute

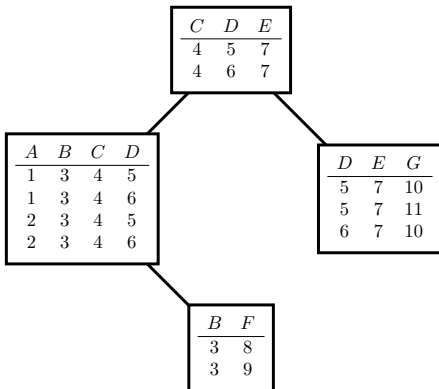
$$F := \pi_{F \cup (X \cap E)} (E \bowtie F).$$

- (iv) Project the relation at the root onto X .

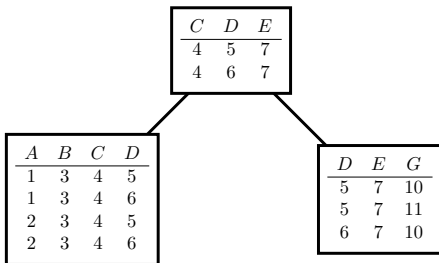
$\pi_{AG} (ABC \bowtie BCD \bowtie BF \bowtie CDE \bowtie DEG)$



$\pi_{AG} (ABC \bowtie BCD \bowtie BF \bowtie CDE \bowtie DEG)$



$\pi_{AG}(ABC \bowtie BCD \bowtie BF \bowtie CDE \bowtie DEG)$



$\pi_{AG}(ABC \bowtie BCD \bowtie BF \bowtie CDE \bowtie DEG)$

<i>A</i>	<i>C</i>	<i>D</i>	<i>E</i>
1	4	5	7
1	4	6	7
2	4	5	7
2	4	6	7

<i>D</i>	<i>E</i>	<i>G</i>
5	7	10
5	7	11
6	7	10

$\pi_{AG}(ABC \bowtie BCD \bowtie BF \bowtie CDE \bowtie DEG)$

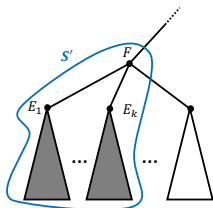
<i>A</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>G</i>
1	4	5	7	10
1	4	5	7	11
1	4	6	7	10
2	4	5	7	10
2	4	5	7	11
2	4	6	7	10

<i>A</i>	<i>G</i>
1	10
1	11
2	10
2	11

Theorem

At every execution of step (iii) in Yannakakis' algorithm, the intermediate result does not contain more than IU tuples, where I and U are the number of tuples in the input and output, respectively.

Proof.



Let $F \in \mathbf{S}$, and let E_1, \dots, E_k be the children of F that have already been visited. Let \mathbf{S}' be the subset of \mathbf{S} indicated on the figure. Then, the relation stored at F is $\pi_{YF}(\bowtie \mathbf{S}')$, where Y contains the attributes of X that appear in some hyperedge of \mathbf{S}' but not in F .

Obviously, $|\pi_{YF}(\bowtie \mathbf{S}')| \leq |\pi_Y(\bowtie \mathbf{S}')| \times |\pi_F(\bowtie \mathbf{S}')|$. Then,

- ▶ since we have applied a full reducer, every intermediate Y -value will appear in the final result. Thus, $|\pi_Y(\bowtie \mathbf{S}')| \leq U$; and
- ▶ since every tuple in $\pi_F(\bowtie \mathbf{S}')$ is in the input, $|\pi_F(\bowtie \mathbf{S}')| \leq I$.

Outline

Motivation

Full Reducer: Definition

Full Reducer: Existence and Construction

Computing an α -Acyclic Join

Join Trees

Computing a Projection of an α -Acyclic Join

New Join Algorithms

Challenge

R	A	B	S	B	C	U	A	C
	?	?		?	?		?	?
	?	?		?	?		?	?

Add 2 tuples to each relation such that $AB \bowtie BC \bowtie AC$ contains at least 3 tuples.

Note About the α -Cyclic “Triangle Query” I

- ▶ An algorithm is said to run in $\Omega(f(n))$ (Big-Omega) time if there exists a constant k such that on inputs of sufficiently large size n , the algorithm uses **at least** $k \times f(n)$ steps.
- ▶ Consider $AB \bowtie AC \bowtie BC$ with $|AB| = |AC| = |BC| = N$. Any of the three join plans below must run in time $\Omega(N^2)$. Why?

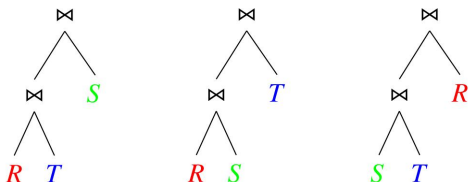


Figure copied from [NRR13].

How large can $|AB \bowtie AC \bowtie BC|$ be?

Note About the α -Cyclic “Triangle Query” II

How large can $|AB \bowtie AC \bowtie BC|$ be?

Let $\pi_A(AB) \cap \pi_A(AC) = \{a_1, \dots, a_n\}$. Then,

$$AB \bowtie AC \bowtie BC = \bigcup_{i=1}^n ((\sigma_{A=a_i}(AB) \bowtie \sigma_{A=a_i}(AC)) \bowtie BC)$$

Consequently,

$$|AB \bowtie AC \bowtie BC| \leq \sum_{i=1}^n \min(|\sigma_{A=a_i}(AB)| \cdot |\sigma_{A=a_i}(AC)|, |BC|)$$

Since $\min(x, y) \leq \sqrt{xy}$,

$$\begin{aligned} |AB \bowtie AC \bowtie BC| &\leq \sum_{i=1}^n \sqrt{|\sigma_{A=a_i}(AB)| \cdot |\sigma_{A=a_i}(AC)| \cdot |BC|} \\ &= \sqrt{|BC|} \cdot \sum_{i=1}^n \sqrt{|\sigma_{A=a_i}(AB)| \cdot |\sigma_{A=a_i}(AC)|} \end{aligned}$$

Note About the α -Cyclic “Triangle Query” III

Cauchy-Schwarz

$$\left(\sum_{i=1}^n x_i y_i \right)^2 \leq \left(\sum_{i=1}^n x_i^2 \right) \cdot \left(\sum_{i=1}^n y_i^2 \right)$$
$$\sum_{i=1}^n x_i y_i \leq \sqrt{\sum_{i=1}^n x_i^2} \cdot \sqrt{\sum_{i=1}^n y_i^2}$$

Thus,

$$\begin{aligned} |AB \bowtie AC \bowtie BC| &\leq \sqrt{|BC|} \cdot \sqrt{\sum_{i=1}^n |\sigma_{A=a_i}(AB)|} \cdot \sqrt{\sum_{i=1}^n |\sigma_{A=a_i}(AC)|} \\ &= \sqrt{|BC|} \cdot \sqrt{|AB|} \cdot \sqrt{|AC|} \end{aligned}$$

If $|AB| = |AC| = |BC| = N$, then the above is $\mathcal{O}(N^{\frac{3}{2}})$.

Note About the α -Cyclic “Triangle Query” IV

Algorithm 1 Computing Q_Δ with power of two choices.

Input: $R(A, B), S(B, C), T(A, C)$ in sorted order

```
1:  $Q_\Delta \leftarrow \emptyset$ 
2:  $L \leftarrow \pi_A(R) \cap \pi_A(T)$ 
3: For each  $a \in L$  do
4:   If  $|\sigma_{A=a}R| \cdot |\sigma_{A=a}T| \geq |S|$  then
5:     For each  $(b, c) \in S$  do
6:       If  $(a, b) \in R$  and  $(a, c) \in T$  then
7:         Add  $(a, b, c)$  to  $Q_\Delta$ 
8:     else
9:       For each  $b \in \pi_B(\sigma_{A=a}R) \wedge c \in \pi_C(\sigma_{A=a}T)$  do
10:        If  $(b, c) \in S$  then
11:          Add  $(a, b, c)$  to  $Q_\Delta$ 
12: Return  $Q$ 
```

Algorithm copied from [NRR13].

Example

R	A	B	T	A	C	S	B	C
	a_1	1		a_1	2		1	2
	a_1	3		a_1	4		3	4
	a_1	5		a_2	4		5	4
	a_2	5		a_2	6			

$$L = \{a_1, a_2\}.$$

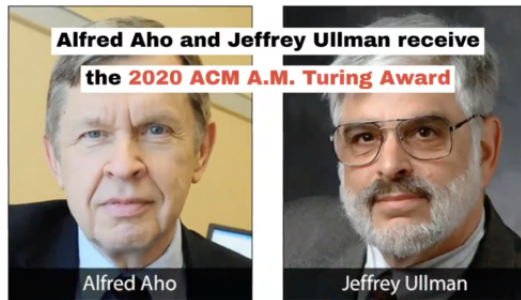
For a_1 , we execute lines 5–7 of Algorithm 1 (because $3 \cdot 2 = 6 \geq |S|$), for a_2 , we execute lines 9–11 (because $1 \cdot 2 = 2 < |S|$).

(Recall: there exists no full reducer for $\{AB, AC, BC\}$.)

Exercises

Exercises taken from [Ull89].

Turing Award 2020



For fundamental algorithms and theory underlying programming language implementation and for synthesizing these results and those of others in their highly influential books, which educated generations of computer scientists.


Exercise on an α -acyclic join

Let $\mathbf{S} = \{A_1A_2, A_2A_3, \dots, A_{n-1}A_n\}$, where for $i \in \{1, 2, \dots, n-1\}$, A_iA_{i+1} is the following relation.²

A_i	A_{i+1}
1	2
1	4
2	1
2	3
3	2
3	4
4	1
4	3

This relation contains all pairs $\langle i, j \rangle$ where $i, j \in \{1, 2, 3, 4\}$ such that i and j are not both odd and are not both even. Show that:

1. \mathbf{S} is α -acyclic;
2. no tuple of any A_iA_{i+1} is dangling with respect to \mathbf{S} ; and
3. the join $A_1A_2 \bowtie A_2A_3 \bowtie \dots \bowtie A_{n-1}A_n$ contains 2^{n+1} tuples.

²Note that by abuse of notation, we confuse R and $\text{sort}(R)$. 

Exercise on a join that is not α -acyclic

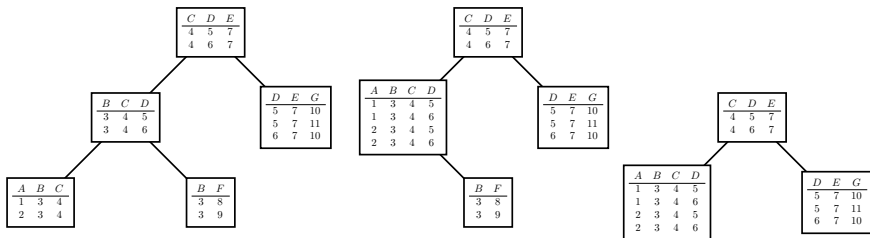
Let $\mathbf{S} = \{A_1A_2, A_2A_3, \dots, A_{n-1}A_n, A_nA_1\}$, where A_iA_{i+1} and A_nA_1 are the “odd-even” relations of the previous question. Show that:

1. \mathbf{S} is α -cyclic;
2. no semijoin program can affect the input relations;
3. any join $A_jA_{j+1} \bowtie A_{j+1}A_{j+2} \bowtie \dots \bowtie A_{\ell-1}A_\ell$ of strictly less than n relations contains $2^{\ell-j+2}$ tuples; and
4. if n is odd, the join $A_1A_2 \bowtie A_2A_3 \bowtie \dots \bowtie A_{n-1}A_n \bowtie A_nA_1$ of n relations is empty.

Question on Yannakakis's algorithm

Show that in step (iii) of Yannakakis's algorithm for $\pi_X (R_1 \bowtie \dots \bowtie R_n)$, we can skip the join, with its parent, of any relation E such that no attribute of E is in X .

For example, for $\pi_{AG} (ABC \bowtie BCD \bowtie BF \bowtie CDE \bowtie DEG)$, we can skip the join of BF with $ABCD$.



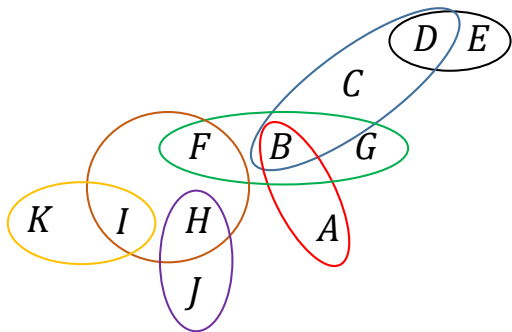
Exercise on Yannakakis's algorithm

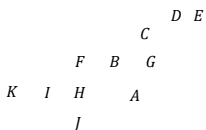
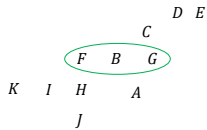
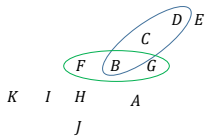
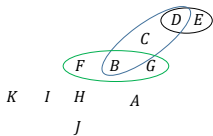
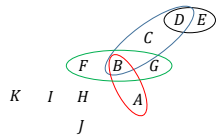
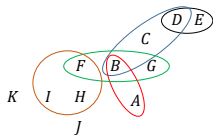
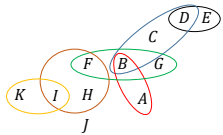
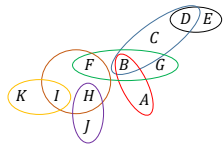
Consider the query

$$\pi_{AEJK} (AB \bowtie BCD \bowtie DE \bowtie BFG \bowtie FHI \bowtie IK \bowtie HJ).$$

1. Construct the hypergraph for the join and show that it is acyclic.
2. Find a parse tree for the hypergraph in which BFG is the root.
3. Construct a full reducer for this join, using the ear-reduction sequence that corresponds to your parse tree from (2).
4. Give the sequence of steps performed by Yannakakis' algorithm after the full reducer sequence of steps from (3).

Solution





FHI
HJ

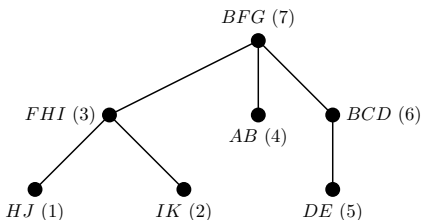
FHI
HJ IK

BFG
FHI
HJ IK

BFG
FHI
HJ IK AB

BFG
FHI
HJ IK AB
BCD
DE

BFG
FHI
HJ IK AB
BCD
DE



$$FHI := FHI \times HJ$$

$$FHI := FHI \times IK$$

$$BFG := BFG \times FHI$$

$$BFG := BFG \times AB$$

$$BCD := BCD \times DE$$

$$BFG := BFG \times BCD$$

$$BCD := BCD \times BFG$$

$$DE := DE \times BCD$$

$$AB := AB \times BFG$$

$$FHI := FHI \times BFG$$

$$IK := IK \times FHI$$

$$HJ := HJ \times FHI$$

$$FHI := \pi_{FHIJ} (FHI \times HJ)$$

$$FHI := \pi_{FHIJK} (FHI \times IK)$$

$$BFG := \pi_{BFGJK} (BFG \times FHI)$$

$$BFG := \pi_{ABFGJK} (BFG \times AB)$$

$$BCD := \pi_{BCDE} (BCD \times DE)$$

$$BFG := \pi_{AEJK} (BFG \times BCD)$$

Same exercise, stated in “Datalog notation”

Consider the conjunctive query

$$\text{Answer}(a, e, j, k) \leftarrow R(a, b), S(b, c, d), T(d, e), U(b, f, g), \\ V(f, h, i), W(i, k), Q(h, j).$$

Give an efficient algorithm to answer this query.

References



Hung Q. Ngo, Christopher Ré, and Atri Rudra.

Skew strikes back: new developments in the theory of join algorithms.

SIGMOD Record, 42(4):5–16, 2013.



Jeffrey D. Ullman.

Principles of Database and Knowledge-Base Systems, Volume II.

Computer Science Press, 1989.