α-Acyclic Joins

Jef Wijsen

September 13, 2024

1 Motivation

Joins in a Distributed Environment

Assume the following relations.[1]

- $M[NN, Field\_of\_Study, Year]$ stores data about students of UMONS. For example, (19950423158, Informatics, BAC3) states that the person with national number 19950423158 is enrolled in BAC3 Informatics. The relation M is stored in Mons.
- $B[NN, Street, Number, City]$ stores the addresses of all Belgian citizens. The relation B is stored in Brussels.

UMONS wants to get the join $M \bowtie B$. Several computations are possible.

1. Transmit relation B from Brussels to Mons, and compute the join at Mons. If we assume ten million Belgians and five thousand students, 99.95% of the transmitted tuples are *dangling*, meaning that they do not join with a tuple from M.
2. Transmit $\pi_{NN}(M)$ (five thousand tuples) from Mons to Brussels. Compute the join $B \bowtie \pi_{NN}(M)$ in Brussels, and transmit the result (five thousand tuples) to Mons. Finally, compute $M \bowtie \big(B \bowtie \pi_{NN}(M)\big)$ in Mons. In this way, only ten thousand tuples are transmitted.

Order of Joins

Assume relations $R[AB]$, $S[BC]$, $T[CD]$, which now reside on a single site. Assume that there are no dangling tuples. We want to join the three relations. Several computations are possible.

1. First compute $R \bowtie T$, which contains $|R| \times |T|$ tuples. Then compute $S \bowtie (R \bowtie T)$, which may contain much less than $|R| \times |T|$ tuples.
2. First compute $R \bowtie S$, then $T \bowtie (R \bowtie S)$. Since there are no dangling tuples, it can be easily seen that the intermediate result will not be larger than the output relation.

Questions

The following questions arise.

1. In a distributed join, can we minimize the amount of tuples transmitted?
2. If we have a join of more than two relations, can we join the relations in a way so as to minimize the size of the intermediate results?

[1] See the course *Bases de Données I* for definitions of relation and the operator $\bowtie$.

2 Preliminaries

We assume *relation names* $R, S, R_1, S_1, R_2, S_2, \ldots$. Each relation name R is associated with a finite set of *attributes*, denoted $\mathsf{sort}(R)$. Letters $A, B, C, \ldots$ denote attributes. We will write $R[X]$ to denote that R is a relation name with $\mathsf{sort}(R) = X$.

A *schema* $\mathbf{S}$ is a finite set of relation names such that for all $R_1, R_2 \in \mathbf{S}$, if $R_1 \neq R_2$, then $\mathsf{sort}(R_1) \neq \mathsf{sort}(R_2)$. Thus, we require that no two distinct relation names are associated with the same set of attributes. This restriction is not fundamental, but simplifies the technical treatment: an element $R[X]$ of a schema is uniquely identified by X.

A *database* over a schema $\mathbf{S}$ associates to each relation name $R \in \mathbf{S}$ a relation over $\mathsf{sort}(R)$.

Whenever a database is fixed, we do not distinguish between the relation name R and the relation associated with R. For example, when we talk about the "join of R and S," we mean the join of the relations associated with R and S.

Also, we will use R as a shorthand for $\mathsf{sort}(R)$. For example, we will write $\pi_R(R \bowtie S)$ instead of $\pi_{\mathsf{sort}(R)}(R \bowtie S)$.

3 Semijoin

Recall that $\mathsf{sort}(R \bowtie S) := \mathsf{sort}(R) \cup \mathsf{sort}(S)$ and $R \bowtie S := \{t \mid t[R] \in R \text{ and } t[S] \in S\}$. The operator $\bowtie$ is commutative and associative.

The *semijoin* of R and S, denoted $R \ltimes S$, is the subset of R containing each tuple of R that joins with some tuple of S. Formally, $R \ltimes S := \pi_R(R \bowtie S)$. A tuple of R that does not belong to $R \ltimes S$ is called *dangling*.

Exercise 1 Show that $R \ltimes S = R \bowtie \pi_{R\cap S}(S)$.

Assume that R and S reside on different sites, and that we want to compute $R \ltimes S$. The amount of transmitted data must be minimized. We can ship S to the site of R. However, the expression of Exercise 1 tells us that it is sufficient to ship $\pi_{R\cap S}(S)$ to the site of R.

4 Joining Two Relations Residing at Different Sites

Show the following.

$$\begin{aligned} R \bowtie S &= (R \ltimes S) \bowtie S & (1) \\ &= (S \ltimes R) \bowtie R & (2) \end{aligned}$$

Assume that R and S reside on different sites, and that we want to compute $R \bowtie S$. Equation (1) tells us that we can compute $R \ltimes S$ as in Section 3, and ship the result to the site of S. In this way, we avoid the transmission of dangling tuples of R. In summary [**?**, p. 701],

1. Compute $\pi_{R\cap S}(S)$ at the site of S.
2. Ship $\pi_{R\cap S}(S)$ to the site of R.
3. Compute $R \ltimes S$ at the site of R, using the fact that $R \ltimes S = R \bowtie \pi_{R\cap S}(S)$.
4. Ship $R \ltimes S$ to the site of S.
5. Compute $R \bowtie S$ at the site of S, using the fact that $R \bowtie S = (R \ltimes S) \bowtie S$.

There is a symmetric strategy, with R and S interchanged.

R	A	B
	1	2
	2	4
	3	6
	4	8

S	B	C
	1	2
	2	4
	3	6
	4	8

T	C	D
	1	2
	2	4
	3	6
	4	8

Figure 1: Three relations to be joined.

R	A	B
	a	b
	d	e

S	B	C
	b	c
	e	f

U	C	A
	c	d
	f	a

Figure 2: Three relations to be joined.

5 Joining Three or More Relations

Let $\mathbf{db}$ be a database over schema $\mathbf{S} = \{R_1, \ldots, R_n\}$. We say that a tuple t of R_i is *dangling* with respect to $\mathbf{S}$ if $t \notin \pi_{R_i}(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n)$. We can try to eliminate dangling tuples by applying semijoins as in Section 4. A *semijoin program* for $\mathbf{S}$ is a sequence of commands

$$\begin{array}{rcl} R_{i_1} & := & R_{i_1} \ltimes R_{j_1}; \\ R_{i_2} & := & R_{i_2} \ltimes R_{j_2}; \\ & \vdots & \\ R_{i_p} & := & R_{i_p} \ltimes R_{j_p}; \end{array}$$

This is called a *full reducer* for $\mathbf{S}$ if for each database $\mathbf{db}$ over $\mathbf{S}$, applying this program yields a database without dangling tuples.

Example 1 Consider the database of Fig. 1 and the semijoin program

$$\begin{array}{ll} R & := R \ltimes S \\ S & := S \ltimes T \\ T & := T \ltimes S \end{array}$$

The first step eliminates tuples $(3, 6)$ and $(4, 8)$ from R, and the second step does the same to S. The third step eliminates $(1, 2)$ and $(3, 6)$ from T. If we then take the join of the three relations, we find that the only tuple in the join $R \bowtie S \bowtie T$ is $(1, 2, 4, 8)$. That is, tuple $(2, 4)$ is still dangling in R and T, and tuple $(1, 2)$ is dangling in S. Thus, this semijoin program is not a full reducer.

Example 2 A full reducer for the relations of Fig. 1 is

$$\begin{array}{rcl} S & := & S \ltimes R \\ T & := & T \ltimes S \\ S & := & S \ltimes T \\ R & := & R \ltimes S \end{array}$$

We will show in Theorem 2 that this program eliminates dangling tuples from R, S, and T independent of the initial values of these relations.

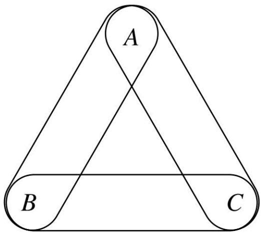


Figure 3: α-Cyclic hypergraph.

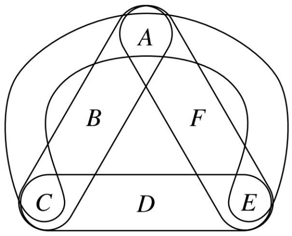


Figure 4: α-Acyclic hypergraph. ABC is an ear that can be removed in favor of ACE, because $ABC \setminus ACE = B$ and B is unique to ABC.

Example 3 Consider the database of Fig. 2. Notice the following.

$$\begin{array}{rcl} R & = & R \ltimes S \\ R & = & R \ltimes U \\ S & = & S \ltimes R \\ S & = & S \ltimes U \\ U & = & U \ltimes R \\ U & = & U \ltimes S \end{array}$$

Since $R \bowtie S \bowtie U = \{\}$, it is correct to conclude that there exists no full reducer for this schema.

Example 3 raises an important question: which schemas have a full reducer?

6 α-Acyclic Schemas

A *hypergraph* is a pair (V, E) where V is a set of vertexes and E is a family of distinct nonempty subsets of V, called *hyperedges*.

The hypergraph of schema $\mathbf{S}$ is the pair (V, E) where $V = \bigcup\{\mathsf{sort}(R) \mid R \in \mathbf{S}\}$ and $E = \{\mathsf{sort}(R) \mid R \in \mathbf{S}\}$.

Let E and F be two hyperedges, and suppose that the attributes of $E \setminus F$ are *unique* to E; that is, they appear in no hyperedge but E. Then we call E an *ear*, and we term the removal of E from the hypergraph in question *ear removal*. We sometimes say "E is removed in favor of F" in this situation. As a special case, if a hyperedge intersects no other hyperedge, then that hyperedge is an ear, and we can remove that hyperedge by "ear removal."

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; otherwise it is α-cyclic.

Exercise 2 Show that the hypergraph of Fig. 3 is α-cyclic, and that the hypergraph of Fig. 4 is α-acyclic.

Theorem 1 *The GYO-reduction of a hypergraph is unique, independent of the sequence of ear removals chosen.*

Proof Note that a potential removal is still possible if another removal is chosen. For example, suppose E_1 could be removed in favor of E_2. That is, the vertices of $E_1 \setminus E_2$ are unique to E_1. We distinguish two cases.

1. If we do an ear removal of a hyperedge other than E_2, we can still remove E_1.
2. Suppose we first remove E_2 in favor of some E_3. It suffices to show $E_1 \setminus E_3 \subseteq E_1 \setminus E_2$, so E_1 is still an ear and can be removed in favor of E_3. Suppose towards a contradiction that there exists a vertex $N \in E_1 \setminus E_3$ such that $N \notin E_1 \setminus E_2$. Then $N \in E_2 \setminus E_3$ and $N \in E_1$ (thus, N is not unique to $E_2 \setminus E_3$), contradicting the assumption that E_2 was an ear that could be removed in favor of E_3.

This concludes the proof. $\square$

Theorem 2 *A schema is α-acyclic if and only if it has a full reducer.*

Proof of the $\Longrightarrow$-direction The proof runs by induction on the cardinality of $\mathbf{S}$. Clearly, if $|\mathbf{S}| = 1$, then the empty semijoin program is a full reducer for $\mathbf{S}$. For the induction step, let $\mathbf{S}$ be an α-acyclic schema with $|\mathbf{S}| \geq 2$. Let $\mathcal{G}$ be the hypergraph of $\mathbf{S}$. Since $\mathcal{G}$ is α-acyclic, we can assume an ear S_1 that can be removed in favor of some hyperedge T_1. Let $\mathcal{H}$ be the resulting hypergraph, which must be α-acyclic. By the induction hypothesis, we can assume a full reducer $P_{\mathcal{H}}$ for $\mathbf{S} \setminus \{S_1\}$. Consider the following semijoin program (call it $P_{\mathcal{G}}$).

$$\begin{array}{c} T_1 \coloneqq T_1 \ltimes S_1; \\ \boxed{\textit{all commands of } P_{\mathcal{H}}} \\ S_1 \coloneqq S_1 \ltimes T_1; \end{array}$$

Let $S_1, \ldots, S_n$ be an ordering of $\mathbf{S}$ corresponding to a sequence of ear removals in a GYO reduction. Since S_1 could be removed in favor of T_1, it follows

$$\mathsf{sort}(S_1) \cap \Big(\bigcup_{i=2}^{n} \mathsf{sort}(S_i)\Big) \subseteq \mathsf{sort}(T_1) \tag{3}$$

We need to show that $P_{\mathcal{G}}$ is a full reducer for $\mathbf{S}$. That is, we need to show that no tuple is dangling with respect to $\mathbf{S}$. We distinguish between tuples from $S_2, \ldots, S_n$, and tuples from S_1.

For every $i \in \{2, \ldots, n\}$, no tuple of S_i is dangling with respect to S. Let $i \in \{2, \ldots, n\}$ and let $s_i \in S_i$. The full reducer $P_{\mathcal{H}}$ ensures that s_i is not dangling with respect to $\{S_2, \ldots, S_n\}$. That is, there exists a tuple $t \in S_2 \bowtie \cdots \bowtie S_n$ such that $t[S_i] = s_i$. The command $T_1 \coloneqq T_1 \ltimes S_1$ of $P_{\mathcal{G}}$ ensures that $t[T_1]$ joins with some tuple $s_1 \in S_1$. Since $T_1 \in \{S_2, \ldots, S_n\}$ and by (3), the tuple s_1 joins with t. It follows that s_i is not dangling with respect to $\mathbf{S}$. Note also that s_1 is not removed by the last command of $P_{\mathcal{G}}$.

No tuple of S_1 is dangling with respect to S. Let $s_1 \in S_1$. The command $S_1 \coloneqq S_1 \ltimes T_1$ of $P_{\mathcal{G}}$ ensures that s_1 joins with some tuple $t_1 \in T_1$. The full reducer $P_{\mathcal{H}}$ ensures that t_1 is not dangling with respect to $\{S_2, \ldots, S_n\}$ (recall that $T_1 \in \{S_2, \ldots, S_n\}$). Thus, there exists $t \in S_2 \bowtie \cdots \bowtie S_n$ such that $t[T_1] = t_1$. By (3), s_1 joins with t, hence s_1 is not dangling with respect to $\mathbf{S}$.

$\square$

Example 4 The following GYO-reduction shows that schema $\mathbf{S} = \{R[AB], S[BC], T[CD]\}$ is α-acyclic.

1. Remove the ear R in favor of S.
2. In $\{S[BC], T[CD]\}$, remove the ear S in favor of T.
3. Remove the ear T.

A full reducer for $\mathbf{S}$ is built "from the inside out."

1. The empty semijoin program is a full reducer for $\{T\}$.
2. A full reducer for $\{S, T\}$ is given by

$$\begin{array}{rcl} T & := & T \ltimes S; \\ S & := & S \ltimes T; \end{array}$$

3. A full reducer for $\{R, S, T\}$ is given by

$$\begin{array}{rcl} S & := & S \ltimes R; \\ T & := & T \ltimes S; \\ S & := & S \ltimes T; \\ R & := & R \ltimes S; \end{array}$$

7 Order of Joins

Let $\mathbf{S}$ be an α-acyclic database schema. Suppose we have applied a full reducer. We must now join all relations. Suppose we have removed $S_1, S_2, \ldots, S_n$ in that order. That is, S_1 was the first ear removed, S_2 was the second ear removed, and so on. Assume that for all $i \in \{1, \ldots, n-1\}$, S_i was removed in favor of $T_i \in \{S_{i+1}, \ldots, S_n\}$. In particular, $T_{n-1} = S_n$. The full reducer in the proof of Theorem 2 is the following.

$$\begin{array}{rcl} T_1 & := & T_1 \ltimes S_1 \\ T_2 & := & T_2 \ltimes S_2 \\ & \vdots & \\ T_{n-1} & := & T_{n-1} \ltimes S_{n-1} \\ S_{n-1} & := & S_{n-1} \ltimes T_{n-1} \\ & \vdots & \\ S_i & := & S_i \ltimes T_i \\ & \vdots & \\ S_2 & := & S_2 \ltimes T_2 \\ S_1 & := & S_1 \ltimes T_1 \end{array}$$

Now we join relations in reverse order, that is,

$$\begin{array}{rcl} \mathit{Result} & := & S_n \\ \mathit{Result} & := & S_{n-1} \bowtie \mathit{Result} \\ \mathit{Result} & := & S_{n-2} \bowtie \mathit{Result} \\ & \vdots & \\ \mathit{Result} & := & S_i \bowtie \mathit{Result} \\ & \vdots & \\ \mathit{Result} & := & S_1 \bowtie \mathit{Result} \end{array}$$

We argue that the size of Result cannot decrease. When we join S_i to $S_{i+1} \bowtie \cdots \bowtie S_n$, we know that every tuple of S_i joins with some tuple of $S_{i+1} \bowtie \cdots \bowtie S_n$, because the command $S_i := S_i \ltimes T_i$ in the full reducer ensures that S_i has no dangling tuples with respect to $\{S_{i+1}, \ldots, S_n\}$. As a consequence, no intermediate join can have more tuples than the output relation.

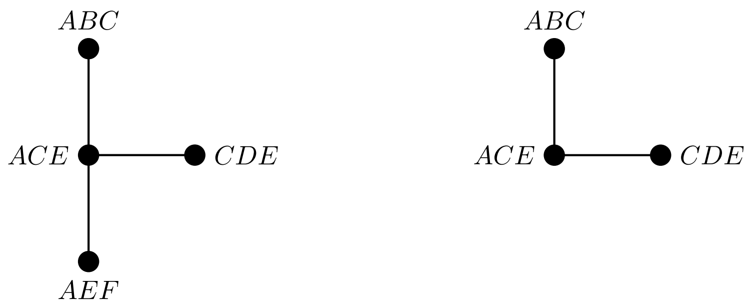


Figure 5: A join tree (left) and the subgraph induced by the vertices containing C (right).

Example 5 We continue Example 4. The GYO-reduction of $\mathbf{S} = \{R[AB], S[BC], T[CD]\}$ shown there removes R, S, T in that order. So the order of the join is $R \bowtie (S \bowtie T)$. The command $S := S \ltimes T$ in the full reducer (see Example 4) ensures that every tuple of S joins with some tuple of T. The command $R := R \ltimes S$ in the full reducer ensures that every tuple of R joins with some tuple of $S \bowtie T$.

8 Join Tree

A *join tree* of a hypergraph $\mathcal{G} = (V, E)$ is a tree (i.e., a connected acyclic undirected graph) whose vertices are the hyperedges of $\mathcal{G}$ such that the following condition holds:

Connectedness Condition: For every $A \in V$, the subgraph of the tree induced by the vertices that contain A is connected.

The connectedness condition is equivalent to saying that for all vertices E_1 and E_2 in the tree (i.e., E_1 and E_2 are hyperedges in the hypergraph), if some $A \in V$ belongs to $E_1 \cap E_2$, then A belongs to every vertex on the (unique) path between E_1 and E_2 in the tree.

Theorem 3 *A hypergraph is α-acyclic if and only if it has a join tree.*

Proof $\boxed{\Longrightarrow}$ Let $\mathcal{G}$ be an α-acyclic hypergraph. Build a tree whose nodes correspond to the hyperedges, and E is a child of F if we eliminate E by ear removal, in favor of F. We show that this tree satisfies the *Connectedness Condition*. Assume towards a contradiction that the *Connectedness Condition* is not satisfied. Then for some A, the tree must contain a simple[2] path $\langle E_1, E_2, \ldots, E_n \rangle$ with $n > 2$ such that $A \in E_1 \cap E_n$ and for $i \in \{2, \ldots, n-1\}$, $A \notin E_i$. We can assume without loss of generality that in the GYO-reduction, the removal of E_1 preceded the removal of E_n. Then, E_1 cannot be a child of E_2, because we cannot have removed E_1 by ear removal in favor of E_2 (because $A \in E_1 \setminus E_2$ also belongs to E_n). So it must be that E_2 is a child of E_1. But then E_n must be a descendant of E_1 (because the path is simple), hence the removal of E_n preceded the removal of E_1 in the GYO-reduction, a contradiction. We conclude by contradiction that the tree satisfies the *Connectedness Condition*.

$\boxed{\Longleftarrow}$ Let τ be a join tree of a hypergraph $\mathcal{G}$. Pick any vertex R (i.e., any hyperedge of $\mathcal{G}$) and consider the rooted tree (τ, R) (i.e., the tree τ in which R is singled out as the root). We show that $\mathcal{G}$ has a GYO-reduction that removes all hyperedges. The proof runs by induction on the number of hyperedges in $\mathcal{G}$. Clearly, if $\mathcal{G}$ has only one hyperedge, then this hyperedge is an ear and can be removed. For the induction step, assume that $\mathcal{G}$ has two or more hyperedges. Assume that E is a leaf and a child of F in the rooted tree (τ, R). For every $A \in E \setminus F$, it must be the case that A is unique to E because of the *Connectedness Condition*. Therefore E can be eliminated by ear removal, in favor of F. Clearly, the tree obtained after removal of E still satisfies the *Connectedness Condition*, and hence, by the induction hypothesis, has a GYO-reduction that removes all

[2]A path is simple if all its vertices are distinct.

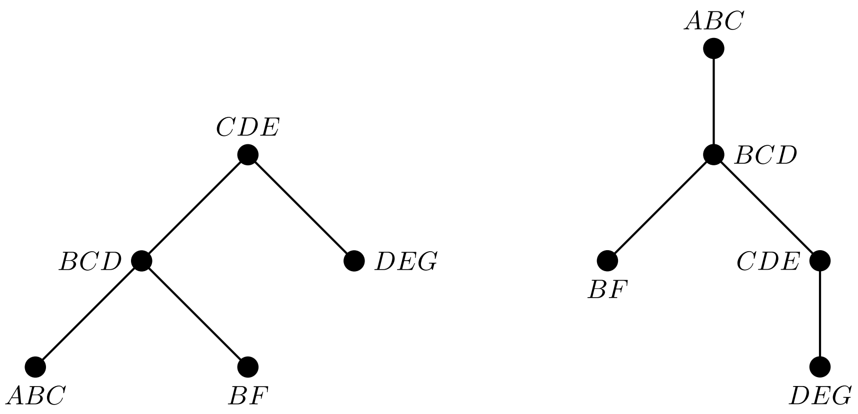


Figure 6: The same join trees with different roots.

hyperedges. □

If $\mathbf{S} = \{R_1, \ldots, R_n\}$ is an acyclic database schema, then the tree built in the proof of Theorem 3 can be viewed as a "parse tree" for the join expression $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$. Notice that the proof of Theorem 3 implies that we may take whichever hyperedge we wish to be the root. See Fig. 6.

9 Computing a Projection of an α-Acyclic Join

We now investigate how to compute a projection of an α-acyclic join. We will first apply a full reducer. Unfortunately, since a projection can reduce the number of tuples, we cannot ensure that no intermediate relation contains more tuples than the final output.

Example 6 The join $R \bowtie S$ contains 8 tuples, but the projection $\pi_{AC}(R \bowtie S)$ contains only 7 tuples.

R: A	B
1	d
2	d
2	e
3	e

S: B	C
d	4
d	5
e	5
e	6

$R \bowtie S$: A	B	C
1	d	4
1	d	5
2	d	4
2	d	5
2	e	5
2	e	6
3	e	5
3	e	6

$\pi_{AC}(R \bowtie S)$: A	C
1	4
1	5
2	4
2	5
2	6
3	5
3	6

We now present Yannakakis' algorithm to compute $\pi_X(R_1 \bowtie \cdots \bowtie R_n)$ where the schema $\mathbf{S} = \{R_1, \ldots, R_n\}$ is acyclic. We will first give the algorithm and then prove that no intermediate relation in its execution will contain more tuples than IU, where I is the total number of tuples in the input relations and U is the number of tuples in the output. That is, the cardinality of all intermediate relations is quadratically bounded by the cardinality of input and output (since $IU \leq (I + U)^2$).

The algorithm consists of the following steps.

(i) Apply a full reducer.

(ii) Construct a rooted join tree for $\mathbf{S}$.

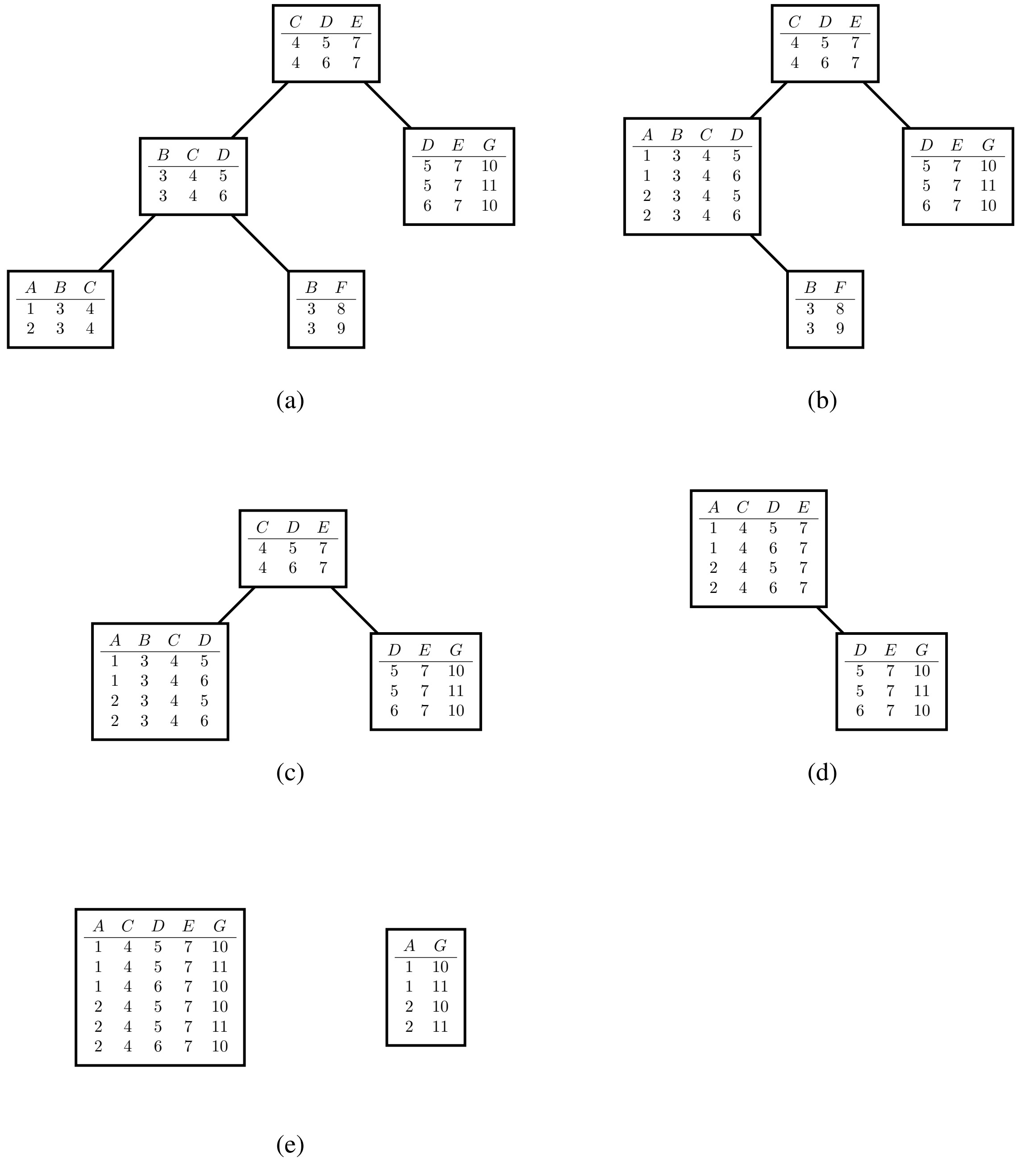


Figure 7: Efficient computation of $\pi_{AG}(ABC \bowtie BCD \bowtie BF \bowtie CDE \bowtie DEG)$ using the rooted join tree of Fig. 6 (left).

(iii) Visit each node of the rooted join tree, other than the root, in some bottom-up order; that is, visit each node after having visited all its children. When we visit E whose parent is F (where $E, F \in \mathbf{S}$), we execute

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

That is, we replace the current relation of F by $\pi_{F \cup (X \cap E)}(E \bowtie F)$.[3]

The projection projects out the attributes that are not in F and are not in X. The attributes that are projected out are not in the final projection and are not needed in any future join. Indeed, if an attribute A is in E but not in F, then, by the *Connectedness Conditions*, A cannot occur in any relation that is still to be visited.

(iv) Project the relation at the root onto X. This step should be performed at the time we join the last child of the root with the root, during step (iii).

Yannakakis' algorithm is illustrated in Fig. 7. Theorem 4 states a bound on the number of tuples in the intermediate relations during the execution of step (iii). It uses the following helping lemma.

Lemma 1 *Let R be a relation name and let X, Y be (not necessarily disjoint) subsets of* $\mathsf{sort}(R)$. *Then,*

1. $\pi_{XY}(R) \sqsubseteq \pi_X(R) \bowtie \pi_Y(R)$*;*
2. *if $Y \subseteq X$, then (the evaluation of) $\pi_Y(R)$ cannot contain more tuples than $\pi_X(R)$.*

Proof Left as an exercise. Note that the inclusion in the first item can be strict. □

Theorem 4 *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 be a relation name in $\{R_1, \ldots, R_n\}$, and let f be the relation that is the original value of F. Note that the value (and the schema) of F changes during the execution of step (iii). It can be easily seen that at all times, the value of F is given by $\pi_{FY}(F \bowtie F_1 \bowtie \cdots \bowtie F_m)$ where

1. $F_1, \ldots, F_m$ constitute the "self-or-descendant" axis of all children C of F such that C has already been visited; and
2. Y contains all (and only) the attributes of X that are in some F_i but not in F. That is, $Y \subseteq X$.

To ease the notation, let $T := F \bowtie F_1 \bowtie \cdots \bowtie F_m$. By Lemma 1, we have $\pi_{FY}(T) \subseteq \pi_F(T) \bowtie \pi_Y(T)$. Since $|\pi_F(T) \bowtie \pi_Y(T)| \leq |\pi_F(T)| \times |\pi_Y(T)|$ is obvious, we have

$$|\pi_{FY}(T)| \leq |\pi_F(T)| \times |\pi_Y(T)|.$$

Now it suffices to show that $|\pi_F(T)| \leq I$ and $|\pi_Y(T)| \leq U$.

$\boxed{\text{Proof that } |\pi_F(T)| \leq I.}$ We have that $\pi_F(T) \subseteq f$, since F is one of the relation names in the join T. Clearly, $|f| \leq I$.

$\boxed{\text{Proof that } |\pi_Y(T)| \leq U.}$ This follows from the following two observations:

1. $\pi_Y(T) = \pi_Y(R_1 \bowtie \cdots \bowtie R_n)$, because we have applied a full reducer in step (i); and
2. $|\pi_Y(R_1 \bowtie \cdots \bowtie R_n)| \leq |\pi_X(R_1 \bowtie \cdots \bowtie R_n)|$ follows from Lemma 1 and $Y \subseteq X$. Notice that $|\pi_X(R_1 \bowtie \cdots \bowtie R_n)| = U$.

□

[3] Notice that if $X \cap E \not\subseteq F$, this also changes the schema of F.

10 Exercises

Exercises taken from [**?**].

1. Let $\mathbf{S} = \{A_1A_2, A_2A_3, \ldots, A_{n-1}A_n\}$, where for $i \in \{1, 2, \ldots, n-1\}$, A_iA_{i+1} is the following relation.[4]

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

   Note that 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:

   (a) $\mathbf{S}$ is α-acyclic;

   (b) no tuple of any A_iA_{i+1} is dangling with respect to $\mathbf{S}$; and

   (c) the join $A_1A_2 \bowtie A_2A_3 \bowtie \cdots \bowtie A_{n-1}A_n$ contains 2^{n+1} tuples.

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

   (a) $\mathbf{S}$ is α-cyclic;

   (b) no semijoin program can affect the input relations;

   (c) any join $A_jA_{j+1} \bowtie A_{j+1}A_{j+2} \bowtie \cdots \bowtie A_{\ell-1}A_\ell$ of strictly less than n relations contains $2^{\ell-j+2}$ tuples; and

   (d) if n is odd, the join $A_1A_2 \bowtie A_2A_3 \bowtie \cdots \bowtie A_{n-1}A_n \bowtie A_nA_1$ of n relations is empty.

3. Show that in step (iii) of Yannakakis's algorithm for $\pi_X(R_1 \bowtie \cdots \bowtie R_n)$, we can skip the join, with its parent, of any relation E such that no attribute of E is in X. For example, in Fig. 7, we can skip the join of BF with $ABCD$.

4. Consider the query

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

   (a) Construct the hypergraph for the join and show that it is acyclic.

   (b) Find a parse tree for the hypergraph in which BFG is the root.

   (c) Construct a full reducer for this join, using the ear-reduction sequence that corresponds to your parse tree from (4b).

   (d) Give the sequence of steps performed by Yannakakis' algorithm after the full reducer sequence of steps from (4c).

5. Consider the conjunctive query

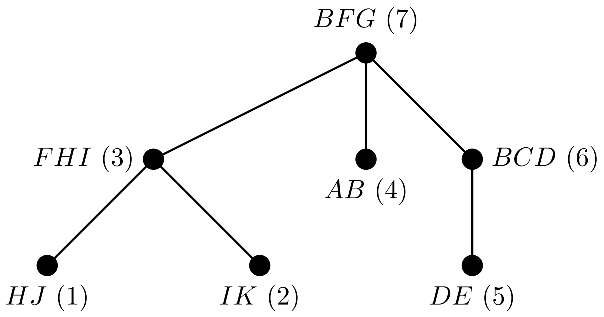
$$\mathsf{Answer}(a, e, j, k) \leftarrow R(a, b), S(b, c, d), T(d, e), U(b, f, g), V(f, h, i), W(i, k), Q(h, j).$$

   Give an efficient algorithm to answer this query.

[4] Note that by abuse of notation, we confuse R and $\mathsf{sort}(R)$.

Partial Solution for Exercise 4

The numbers between parentheses indicate the order in which ears are removed, starting with 1.



$$
\begin{array}{lcl}
FHI & := & FHI \ltimes HJ \\
FHI & := & FHI \ltimes IK \\
BFG & := & BFG \ltimes FHI \\
BFG & := & BFG \ltimes AB \\
BCD & := & BCD \ltimes DE \\
BFG & := & BFG \ltimes BCD \\
BCD & := & BCD \ltimes BFG \\
DE & := & DE \ltimes BCD \\
AB & := & AB \ltimes BFG \\
FHI & := & FHI \ltimes BFG \\
IK & := & IK \ltimes FHI \\
HJ & := & HJ \ltimes FHI
\end{array}
$$

$$
\begin{array}{lcl}
FHI & := & \pi_{FHIJ}(FHI \bowtie HJ) \\
FHI & := & \pi_{FHIJK}(FHI \bowtie IK) \\
BFG & := & \pi_{BFGJK}(BFG \bowtie FHI) \\
BFG & := & \pi_{ABFGJK}(BFG \bowtie AB) \\
BCD & := & \pi_{BCDE}(BCD \bowtie DE) \\
BFG & := & \pi_{AEJK}(BFG \bowtie BCD)
\end{array}
$$