CSE599d: Advanced Query Processing

Lecture 5: Very Large Joins

Dan Suciu

University of Washington

Short Discussion of HW1

Recap: the Homomorphism Theorem

Theorem Let Q_1, Q_2 be Boolean CQs. The following are equivalent:

- Containment holds: $Q_1 \subseteq Q_2$
- There exists a homomorphism $h : Q_2 \rightarrow Q_1$
- $Q_2(\boldsymbol{D}_{Q_1}) = \text{true}$.

$$Q_1 \equiv Q_2 \quad \text{iff} \quad Q_1 \subseteq Q_2 \text{ and } Q_2 \subseteq Q_1$$

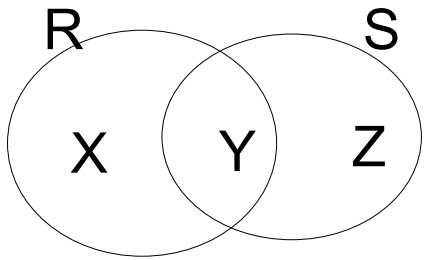
Review from Lecture 1

$$R \bowtie S = (R \ltimes S) \bowtie S$$

Review from Lecture 1

$$R \bowtie S = (R \ltimes S) \bowtie S$$

Denote $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ the set of variables:



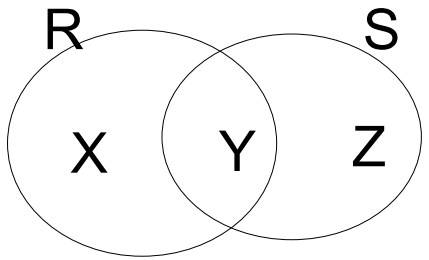
$$\begin{aligned} Q_1(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) &= R(\boldsymbol{X}, \boldsymbol{Y}) \wedge S(\boldsymbol{Y}, \boldsymbol{Z}) \\ Q_2(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) &= (\exists \boldsymbol{Z}\, (R(\boldsymbol{X}, \boldsymbol{Y}) \wedge S(\boldsymbol{Y}, \boldsymbol{Z}))) \wedge S(\boldsymbol{Y}, \boldsymbol{Z}) \\ &= \exists \boldsymbol{U} R(\boldsymbol{X}, \boldsymbol{Y}) \wedge S(\boldsymbol{Y}, \boldsymbol{U}) \wedge S(\boldsymbol{Y}, \boldsymbol{Z}) \end{aligned}$$

We renamed $\exists Z$ to $\exists U$ so it doesn't clash with the head variable $\boldsymbol{Z}$.

Review from Lecture 1

$$R \bowtie S = (R \ltimes S) \bowtie S$$

Denote $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ the set of variables:



$$\begin{aligned} Q_1(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) &= R(\boldsymbol{X}, \boldsymbol{Y}) \wedge S(\boldsymbol{Y}, \boldsymbol{Z}) \\ Q_2(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) &= (\exists \boldsymbol{Z}\, (R(\boldsymbol{X}, \boldsymbol{Y}) \wedge S(\boldsymbol{Y}, \boldsymbol{Z}))) \wedge S(\boldsymbol{Y}, \boldsymbol{Z}) \\ &= \exists \boldsymbol{U} R(\boldsymbol{X}, \boldsymbol{Y}) \wedge S(\boldsymbol{Y}, \boldsymbol{U}) \wedge S(\boldsymbol{Y}, \boldsymbol{Z}) \end{aligned}$$

We renamed $\exists Z$ to $\exists U$ so it doesn't clash with the head variable $\boldsymbol{Z}$.

$h_1 : Q_1 \to Q_2$ maps $(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) \mapsto (\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z})$.

$h_2 : Q_2 \to Q_1$ maps $(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{Y}, \boldsymbol{Z}) \mapsto (\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{Y}, \boldsymbol{Z})$.

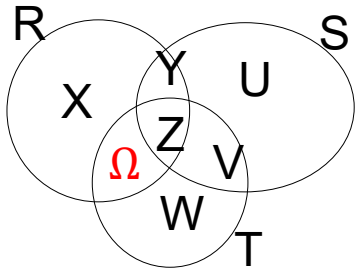
Therefore, $\boxed{Q_1 \equiv Q_2}$.

HW1 Semi-Join Question

If $\text{Vars}(R) \cap \text{Vars}(T) \subseteq \text{Vars}(S)$.
then $\boxed{R \ltimes (S \bowtie T) = R \ltimes (S \ltimes T)}$

HW1 Semi-Join Question

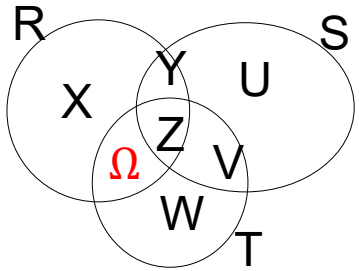
If $\text{Vars}(R) \cap \text{Vars}(T) \subseteq \text{Vars}(S)$.
then $\boxed{R \ltimes (S \bowtie T) = R \ltimes (S \ltimes T)}$



Ω doesn't exist.

HW1 Semi-Join Question

If $\text{Vars}(R) \cap \text{Vars}(T) \subseteq \text{Vars}(S)$.
then $\boxed{R \ltimes (S \bowtie T) = R \ltimes (S \ltimes T)}$



Ω doesn't exist.

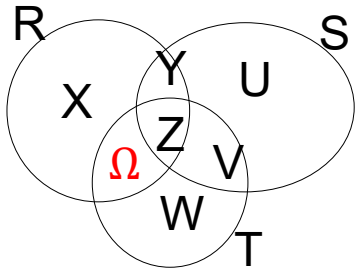
$$Q_1(X, Y, Z, \Omega) = \exists U, V, W(R(X, Y, Z, \Omega) \wedge (S(Y, Z, U, V) \wedge T(Z, V, W, \Omega)))$$

$$Q_2(X, Y, Z, \Omega) = \exists U, V(R(X, Y, Z, \Omega) \wedge \exists W, \Phi(S(Y, Z, U, V) \wedge T(Z, V, W, \Phi$$

If Ω doesn't exist, then $Q_1 \equiv Q_2$ by identity homomorphisms.

HW1 Semi-Join Question

If $\text{Vars}(R) \cap \text{Vars}(T) \subseteq \text{Vars}(S)$.
then $\boxed{R \ltimes (S \bowtie T) = R \ltimes (S \ltimes T)}$



Ω doesn't exist.

$$Q_1(X, Y, Z, \Omega) = \exists U, V, W(R(X, Y, Z, \Omega) \wedge (S(Y, Z, U, V) \wedge T(Z, V, W, \Omega)))$$
$$Q_2(X, Y, Z, \Omega) = \exists U, V(R(X, Y, Z, \Omega) \wedge \exists W, \Phi(S(Y, Z, U, V) \wedge T(Z, V, W, \Phi$$

If Ω doesn't exist, then $Q_1 \equiv Q_2$ by identity homomorphisms.
If Ω does exists, then a counterexample is the canonical database for Q_2:

$\boldsymbol{D} =$

R			
X	Y	Z	Ω
x	y	z	ω

S			
Y	Z	U	V
y	z	u	v

T			
Z	V	W	Ω
z	v	w	ϕ

$Q_1(\boldsymbol{D}) = \emptyset$ $\quad$ $Q_2(\boldsymbol{D}) = \{x, y, z, \omega\}$

Outline for Today

- Quick recap of Query Optimization via Dynamic Programming.
- Optimizing Large Joins.
- The IKKBZ Algorithm for left-linear plans.

Dynamic Programming

$\boldsymbol{R} = \{R_1, \ldots, R_n\}$

- DPsize: all sets $S_1, S_2 \subseteq \boldsymbol{R}$. runtime??
- DPsub: all disjoint sets $S_1, S_2 \subseteq \boldsymbol{R}$, $S_1 \cap S_2 = \emptyset$. runtime??
- DPccp Connected Component Pairs S_1, S_2.
  - Chain query: runtime??
  - Star query: runtime??

Dynamic Programming

$\boldsymbol{R} = \{R_1, \ldots, R_n\}$

- DPsize: all sets $S_1, S_2 \subseteq \boldsymbol{R}$. $O(4^n)$
- DPsub: all disjoint sets $S_1, S_2 \subseteq \boldsymbol{R}$, $S_1 \cap S_2 = \emptyset$. runtime??
- DPccp Connected Component Pairs S_1, S_2.
  - Chain query: runtime??
  - Star query: runtime??

Dynamic Programming

$\boldsymbol{R} = \{R_1, \ldots, R_n\}$

- DPsize: all sets $S_1, S_2 \subseteq \boldsymbol{R}$. $O(4^n)$
- DPsub: all disjoint sets $S_1, S_2 \subseteq \boldsymbol{R}$, $S_1 \cap S_2 = \emptyset$. $O(3^n)$
- DPccp Connected Component Pairs S_1, S_2.
  - Chain query: runtime??
  - Star query: runtime??

Dynamic Programming

$\boldsymbol{R} = \{R_1, \ldots, R_n\}$

- DPsize: all sets $S_1, S_2 \subseteq \boldsymbol{R}$. $O(4^n)$
- DPsub: all disjoint sets $S_1, S_2 \subseteq \boldsymbol{R}$, $S_1 \cap S_2 = \emptyset$. $O(3^n)$
- DPccp Connected Component Pairs S_1, S_2.
  - Chain query: $O(n^3)$
  - Star query: runtime??

Dynamic Programming

$\boldsymbol{R} = \{R_1, \ldots, R_n\}$

- DPsize: all sets $S_1, S_2 \subseteq \boldsymbol{R}$. $O(4^n)$
- DPsub: all disjoint sets $S_1, S_2 \subseteq \boldsymbol{R}$, $S_1 \cap S_2 = \emptyset$. $O(3^n)$
- DPccp Connected Component Pairs S_1, S_2.
  - Chain query: $O(n^3)$
  - Star query: $O(3^n)$

Recap: Enumerate Connected Subgraphs

Algorithm 1: EnumerateCsg($G = (V, E)$)

for $i = n-1, n-2, \ldots, 1, 0$ **do**
  emit($\{v_i\}$);
  EnumerageCsgRec($G, \{v_i\}, \{v_0, \ldots, v_i\}$);

Algorithm 2: EnumerageCsgRec(G, S, X)

Output: Enumerate connected subgraphs $\supseteq S$, disjoint from X

$N := \mathcal{N}(S) - X$; ; // $\mathcal{N}(S)$ = neighbors of S

for $\emptyset \subsetneq S' \subseteq N$ **do**
  emit($S \cup S'$);
  EnumerageCsgRec($G, (S \cup S'), (X \cup N)$);

For each S_1, we also enumerate S_2. (The function with Errata!)

Recap: Optimization based on Connected Component Pairs

Algorithm 3: DPccp

for Connected component pairs $S_1, S_2 \subseteq \boldsymbol{R}$ **do**
  if APPLICABLE$(S_1, S_2, \circ)$ **then**
    $(P_1, c_1) :=$ PlanCost$[S_1]$;
    $(P_2, c_2) :=$ PlanCost$[S_2]$;
    $c := c_1 + c_2 + \text{Cost}(P_1 \circ P_2)$;
    if $c <$ PlanCost$[S_1 \cup S_2]$.cost **then**
      PlanCost$[S_1 \cup S_2] := (P_1 \circ P_2, c)$;

Discussion

- Complexity: $O(n^3) \cdots O(3^n)$. Hence $n \leq 14 \cdots 100$
- Large joins ($n \gg 100$) occur because of views over views over views.
- Rare, but they exists [Neumann and Radke, 2018]:
  - SAP Hana: query with 4,598, one with 2,298, a few with > 1000 relations.
  - Tableau Public Data: long tail of queries with > 100 relations, up to 369.
- Query optimizer needs to cope with large joins too.

Large Joins

Overview

[Neumann and Radke, 2018]:

- Small query: DPccp. — Optimal
- Medium query ($\leq$ 100 tables): Linearized DPccp. — Almost Optimal
- Large query: Greedy. — Degrade Gracefully

Small Query

What is Small?

Runtime of DPccp depends heavily on the query's topology:

- Linear query (easiest): $O(n^3)$ $n \leq 100$
- Star query (hardest): $O(3^n)$ $n \leq 14$.

A workable definition of "small":

number of connected subgraphs $csg \leq 10000$

csg can be computed efficiently.

Computing the Number oF Connected Subgraphs

Almost identical to EnumerateCsg

Function csg(G, budget) $G = (V, E)$

$c := 0$;
for $i = n-1, n-2, \ldots, 1, 0$ **do**
  if $c{+}{+} >$ budget **then return** c;
  $c :=$ csgRec($G, \{v_i\}, \{v_0, \ldots, v_i\}, c$, budget);

Function csgRec(G, S, X, c, budget)

$N := \mathcal{N}(S) - X$;
for $\emptyset \subsetneq S' \subseteq N$ **do**
  if $c{+}{+} >$ budget **then return** c;
  $c :=$ csgRec($G, (S \cup S'), (X \cup N), c$, budget);

Small Queries: Discussion

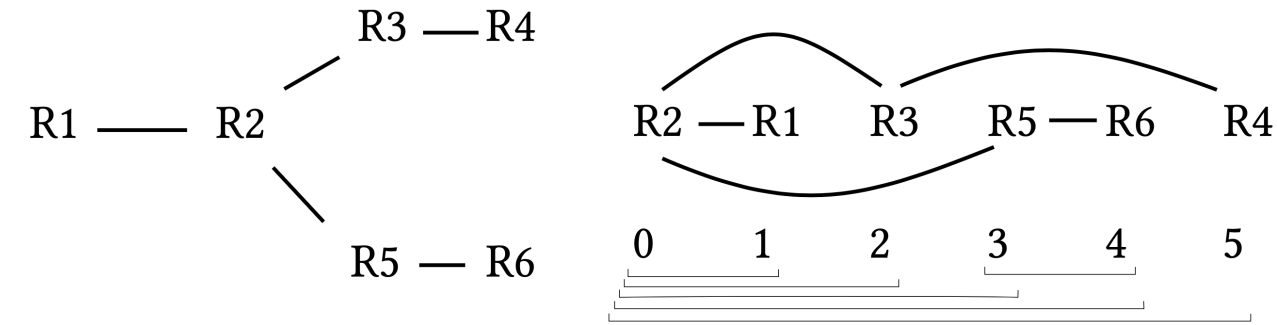
- When $csg \leq 10000$ then run DPccp.
- Optimizer returns the optimal plan.
- Notice that csg is the size of DP table.

Medium Query

Query Linearization

- Fix a linear order on the relations: $R_1, R_2, \ldots, R_n$.
- Modify DPccp to consider only sets S that are connected subchains.
- Size of the DP table: $O(n^2)$ as opposed to $O(2^n)$.

Query Linearization



enumerations by full DP:

{R1,R2}, {R2,R3}, {R2,R5}, {R3,R4}, {R5,R6}, {R1,R2,R3}, {R1,R2,R5}, {R2,R3,R4}, {R2,R3,R5}, {R2,R5,R6}, {R1,R2,R3,R4}, {R1,R2,R3,R5}, {R1,R2,R5,R6}, {R2,R3,R4,R5}, {R1,R2,R3,R4,R5}, {R2,R3,R4,R5,R6}, {R1,R2,R3,R4,R5,R6}

enumerations by linearized DP:

{R1,R2}, {R5,R6}, {R1,R2,R3}, {R1,R2,R3,R5}, {R1,R2,R3,R5,R6}, {R1,R2,R3,R4,R5,R6}

Figure 4: Example for Search Space Linearization

Reduces the size of the DP table from $O(2^n)$ to $O(n^2)$.

LinearizedDP

Algorithm 4: LinearizedDP

Data: Total Order on Relations: $R_1, R_2, \ldots, R_n$
Data: Consider only sets $S = \{R_i, R_{i+1}, \ldots, R_{i+s-1}\}$
for $s_1, s_2 := 1, n-1$: $s_1 + s_2 \leq n$ **do**
  for $i := 1, n - s_1 - s_2$ **do**
    $S_1 := \{R_i, \ldots, R_{i+s_1-1}\}$;
    $S_2 := \{R_{i+s_1}, \ldots, R_{i+s_1+s_2-1}\}$;
    if S_1, S_2 are connected and joinable **then**
      $(P_1, c_1) :=$ PlanCost$[S_1]$;
      $(P_2, c_2) :=$ PlanCost$[S_2]$;
      $c := c_1 + c_2 +$ Cost$(P_1 \circ P_2)$;
      if $c <$ PlanCost$[S_1 \cup S_2]$.cost **then**
        PlanCost$[S_1 \cup S_2] := (P_1 \circ P_2, c)$;

Runtime: $\boxed{O(n^3)}$

Discussion

- Linearizaton reduces runtime from $O(3^n)$ to $O(n^3)$.
- Output plan is no longer optimal: it depends on the linearization order.

How do we choose a good linearization order?

Answer: use the IKKBZ algorithm (coming up).

Large Query

Large Query

- $n > 100$.
- Use a simple Greedy algorithm to compute an initial plan P.
- Iterative refinement: optimize subtrees of size k (e.g. $k = 100$).

Greedy Plan

Given $n \approx 1000$ relations $R_1, \ldots, R_n$,

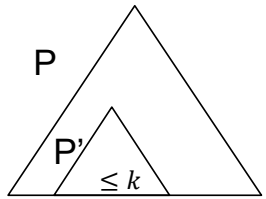
describe a simple greedy algorithm to compute a bushy plan without cross products.

Greedy Plan

[Fegaras, 1998]

- Start from base tables: $P_1 := \{R_1\}, \ldots, P_n := \{R_n\}$.
- Choose two join-able plans P_i, P_j such that $\text{Cost}(P_i \bowtie P_j)$ is minimal.
- Replace P_i, P_j with a new plan $P_k := P_i \bowtie P_j$. Repeat.
- Stop when there is a single plan, with all relations.

Iterative Refinement



- Start from complete join plan P.
- Find a subplan (subtree) P' such that:
  - $|P'| \leq 100$ (so we can run LinearizedDP).
  - Cost(P') is maximal.
- Replace P' with LinearizedDP(P')
- Replace P' with one relation (so we don't optimize it further). Repeat.
- Stop after some fixed number of iterations.

Summary of Large Joins from [Neumann and Radke, 2018]

- Small query: DPccp.
- Medium query: LinearizedDP. Need IKKBZ (next)
- Large query: Greedy, then local LinearizedDP

The IKKBZ Algorithm

Overview of IKKBZ

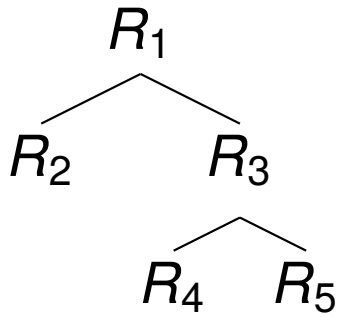
Optimal left linear plan w/o cross products, time $O(n^2)$. Assumes ASI property.

- ASI originates in the job scheduling literature [Monma and Sidney, 1979].
- Adapted to query optimization [Ibaraki and Kameda, 1984].
- Further refined [Krishnamurthy et al., 1986]: we follow this.
- Called IKKBZ in [Neumann and Radke, 2018].

Problem Setting

Input: acyclic query graph.

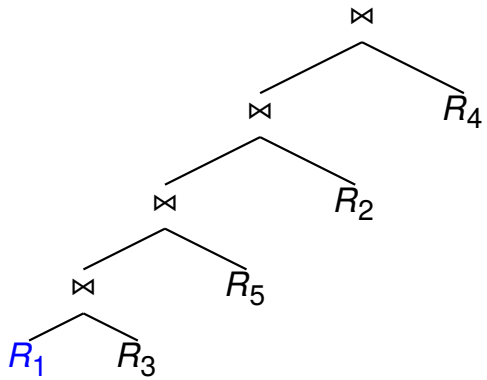
R_1 joins with R_2 and R_3.
R_1 does not join with R_4 or R_5.



Goal: find optimal left linear plan w/o cross products

For each i, consider all topological orders of tree rooted at R_i

Root R_1, e.g. left-linear plan: $R_1R_3R_5R_2R_4$
Root R_3, e.g. left-linear plan: $R_3R_4R_1R_5R_2$



Assumptions on the Cost Function

Fix a root of the query, say R_1. Set $p(i) \stackrel{\text{def}}{=} \text{parent}(i)$.

$$\boxed{\texttt{Est}(R_{p(i)} \ltimes R_i) \stackrel{\text{def}}{=} \theta_i \cdot |R_{p(i)}| \cdot |R_i|}$$

$$\boxed{\text{Cost}(R \ltimes R_i) \stackrel{\text{def}}{=} |R| \cdot g(|R_i|)}$$

Assumptions on the Cost Function

Fix a root of the query, say R_1. Set $p(i) \stackrel{\text{def}}{=} \text{parent}(i)$.

$$\boxed{\texttt{Est}(R_{p(i)} \ltimes R_i) \stackrel{\text{def}}{=} \theta_i \cdot |R_{p(i)}| \cdot |R_i|}$$

$$\boxed{\text{Cost}(R \ltimes R_i) \stackrel{\text{def}}{=} |R| \cdot g(|R_i|)}$$

$\texttt{Est}(\ltimes_{i=k,m} R_i) = \prod_{i=k,m} \theta_i |R_i|$

Notice $\texttt{Est}(R_i) = \theta_i |R_i|$.

$\text{Cost}(\ltimes_{i=k,m} R_i) =$
$\sum_{j=k+1,m} \prod_{i=k,j-1} (\theta_i |R_i|) \cdot g(|R_j|)$

Assumptions on the Cost Function

Fix a root of the query, say R_1. Set $p(i) \stackrel{\text{def}}{=} \text{parent}(i)$.

$$\boxed{\texttt{Est}(R_{p(i)} \ltimes R_i) \stackrel{\text{def}}{=} \theta_i \cdot |R_{p(i)}| \cdot |R_i|}$$

$$\boxed{\text{Cost}(R \ltimes R_i) \stackrel{\text{def}}{=} |R| \cdot g(|R_i|)}$$

$\texttt{Est}(\ltimes_{i=k,m} R_i) = \prod_{i=k,m} \theta_i |R_i|$

Notice $\texttt{Est}(R_i) = \theta_i |R_i|$.

$\text{Cost}(\ltimes_{i=k,m} R_i) =$
$\sum_{j=k+1,m} \prod_{i=k,j-1} (\theta_i |R_i|) \cdot g(|R_j|)$

$$\boxed{\begin{aligned} \texttt{Est}(S_1 S_2) &= \texttt{Est}(S_1) \cdot \texttt{Est}(S_2) \\ \text{Cost}(S_1 S_2) &= \text{Cost}(S_1) + \texttt{Est}(S_1) \cdot \text{Cost}(S_2) \end{aligned}}$$

E.g. $\text{Cost}(R_4R_5R_6R_7R_8) = \text{Cost}(R_4R_5R_6) + \texttt{Est}(R_4R_5R_6) \cdot \text{Cost}(R_7R_8)$

The ASI Property

Cost satisfies the Adjacent Sequence Interchange (ASI) property
if there exists a function Rank such that for all sequences A, B, U, V:

$$\boxed{\text{Rank}(U) \leq \text{Rank}(V)} \quad \text{iff} \quad \boxed{\text{Cost}(AUVB) \leq \text{Cost}(AVUB)}$$

The ASI Property for Query Optimization

Theorem Cost satisfies ASI with $\boxed{\text{Rank}(S) \stackrel{\text{def}}{=} \frac{\texttt{Est}(S)-1}{\text{Cost}(S)}}$

Proof to check $\boxed{\text{Cost}(AUVB) \leq \text{Cost}(AVUB)}$ iff $\boxed{\text{Rank}(U) \leq \text{Rank}(V)}$

The ASI Property for Query Optimization

Theorem Cost satisfies ASI with $\boxed{\text{Rank}(S) \stackrel{\text{def}}{=} \frac{\texttt{Est}(S)-1}{\text{Cost}(S)}}$

Proof to check $\boxed{\text{Cost}(AUVB) \leq \text{Cost}(AVUB)}$ iff $\boxed{\text{Rank}(U) \leq \text{Rank}(V)}$

$$\text{Cost}(AUVB) = \text{Cost}(A) + \texttt{Est}(A) \cdot \text{Cost}(U) + \texttt{Est}(AU) \cdot \text{Cost}(V) + \texttt{Est}(AUV) \cdot \text{Cost}(B)$$

$$\text{Cost}(AVUB) = \text{Cost}(A) + \texttt{Est}(A) \cdot \text{Cost}(V) + \texttt{Est}(AV) \cdot \text{Cost}(U) + \texttt{Est}(AVU) \cdot \text{Cost}(B)$$

The ASI Property for Query Optimization

Theorem Cost satisfies ASI with $\boxed{\text{Rank}(S) \stackrel{\text{def}}{=} \frac{\texttt{Est}(S)-1}{\text{Cost}(S)}}$

Proof to check $\boxed{\text{Cost}(AUVB) \leq \text{Cost}(AVUB)}$ iff $\boxed{\text{Rank}(U) \leq \text{Rank}(V)}$

$$\text{Cost}(AUVB) = \text{Cost}(A) + \texttt{Est}(A) \cdot \text{Cost}(U) + \texttt{Est}(AU) \cdot \text{Cost}(V) + \texttt{Est}(AUV) \cdot \text{Cost}(B)$$
$$\text{Cost}(AVUB) = \text{Cost}(A) + \texttt{Est}(A) \cdot \text{Cost}(V) + \texttt{Est}(AV) \cdot \text{Cost}(U) + \texttt{Est}(AVU) \cdot \text{Cost}(B)$$

$$\text{Cost}(AUVB) \leq \text{Cost}(AVUB)$$
$$\text{iff} \quad \texttt{Est}(A) \cdot \text{Cost}(U) + \texttt{Est}(AU) \cdot \text{Cost}(V) \leq \texttt{Est}(A) \cdot \text{Cost}(V) + \texttt{Est}(AV) \cdot \text{Cost}(U)$$

The ASI Property for Query Optimization

Theorem Cost satisfies ASI with $\boxed{\text{Rank}(S) \stackrel{\text{def}}{=} \frac{\texttt{Est}(S)-1}{\text{Cost}(S)}}$

Proof to check $\boxed{\text{Cost}(AUVB) \leq \text{Cost}(AVUB)}$ iff $\boxed{\text{Rank}(U) \leq \text{Rank}(V)}$

$$\text{Cost}(AUVB) = \text{Cost}(A) + \texttt{Est}(A) \cdot \text{Cost}(U) + \texttt{Est}(AU) \cdot \text{Cost}(V) + \texttt{Est}(AUV) \cdot \text{Cost}(B)$$
$$\text{Cost}(AVUB) = \text{Cost}(A) + \texttt{Est}(A) \cdot \text{Cost}(V) + \texttt{Est}(AV) \cdot \text{Cost}(U) + \texttt{Est}(AVU) \cdot \text{Cost}(B)$$

$$\text{Cost}(AUVB) \leq \text{Cost}(AVUB)$$
$$\text{iff} \quad \texttt{Est}(A) \cdot \text{Cost}(U) + \texttt{Est}(AU) \cdot \text{Cost}(V) \leq \texttt{Est}(A) \cdot \text{Cost}(V) + \texttt{Est}(AV) \cdot \text{Cost}(U)$$
$$\text{iff} \quad \text{Cost}(U) + \texttt{Est}(U) \cdot \text{Cost}(V) \leq \text{Cost}(V) + \texttt{Est}(V) \cdot \text{Cost}(U)$$

The ASI Property for Query Optimization

Theorem Cost satisfies ASI with $\boxed{\text{Rank}(S) \stackrel{\text{def}}{=} \frac{\texttt{Est}(S)-1}{\text{Cost}(S)}}$

Proof to check $\boxed{\text{Cost}(AUVB) \leq \text{Cost}(AVUB)}$ iff $\boxed{\text{Rank}(U) \leq \text{Rank}(V)}$

$$\text{Cost}(AUVB) = \text{Cost}(A) + \texttt{Est}(A) \cdot \text{Cost}(U) + \texttt{Est}(AU) \cdot \text{Cost}(V) + \texttt{Est}(AUV) \cdot \text{Cost}(B)$$
$$\text{Cost}(AVUB) = \text{Cost}(A) + \texttt{Est}(A) \cdot \text{Cost}(V) + \texttt{Est}(AV) \cdot \text{Cost}(U) + \texttt{Est}(AVU) \cdot \text{Cost}(B)$$

$$\begin{aligned}
& \text{Cost}(AUVB) \leq \text{Cost}(AVUB) \\
\text{iff} \quad & \texttt{Est}(A) \cdot \text{Cost}(U) + \texttt{Est}(AU) \cdot \text{Cost}(V) \leq \texttt{Est}(A) \cdot \text{Cost}(V) + \texttt{Est}(AV) \cdot \text{Cost}(U) \\
\text{iff} \quad & \text{Cost}(U) + \texttt{Est}(U) \cdot \text{Cost}(V) \leq \text{Cost}(V) + \texttt{Est}(V) \cdot \text{Cost}(U) \\
\text{iff} \quad & \texttt{Est}(U) \cdot \text{Cost}(V) - \text{Cost}(V) \leq \texttt{Est}(V) \cdot \text{Cost}(U) - \text{Cost}(U)
\end{aligned}$$

The ASI Property for Query Optimization

Theorem Cost satisfies ASI with $\boxed{\text{Rank}(S) \stackrel{\text{def}}{=} \frac{\texttt{Est}(S)-1}{\text{Cost}(S)}}$

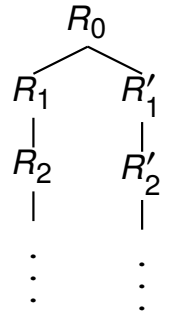
Proof to check $\boxed{\text{Cost}(AUVB) \leq \text{Cost}(AVUB)}$ iff $\boxed{\text{Rank}(U) \leq \text{Rank}(V)}$

$$\begin{aligned}\text{Cost}(AUVB) =& \text{Cost}(A) + \texttt{Est}(A)\cdot\text{Cost}(U) + \texttt{Est}(AU)\cdot\text{Cost}(V) + \texttt{Est}(AUV)\cdot\text{Cost}(B)\\ \text{Cost}(AVUB) =& \text{Cost}(A) + \texttt{Est}(A)\cdot\text{Cost}(V) + \texttt{Est}(AV)\cdot\text{Cost}(U) + \texttt{Est}(AVU)\cdot\text{Cost}(B)\end{aligned}$$

$$\begin{aligned}& & \text{Cost}(AUVB) \leq& \text{Cost}(AVUB)\\ \text{iff} & \quad \texttt{Est}(A)\cdot\text{Cost}(U) + \texttt{Est}(AU)\cdot\text{Cost}(V) \leq& \texttt{Est}(A)\cdot\text{Cost}(V) + \texttt{Est}(AV)\cdot\text{Cost}(U)\\ \text{iff} & \quad \text{Cost}(U) + \texttt{Est}(U)\cdot\text{Cost}(V) \leq& \text{Cost}(V) + \texttt{Est}(V)\cdot\text{Cost}(U)\\ \text{iff} & \quad \texttt{Est}(U)\cdot\text{Cost}(V) - \text{Cost}(V) \leq& \texttt{Est}(V)\cdot\text{Cost}(U) - \text{Cost}(U)\\ \text{iff} & \quad \text{Rank}(U) = \frac{\texttt{Est}(U)-1}{\text{Cost}(U)} \leq& \frac{\texttt{Est}(V)-1}{\text{Cost}(V)} = \text{Rank}(V)\end{aligned}$$

The IKKBZ Algorithm

- Choose node R_0 with two chain children.



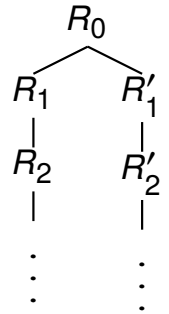
- By induction:

$$\text{Rank}(R_1) \leq \text{Rank}(R_2) \leq \cdots$$
$$\text{Rank}(R'_1) \leq \text{Rank}(R'_2) \leq \cdots$$

[1] No future S can go between R_0 and R''_i why?. Either $SR_{0:i}$ or $R_{0:i}S$.

The IKKBZ Algorithm

- Choose node R_0 with two chain children.

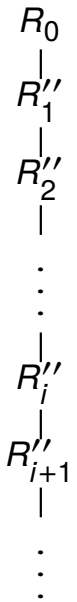


- By induction:

$$\text{Rank}(R_1) \leq \text{Rank}(R_2) \leq \cdots$$
$$\text{Rank}(R'_1) \leq \text{Rank}(R'_2) \leq \cdots$$

- Merge the two chains to obtain:

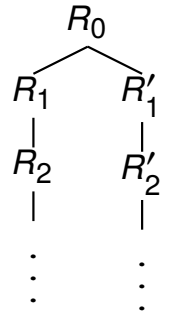
$$\text{Rank}(R_0) \overset{??}{\leqslant} \text{Rank}(R''_1) \leq \text{Rank}(R''_2) \leq \cdots$$



[1] No future S can go between R_0 and R''_i why?. Either $SR_{0:i}$ or $R_{0:i}S$.

The IKKBZ Algorithm

- Choose node R_0 with two chain children.

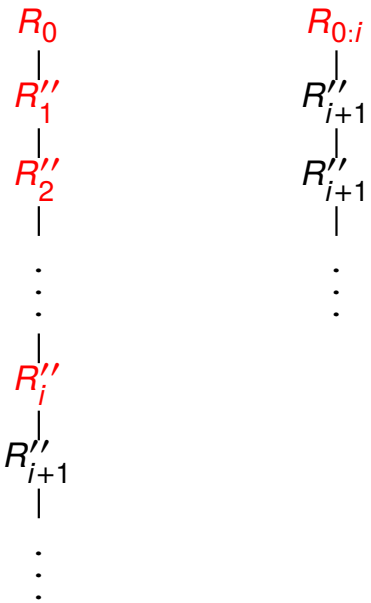


- By induction:

$$\text{Rank}(R_1) \le \text{Rank}(R_2) \le \cdots$$
$$\text{Rank}(R'_1) \le \text{Rank}(R'_2) \le \cdots$$

- Merge the two chains to obtain:

$$\text{Rank}(R_0) \overset{??}{\le} \text{Rank}(R''_1) \le \text{Rank}(R''_2) \le \cdots$$



- Cannot move R_0. Instead create new relation:[1]

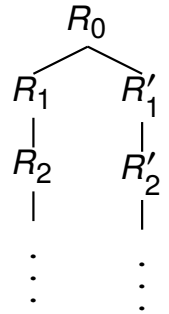
$$\boxed{R_{0:i} := R_0 R''_1 \cdots R''_i} \text{ where}$$

$\text{Rank}(R_0) > \text{Rank}(R''_i)$, $\text{Rank}(R_0) \le \text{Rank}(R''_{i+1})$.

[1] No future S can go between R_0 and R''_i why?. Either $SR_{0:i}$ or $R_{0:i}S$.

The IKKBZ Algorithm

- Choose node R_0 with two chain children.

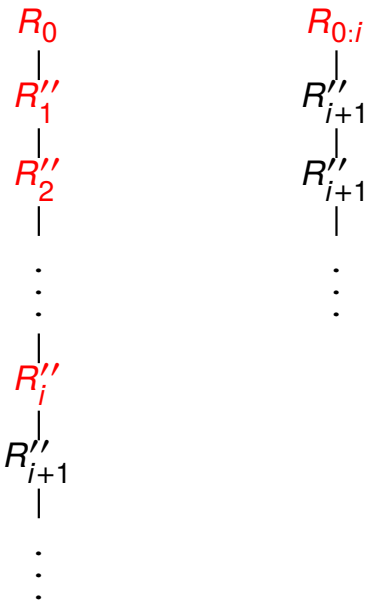


- By induction:

$$\text{Rank}(R_1) \le \text{Rank}(R_2) \le \cdots$$
$$\text{Rank}(R'_1) \le \text{Rank}(R'_2) \le \cdots$$

- Merge the two chains to obtain:

$$\text{Rank}(R_0) \overset{??}{\le} \text{Rank}(R''_1) \le \text{Rank}(R''_2) \le \cdots$$



- Cannot move R_0. Instead create new relation:[1]

$$\boxed{R_{0:i} := R_0 R''_1 \cdots R''_i} \text{ where}$$

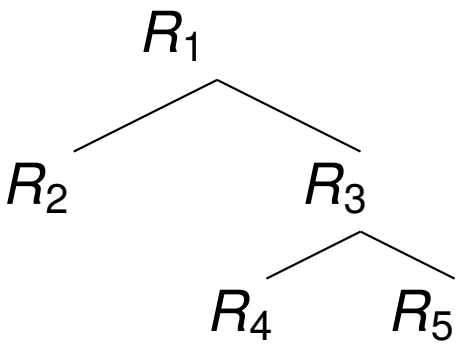
$\text{Rank}(R_0) > \text{Rank}(R''_i)$, $\text{Rank}(R_0) \le \text{Rank}(R''_{i+1})$.

- Repeat, until entire tree becomes a single chain.

[1] No future S can go between R_0 and R''_i why?. Either $SR_{0:i}$ or $R_{0:i}S$.

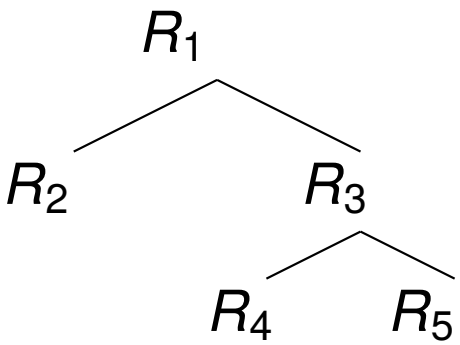
Example: Adapted from [Krishnamurthy et al., 1986]

	$\lvert R_i \rvert$	θ_i	$\text{Cost}(R_i) = g_i(\lvert R_i \rvert)$
R_1	100	1	300
R_2	2000	0.5	9
R_3	1000	0.1	9.9
R_4	400	0.25	3
R_5	100	0.1	30



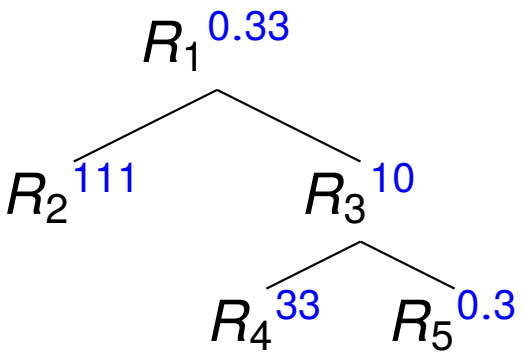
Example: Adapted from [Krishnamurthy et al., 1986]

	$\lvert R_i \rvert$	θ_i	$\text{Cost}(R_i) = g_i(\lvert R_i \rvert)$	$\texttt{Est}(R_i) = \theta_i \lvert R_i \rvert$	$\text{Rank}(R_i) = \frac{\texttt{Est}(R_i)-1}{\text{Cost}(R_i)}$
R_1	100	1	300	100	0.33
R_2	2000	0.5	9	1000	111
R_3	1000	0.1	9.9	100	10
R_4	400	0.25	3	100	33
R_5	100	0.1	30	10	0.3



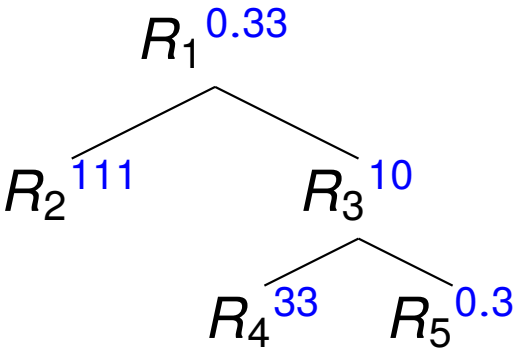
Example: Adapted from [Krishnamurthy et al., 1986]

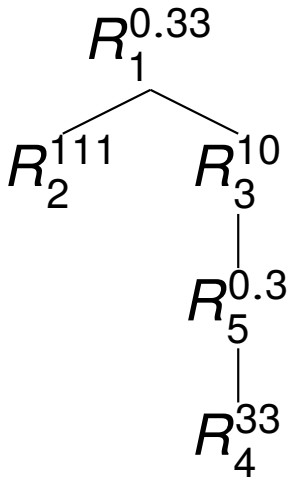
	$\lvert R_i \rvert$	θ_i	$\text{Cost}(R_i) = g_i(\lvert R_i \rvert)$	$\texttt{Est}(R_i) = \theta_i \lvert R_i \rvert$	$\text{Rank}(R_i) = \frac{\texttt{Est}(R_i)-1}{\text{Cost}(R_i)}$
R_1	100	1	300	100	0.33
R_2	2000	0.5	9	1000	111
R_3	1000	0.1	9.9	100	10
R_4	400	0.25	3	100	33
R_5	100	0.1	30	10	0.3



Example: Adapted from [Krishnamurthy et al., 1986]

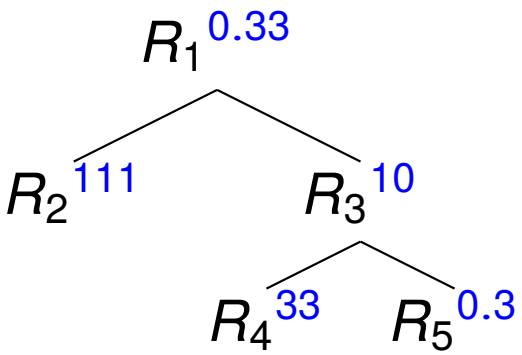
	$\lvert R_i \rvert$	θ_i	$\text{Cost}(R_i) = g_i(\lvert R_i \rvert)$	$\texttt{Est}(R_i) = \theta_i \lvert R_i \rvert$	$\text{Rank}(R_i) = \frac{\texttt{Est}(R_i)-1}{\text{Cost}(R_i)}$
R_1	100	1	300	100	0.33
R_2	2000	0.5	9	1000	111
R_3	1000	0.1	9.9	100	10
R_4	400	0.25	3	100	33
R_5	100	0.1	30	10	0.3

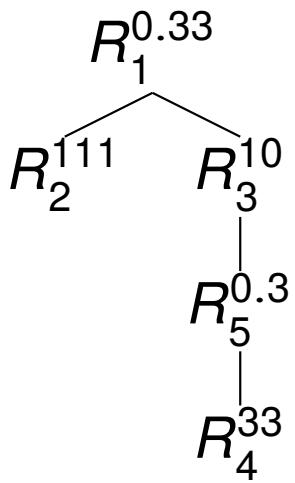


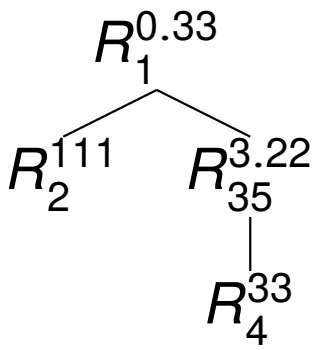


Example: Adapted from [Krishnamurthy et al., 1986]

	$\lvert R_i \rvert$	θ_i	$\text{Cost}(R_i) = g_i(\lvert R_i \rvert)$	$\texttt{Est}(R_i) = \theta_i \lvert R_i \rvert$	$\text{Rank}(R_i) = \frac{\texttt{Est}(R_i)-1}{\text{Cost}(R_i)}$
R_1	100	1	300	100	0.33
R_2	2000	0.5	9	1000	111
R_3	1000	0.1	9.9	100	10
R_4	400	0.25	3	100	33
R_5	100	0.1	30	10	0.3
R_{35}			$\text{Cost}(R_3) + \texttt{Est}(R_3)\text{Cost}(R_5)$ 1000	$\texttt{Est}(R_3)\texttt{Est}(R_5)$ 309.9	 3.223620523

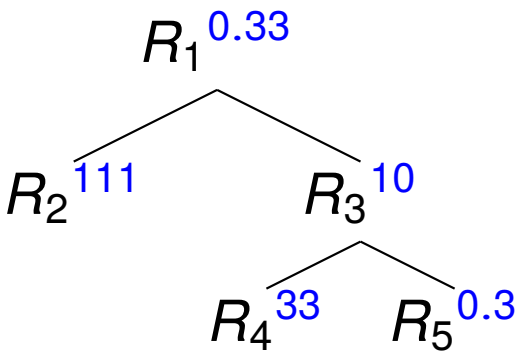


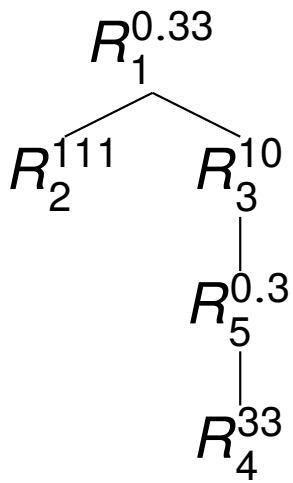


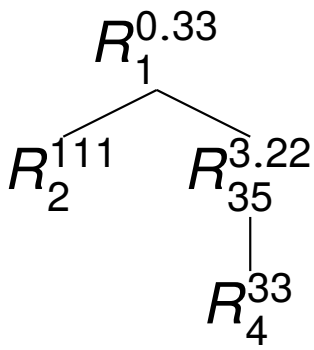


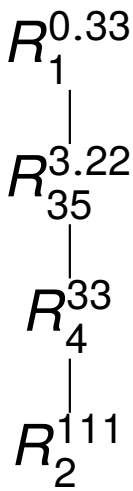
Example: Adapted from [Krishnamurthy et al., 1986]

	$\lvert R_i \rvert$	θ_i	$\text{Cost}(R_i) = g_i(\lvert R_i \rvert)$	$\texttt{Est}(R_i) = \theta_i \lvert R_i \rvert$	$\text{Rank}(R_i) = \frac{\texttt{Est}(R_i)-1}{\text{Cost}(R_i)}$
R_1	100	1	300	100	0.33
R_2	2000	0.5	9	1000	111
R_3	1000	0.1	9.9	100	10
R_4	400	0.25	3	100	33
R_5	100	0.1	30	10	0.3
R_{35}			$\text{Cost}(R_3) + \texttt{Est}(R_3)\text{Cost}(R_5)$ 1000	$\texttt{Est}(R_3)\texttt{Est}(R_5)$ 309.9	 3.223620523



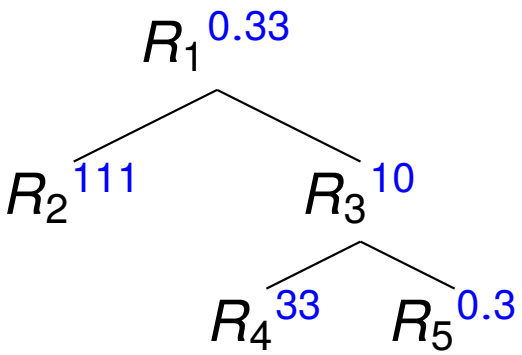


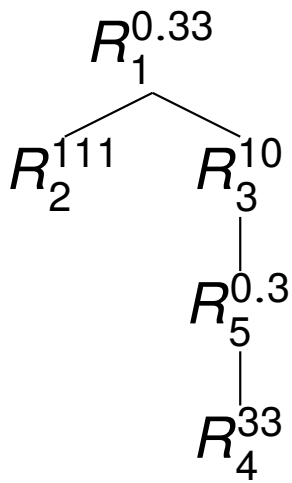


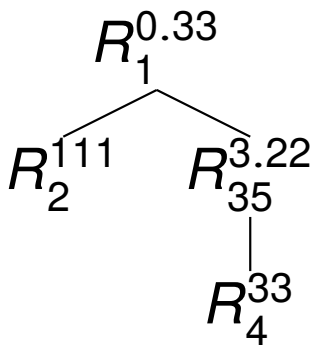


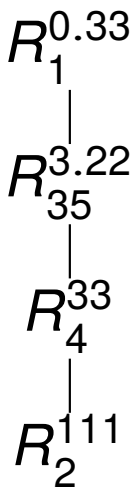
Example: Adapted from [Krishnamurthy et al., 1986]

	$\|R_i\|$	θ_i	$\text{Cost}(R_i) = g_i(\|R_i\|)$	$\texttt{Est}(R_i) = \theta_i \|R_i\|$	$\text{Rank}(R_i) = \frac{\texttt{Est}(R_i)-1}{\text{Cost}(R_i)}$
R_1	100	1	300	100	0.33
R_2	2000	0.5	9	1000	111
R_3	1000	0.1	9.9	100	10
R_4	400	0.25	3	100	33
R_5	100	0.1	30	10	0.3
R_{35}			$\text{Cost}(R_3) + \texttt{Est}(R_3)\text{Cost}(R_5)$ 1000	$\texttt{Est}(R_3)\texttt{Est}(R_5)$ 309.9	 3.223620523









Optimal plan so far: $R_1 \bowtie R_3 \bowtie R_5 \bowtie R_4 \bowtie R_2$.

Next, repeat with root= R_2, then root= R_3, etc. Return the cheapest plan.

Discussion

- Two specialized algorithms: Greedy and IKKBZ
- IKKBZ finds optimal plan in polynomial time, but only under the ASI assumption on Cost.
- Can still use IKKBZ as a heuristics, setting $\text{Rank}(S) = \frac{\texttt{Est}(S)-1}{\text{Cost}(S)}$.

Fegaras, L. (1998).

A new heuristic for optimizing large queries.
In Quirchmayr, G., Schweighofer, E., and Bench-Capon, T. J. M., editors, Database and Expert Systems Applications, 9th International Conference, DEXA '98, Vienna, Austria, August 24-28, 1998, Proceedings, volume 1460 of Lecture Notes in Computer Science, pages 726–735. Springer.

Ibaraki, T. and Kameda, T. (1984).

On the optimal nesting order for computing n-relational joins.
ACM Trans. Database Syst., 9(3):482–502.

Krishnamurthy, R., Boral, H., and Zaniolo, C. (1986).

Optimization of nonrecursive queries.
In Chu, W. W., Gardarin, G., Ohsuga, S., and Kambayashi, Y., editors, VLDB'86 Twelfth International Conference on Very Large Data Bases, August 25-28, 1986, Kyoto, Japan, Proceedings, pages 128–137. Morgan Kaufmann.

Monma, C. L. and Sidney, J. B. (1979).

Sequencing with series-parallel precedence constraints.
Math. Oper. Res., 4(3):215–224.

Neumann, T. and Radke, B. (2018).

Adaptive optimization of very large join queries.
In Das, G., Jermaine, C. M., and Bernstein, P. A., editors, Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018, pages 677–692. ACM.