CSE599d: Advanced Query Processing

Lecture 6: QO by Dynamic Programming – Wrapup

Dan Suciu

University of Washington

The Full Picture

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

- SystemR: introduced DP; Left Linear Plans
- 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: $O(3^n)$
- Specialized Algorithms: Greedy, IKKBZ
- Very Large Joins.

Recap: Optimization based on Connected Component Pairs

Algorithm 1: 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$[S_1 \cup S_2] := (P_1 \circ P_2, c)$;

Runtime: $O(3^n)$.

Star query is the worst: $O(3^n)$
Chain query is the best: $O(n^3)$.

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 must handle large joins: specialized algorithms.

Greedy Algorithm

Computing a Plan Greedily

[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 Cost$(P_i \bowtie P_j)$ is minimal.
- Replace P_i, P_j with $P_i \bowtie P_j$. Repeat.
- Stop when there is a single plan, with all relations.

Computing a Plan Greedily

Function Greedy($R_1, R_2, \ldots, R_n$)

Plans := $\{R_1, R_2, \ldots, R_n\}$;

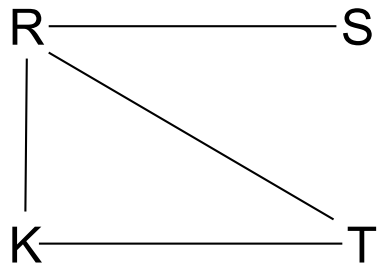
while $|\text{Plans}| > 1$ **do**

  $(L, R) := \text{argmin}_{L,R \in \text{Plans}, L, R \text{ can join}} \text{Cost}(L \bowtie R)$;

  Plans := Plans $- \{L, R\} \cup \{L \bowtie R\}$;

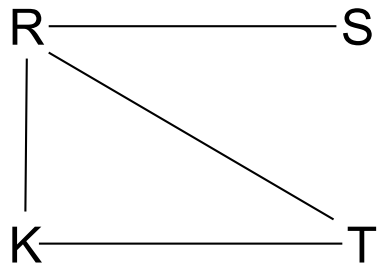
Example

- Step 1: Plans $= \{R, S, T, K\}$:
  $R \bowtie S, R \bowtie T, R \bowtie K, T \bowtie K$



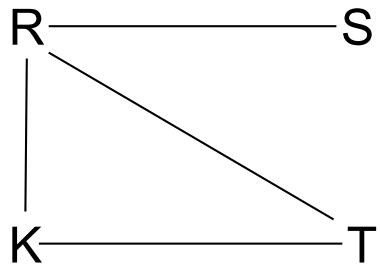
Example

- Step 1: Plans $= \{R, S, T, K\}$:
  $R \bowtie S, R \bowtie T, R \bowtie K, T \bowtie K$



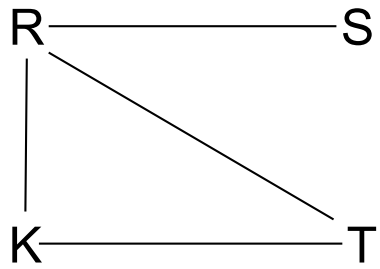
Example

- Step 1: Plans = $\{R, S, T, K\}$:
  $R \bowtie S, R \bowtie T, R \bowtie K, T \bowtie K$
- Step 2: Plans = $\{R, S, TK\}$:
  $R \bowtie S, R \bowtie TK$



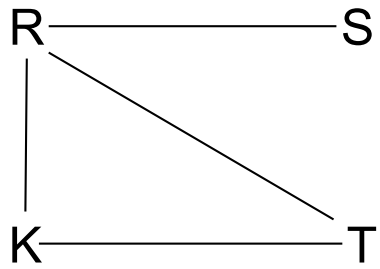
Example

- Step 1: Plans = $\{R, S, T, K\}$:
  $R \bowtie S, R \bowtie T, R \bowtie K, T \bowtie K$
- Step 2: Plans = $\{R, S, TK\}$:
  $R \bowtie S, R \bowtie TK$



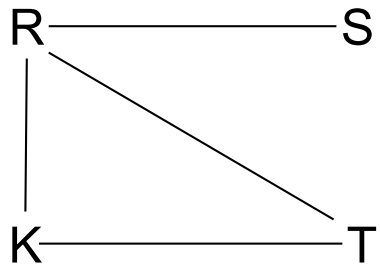
Example

- Step 1: Plans = $\{R, S, T, K\}$:
  $R \bowtie S, R \bowtie T, R \bowtie K, T \bowtie K$
- Step 2: Plans = $\{R, S, TK\}$:
  $R \bowtie S, R \bowtie TK$
- Step3: Plans = $\{S, RTK\}$:
  $S \bowtie RTK$

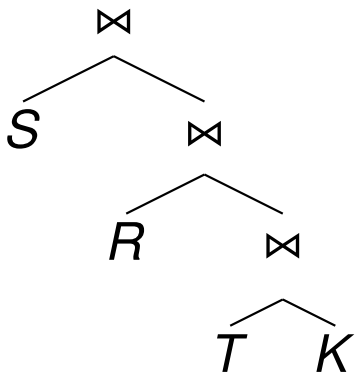


Example

- Step 1: Plans = $\{R, S, T, K\}$:
  $R \bowtie S, R \bowtie T, R \bowtie K, T \bowtie K$
- Step 2: Plans = $\{R, S, TK\}$:
  $R \bowtie S, R \bowtie TK$
- Step3: Plans = $\{S, RTK\}$:
  $S \bowtie RTK$



Final plan:



QO via Shortest Path [Haffner and Dittrich, 2023]

Given a weighted graph $G = (V, E)$, find the shortest path from s to t.

- Dijkstra's algorithm: Single-Source Shortest Path (SSSP) in time $O(|E| + |V| \log |V|)$.
- A^*: a search algorithm where G is given implicitly and has exponential size.

QO via Shortest Path [Haffner and Dittrich, 2023]

Join graph $Q = (\boldsymbol{R}, E)$, where $\boldsymbol{R} = \{R_1, \ldots, R_n\}$, E = all joins.

Define the search graph $G = (CC, J)$, where (informally!):

CC = set of partitions into connected components

$\boldsymbol{R} = C_1 \cup \cdots \cup C_k$

J = pairs of partitions that can represent one join in E.

The weight of an edge in J is the cost of the join.

QO via Shortest Path [Haffner and Dittrich, 2023]

Join graph $Q = (\boldsymbol{R}, E)$, where $\boldsymbol{R} = \{R_1, \ldots, R_n\}$, E = all joins.

Define the search graph $G = (CC, J)$, where (informally!):

CC = set of partitions into connected components

$\boldsymbol{R} = C_1 \cup \cdots \cup C_k$

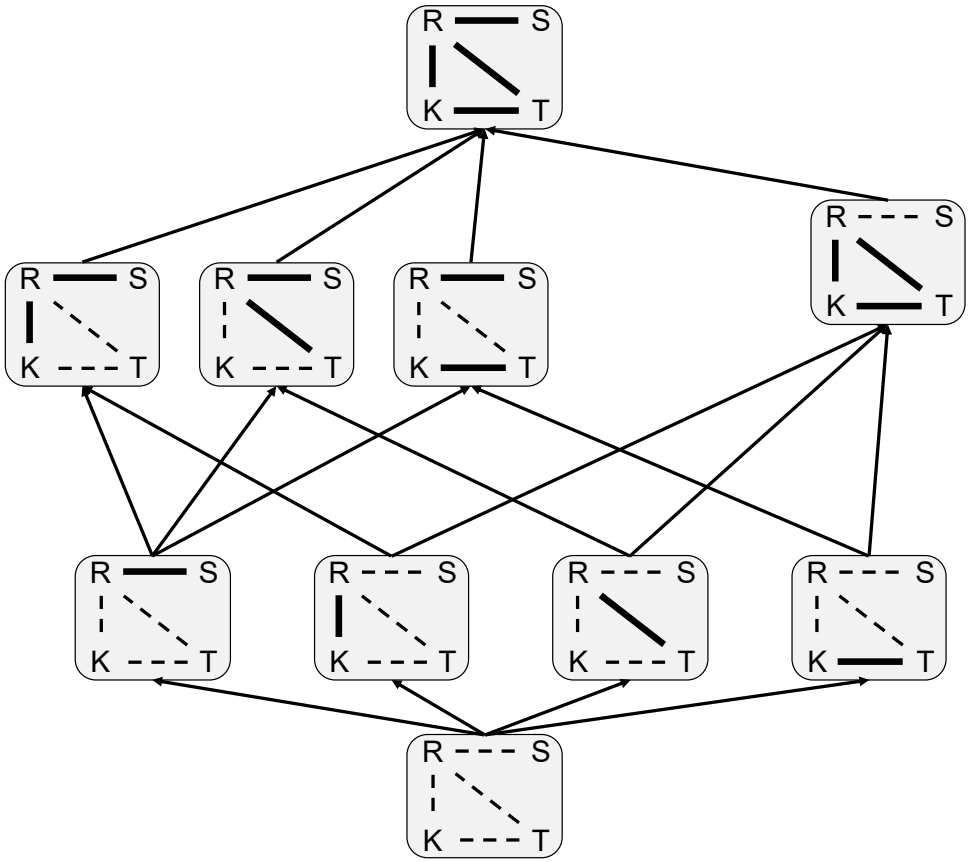
J = pairs of partitions that can represent one join in E.

The weight of an edge in J is the cost of the join.

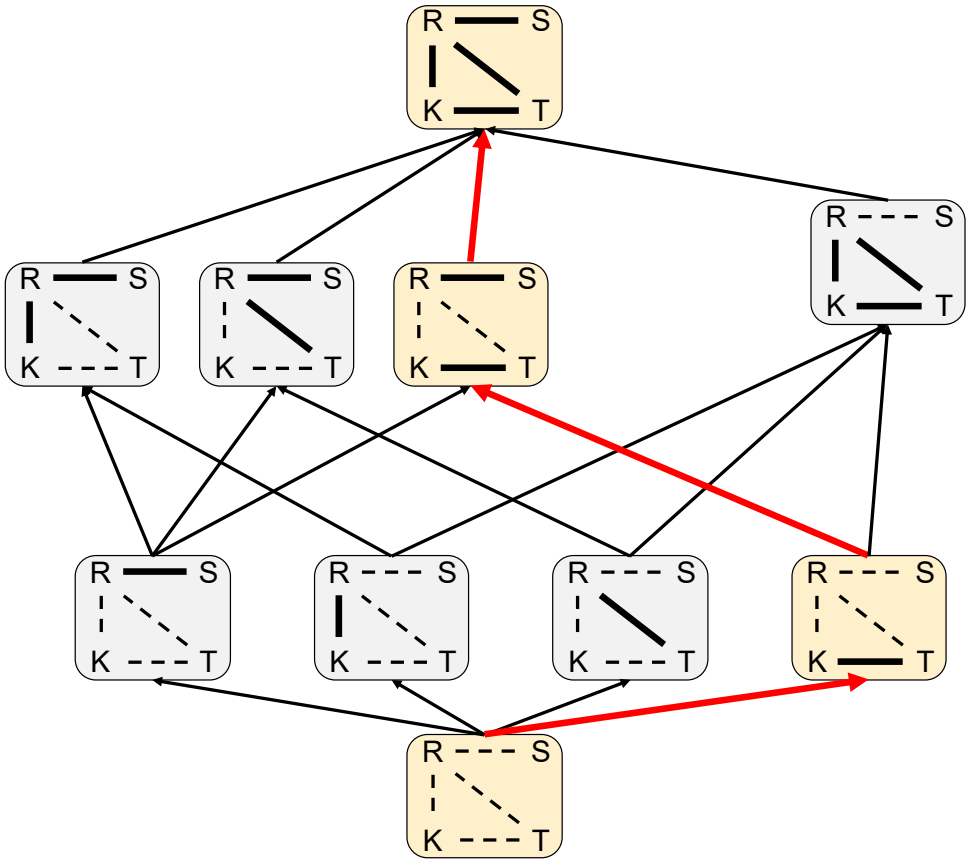
The optimal bushy plan w/o cross products is the shortest path:

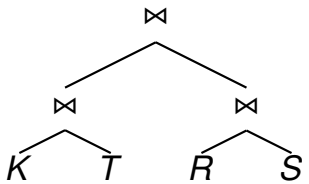
$$\{\{R_1\}, \{R_2\}, \ldots, \{R_n\}\} \rightarrow^* \{\{R_1, \ldots, R_n\}\}$$

QO via Shortest Path [Haffner and Dittrich, 2023]

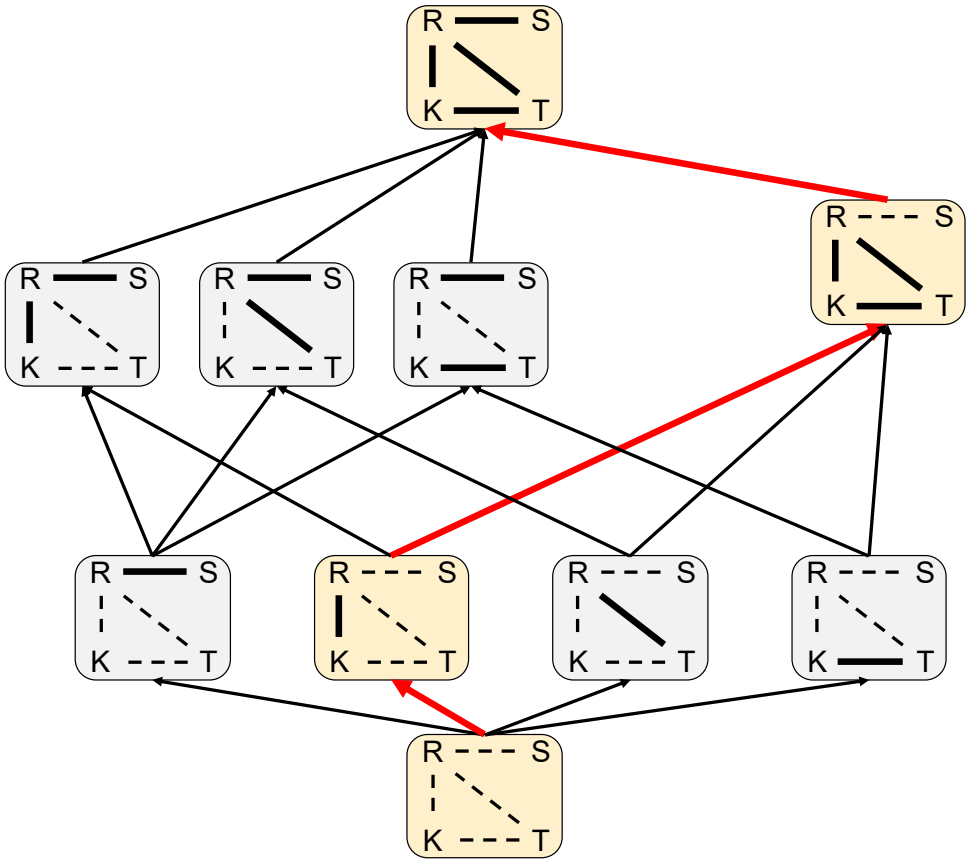


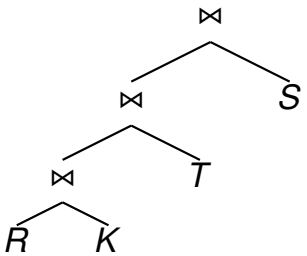
QO via Shortest Path [Haffner and Dittrich, 2023]





QO via Shortest Path [Haffner and Dittrich, 2023]





Discussion

- Greedy: very simple, suboptimal plan

- Graph search: natural extension.
  Cannot construct the full graph (exponential!), but can use A^* or some beam search.

The IKKBZ Algorithm

Overview of IKKBZ

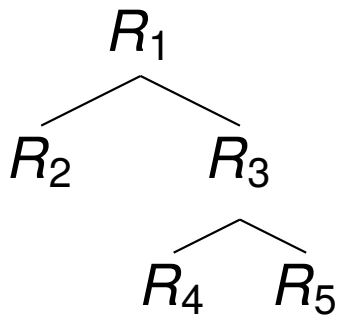
Optimal left linear plan w/o cross products, time $O(n^2)$. Assumes ASI property.

- 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.

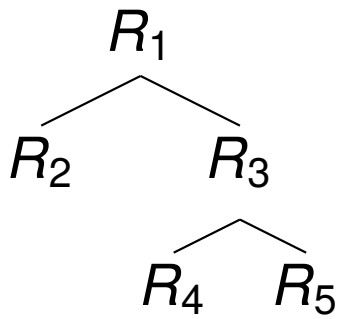
R_1 joins with R_2 and R_3.
R_1 does not join with R_4 or R_5.



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.

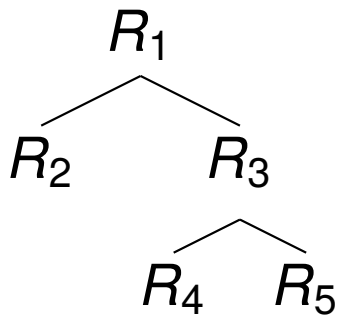


Goal: find optimal left linear plan w/o cross products

Problem Setting

Input: acyclic query graph.

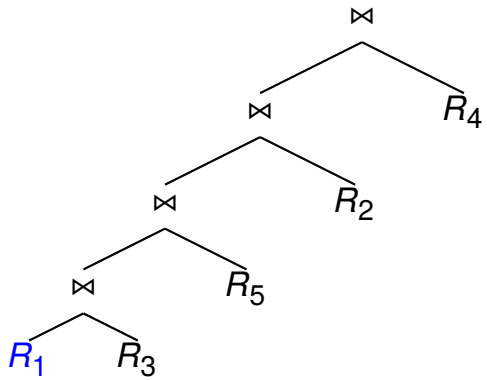
R_1 joins with R_2 and R_3.
R_1 does not join with R_4 or R_5.



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 $p(i) \stackrel{\text{def}}{=} \text{parent}(i)$.

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

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

Assumptions on the Cost Function

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

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

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

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

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

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

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. $\text{Cost}(R_4 R_5 R_6 R_7 R_8) = \text{Cost}(R_4 R_5 R_6) + \texttt{Est}(R_4 R_5 R_6) \cdot \text{Cost}(R_7 R_8)$

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 $\boxed{\text{Rank}(U) \leq \text{Rank}(V)}$

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

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

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 $\boxed{\text{Rank}(U) \leq \text{Rank}(V)}$

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

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

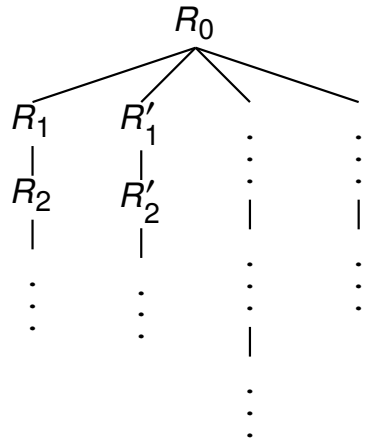
The IKKBZ Algorithm

For each node R, find optimal topological order for tree w/ root R:

- Choose node R_0 all of whose children are chains.
- (Assuming only two children.) By induction:

  $\text{Rank}(R_1) \leq \text{Rank}(R_2) \leq \cdots$

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



The IKKBZ Algorithm

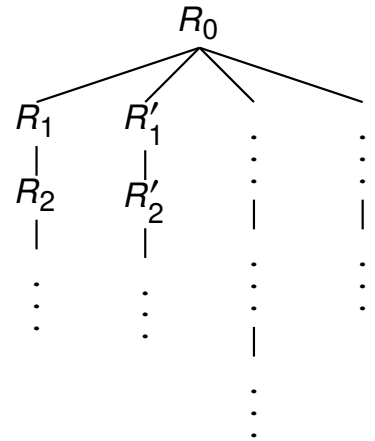
For each node R, find optimal topological order for tree w/ root R:

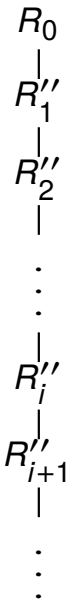
- Choose node R_0 all of whose children are chains.
- (Assuming only two children.) By induction:

  $\text{Rank}(R_1) \leq \text{Rank}(R_2) \leq \cdots$

  $\text{Rank}(R_1') \leq \text{Rank}(R_2') \leq \cdots$
- Merge the two chains to obtain:

  $\text{Rank}(R_0) \overset{??}{\leqslant} \text{Rank}(R_1'') \leq \text{Rank}(R_2'') \leq \cdots$

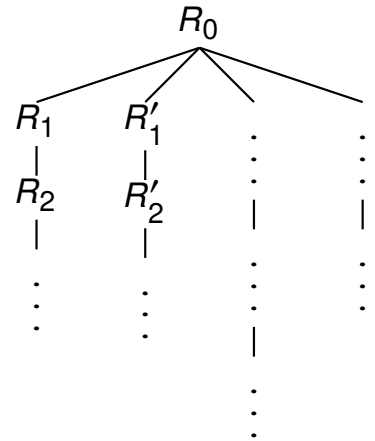


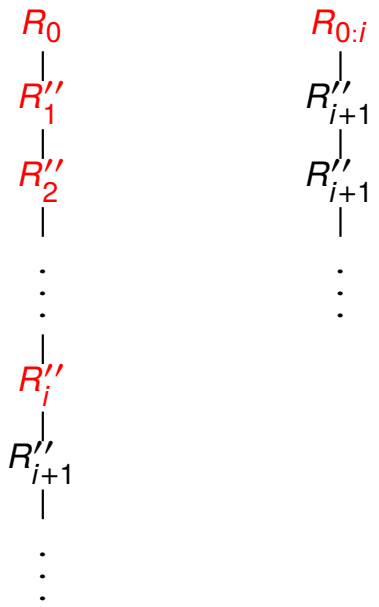


The IKKBZ Algorithm

For each node R, find optimal topological order for tree w/ root R:

- Choose node R_0 all of whose children are chains.
- (Assuming only two 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:
  $\text{Rank}(R_0) \overset{??}{\leqslant} \text{Rank}(R''_1) \le \text{Rank}(R''_2) \le \cdots$
- Cannot move R_0. Instead:
  If $\text{Rank}(R_0) > \text{Rank}(R_1)$: $\boxed{R_{0:1} := R_0 R''_1}$
  If $\text{Rank}(R_{0:1}) > \text{Rank}(R_2)$: $\boxed{R_{0:2} := R_0 R''_1 R''_2}$
  Until $\text{Rank}(R_{0:i}) \le \text{Rank}(R''_{i+1})$.

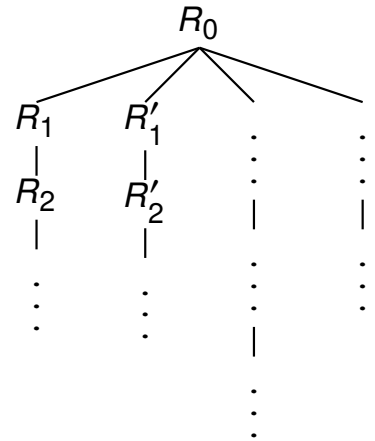


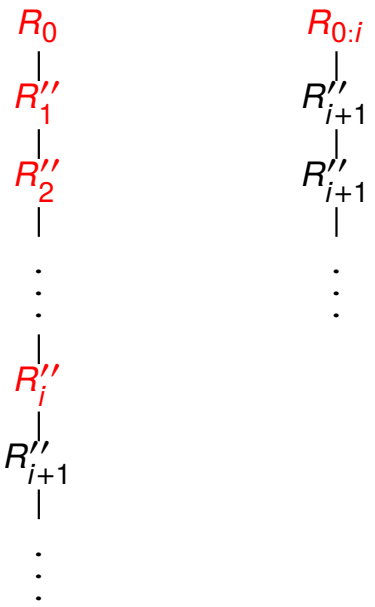


The IKKBZ Algorithm

For each node R, find optimal topological order for tree w/ root R:

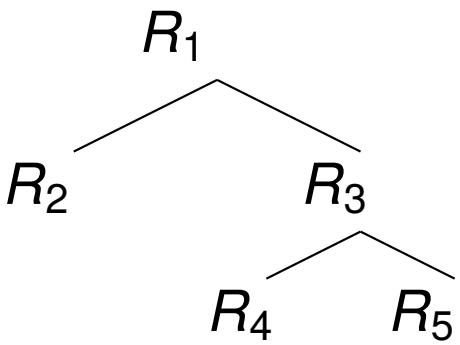
- Choose node R_0 all of whose children are chains.
- (Assuming only two children.) By induction:
  $\text{Rank}(R_1) \le \text{Rank}(R_2) \le \cdots$
  $\text{Rank}(R'_1) \le \text{Rank}(R'_2) \le \cdots$
- Merge the two chains to obtain:
  $\text{Rank}(R_0) \overset{??}{\leqslant} \text{Rank}(R''_1) \le \text{Rank}(R''_2) \le \cdots$
- Cannot move R_0. Instead:
  If $\text{Rank}(R_0) > \text{Rank}(R_1)$: $\boxed{R_{0:1} := R_0 R''_1}$
  If $\text{Rank}(R_{0:1}) > \text{Rank}(R_2)$: $\boxed{R_{0:2} := R_0 R''_1 R''_2}$
  Until $\text{Rank}(R_{0:i}) \le \text{Rank}(R''_{i+1})$.
- Repeat until entire rooted tree becomes a single chain.





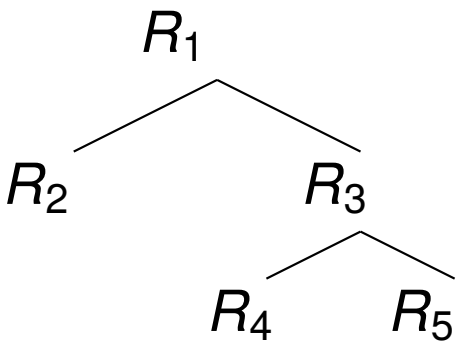
Example: Adapted from [Krishnamurthy et al., 1986]

	$\lvert R_i \rvert$	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



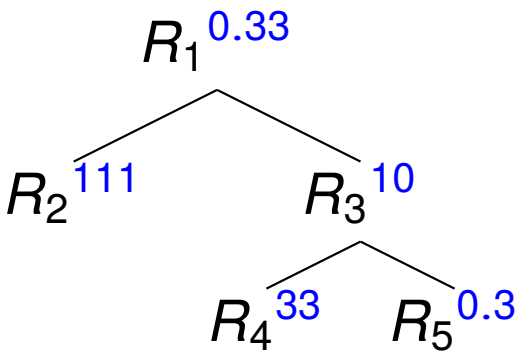
Example: Adapted from [Krishnamurthy et al., 1986]

	$\|R_i\|$	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



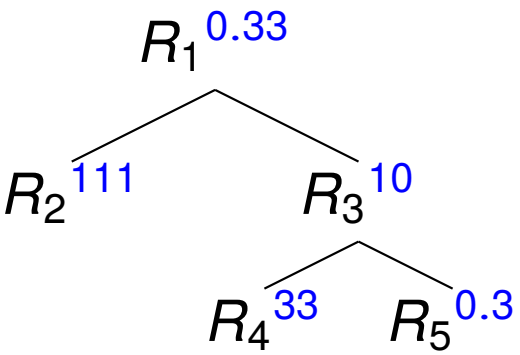
Example: Adapted from [Krishnamurthy et al., 1986]

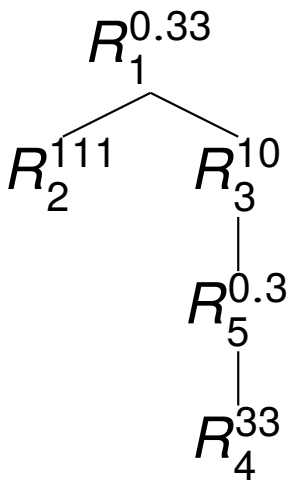
	$\lvert R_i \rvert$	θ_i	$\text{Cost}(R_i) = g_i(\lvert R_i \rvert)$	$\texttt{Est}(R_i) = \theta_i \lvert R_i \rvert$	$\text{Rank}(R_i) = \frac{\texttt{Est}(R_i)-1}{\text{Cost}(R_i)}$
R_1	100	1	300	100	0.33
R_2	2000	0.5	9	1000	111
R_3	1000	0.1	9.9	100	10
R_4	400	0.25	3	100	33
R_5	100	0.1	30	10	0.3



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

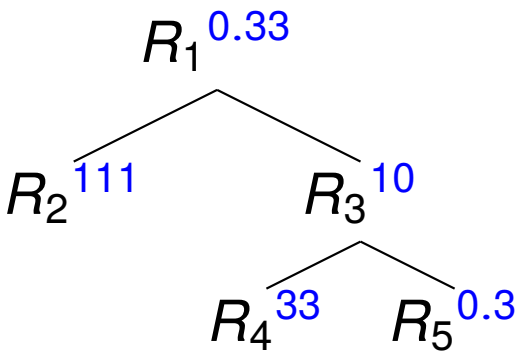
	$\lvert R_i \rvert$	θ_i	$\text{Cost}(R_i) = g_i(\lvert R_i \rvert)$	$\texttt{Est}(R_i) = \theta_i \lvert R_i \rvert$	$\text{Rank}(R_i) = \frac{\texttt{Est}(R_i)-1}{\text{Cost}(R_i)}$
R_1	100	1	300	100	0.33
R_2	2000	0.5	9	1000	111
R_3	1000	0.1	9.9	100	10
R_4	400	0.25	3	100	33
R_5	100	0.1	30	10	0.3

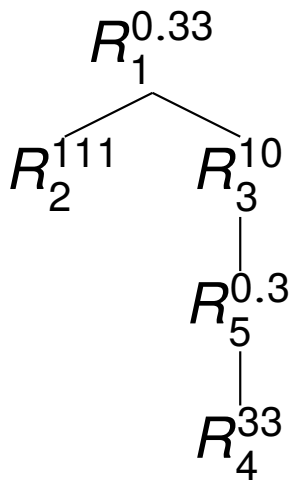


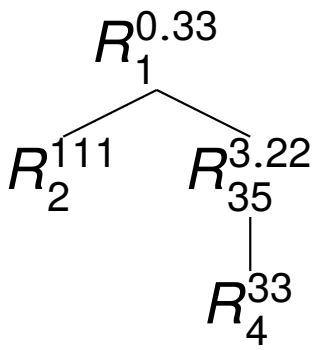


Example: Adapted from [Krishnamurthy et al., 1986]

	$\lvert R_i \rvert$	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

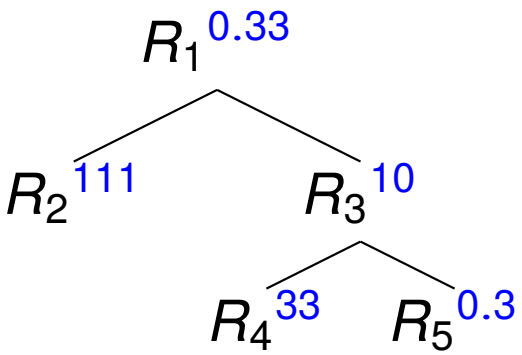


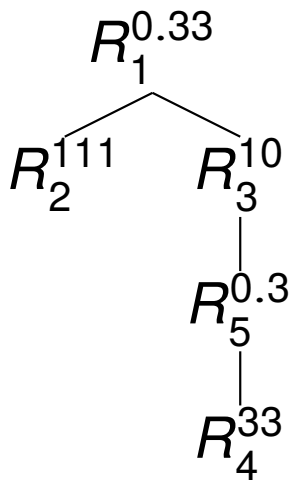


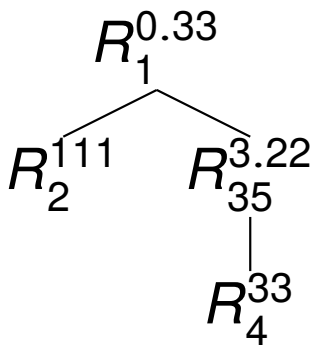


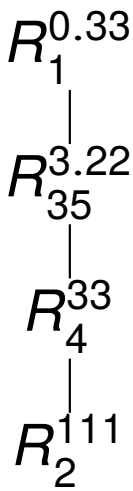
Example: Adapted from [Krishnamurthy et al., 1986]

	$\lvert R_i \rvert$	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



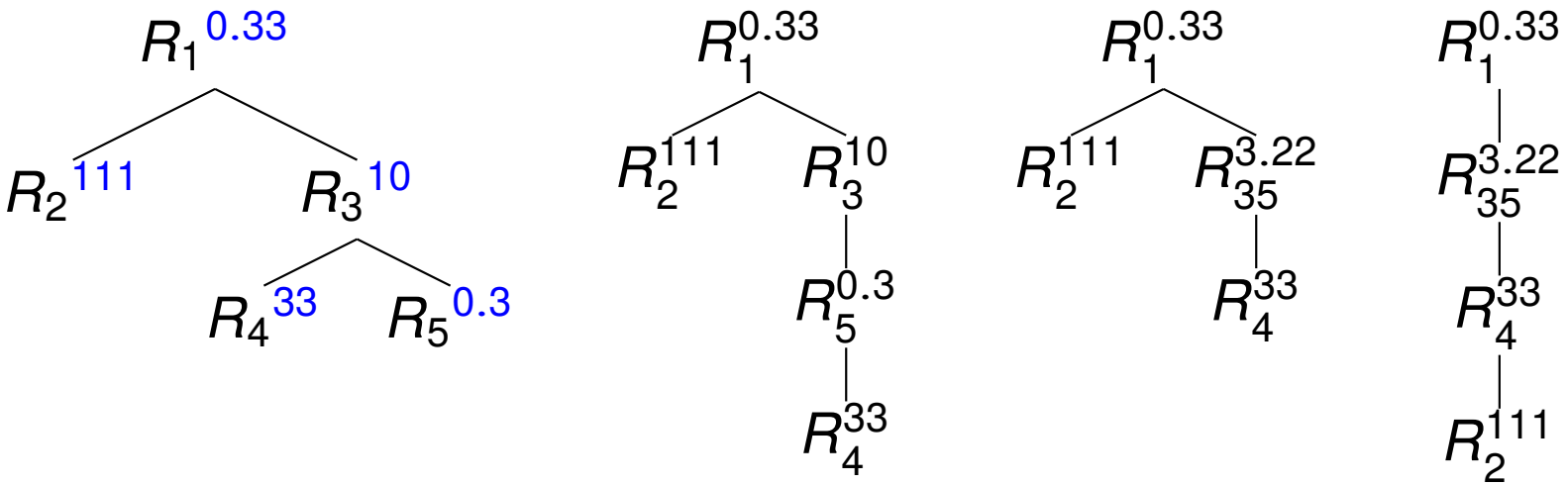






Example: Adapted from [Krishnamurthy et al., 1986]

	$\lvert R_i \rvert$	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



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.

Optimality Proof: Overview

- None of [Krishnamurthy et al., 1986, Ibaraki and Kameda, 1984, Neumann and Radke, 2018] gives the correctness proof.
- A proof will give us better understanding.
- One key step is to explain the need to combine $R_0, R_1, \ldots, R_i$ into $R_{0:i}$: this is the Key Lemma
- Another subtlety is that we want to prove that the output is globally optimal, not locally optimal.

Key Lemma

Assume $\text{Rank}(R_0) > \text{Rank}(R_1), \ldots \text{Rank}(R_{0:i-1}) > \text{Rank}(R_i)$. Then

$$\boxed{\text{Cost}(R_0 \cdots R_{j-1} S R_j \cdots R_i) > \min(\text{Cost}(S R_0 \cdots R_i), \text{Cost}(R_0 \cdots R_i S))}$$

"Inserting S between $R_0 \cdots R_i$ is suboptimal"

Key Lemma

Assume $\text{Rank}(R_0) > \text{Rank}(R_1), \ldots \text{Rank}(R_{0:i-1}) > \text{Rank}(R_i)$. Then

$$\boxed{\text{Cost}(R_0 \cdots R_{j-1} S R_j \cdots R_i) > \min(\text{Cost}(S R_0 \cdots R_i), \text{Cost}(R_0 \cdots R_i S))}$$

"Inserting S between $R_0 \cdots R_i$ is suboptimal"

Proof

$\text{Rank}(R_0) > \text{Rank}(R_1)$ implies $R_0 S R_1$ is suboptimal:

Key Lemma

Assume $\text{Rank}(R_0) > \text{Rank}(R_1), \ldots \text{Rank}(R_{0:i-1}) > \text{Rank}(R_i)$. Then

$$\boxed{\text{Cost}(R_0 \cdots R_{j-1} S R_j \cdots R_i) > \min(\text{Cost}(S R_0 \cdots R_i), \text{Cost}(R_0 \cdots R_i S))}$$

"Inserting S between $R_0 \cdots R_i$ is suboptimal"

Proof

$\text{Rank}(R_0) > \text{Rank}(R_1)$ implies $R_0 S R_1$ is suboptimal:

- If $\text{Rank}(S) < \text{Rank}(R_0)$ then $\text{Cost}(R_0 S R_1) >$???????.

Key Lemma

Assume $\text{Rank}(R_0) > \text{Rank}(R_1), \ldots \text{Rank}(R_{0:i-1}) > \text{Rank}(R_i)$. Then

$$\boxed{\text{Cost}(R_0 \cdots R_{j-1} S R_j \cdots R_i) > \min(\text{Cost}(S R_0 \cdots R_i), \text{Cost}(R_0 \cdots R_i S))}$$

"Inserting S between $R_0 \cdots R_i$ is suboptimal"

Proof

$\text{Rank}(R_0) > \text{Rank}(R_1)$ implies $R_0 S R_1$ is suboptimal:

- If $\text{Rank}(S) < \text{Rank}(R_0)$ then $\text{Cost}(R_0 S R_1) > \text{Cost}(S R_0 R_1)$.

Key Lemma

Assume $\text{Rank}(R_0) > \text{Rank}(R_1), \ldots \text{Rank}(R_{0:i-1}) > \text{Rank}(R_i)$. Then

$$\boxed{\text{Cost}(R_0 \cdots R_{j-1} S R_j \cdots R_i) > \min(\text{Cost}(S R_0 \cdots R_i), \text{Cost}(R_0 \cdots R_i S))}$$

"Inserting S between $R_0 \cdots R_i$ is suboptimal"

Proof

$\text{Rank}(R_0) > \text{Rank}(R_1)$ implies $R_0 S R_1$ is suboptimal:

- If $\text{Rank}(S) < \text{Rank}(R_0)$ then $\text{Cost}(R_0 S R_1) > \text{Cost}(S R_0 R_1)$.
- If $\text{Rank}(S) > \text{Rank}(R_1)$ then $\text{Cost}(R_0 S R_1) >$???????.

Key Lemma

Assume $\text{Rank}(R_0) > \text{Rank}(R_1), \ldots \text{Rank}(R_{0:i-1}) > \text{Rank}(R_i)$. Then

$$\boxed{\text{Cost}(R_0 \cdots R_{j-1} S R_j \cdots R_i) > \min(\text{Cost}(S R_0 \cdots R_i), \text{Cost}(R_0 \cdots R_i S))}$$

"Inserting S between $R_0 \cdots R_i$ is suboptimal"

Proof

$\text{Rank}(R_0) > \text{Rank}(R_1)$ implies $R_0 S R_1$ is suboptimal:

- If $\text{Rank}(S) < \text{Rank}(R_0)$ then $\text{Cost}(R_0 S R_1) > \text{Cost}(S R_0 R_1)$.
- If $\text{Rank}(S) > \text{Rank}(R_1)$ then $\text{Cost}(R_0 S R_1) > \text{Cost}(R_0 R_1 S)$.

Key Lemma

Assume $\text{Rank}(R_0) > \text{Rank}(R_1), \ldots \text{Rank}(R_{0:i-1}) > \text{Rank}(R_i)$. Then

$$\boxed{\text{Cost}(R_0 \cdots R_{j-1} S R_j \cdots R_i) > \min(\text{Cost}(S R_0 \cdots R_i), \text{Cost}(R_0 \cdots R_i S))}$$

"Inserting S between $R_0 \cdots R_i$ is suboptimal"

Proof

$\text{Rank}(R_0) > \text{Rank}(R_1)$ implies $R_0 S R_1$ is suboptimal:

- If $\text{Rank}(S) < \text{Rank}(R_0)$ then $\text{Cost}(R_0 S R_1) > \text{Cost}(S R_0 R_1)$.
- If $\text{Rank}(S) > \text{Rank}(R_1)$ then $\text{Cost}(R_0 S R_1) > \text{Cost}(R_0 R_1 S)$.

$\text{Rank}(R_{0:1}) > \text{Rank}(R_2)$ implies $R_{0:1} S R_1$ is suboptimal.

$\text{Rank}(R_{0:2}) > \text{Rank}(R_3)$ implies $R_{0:2} S R_3$ is suboptimal. Etc.

Key Lemma

Assume $\text{Rank}(R_0) > \text{Rank}(R_1), \ldots \text{Rank}(R_{0:i-1}) > \text{Rank}(R_i)$. Then

$$\boxed{\text{Cost}(R_0 \cdots R_{j-1} S R_j \cdots R_i) > \min(\text{Cost}(S R_0 \cdots R_i), \text{Cost}(R_0 \cdots R_i S))}$$

"Inserting S between $R_0 \cdots R_i$ is suboptimal"

Proof

$\text{Rank}(R_0) > \text{Rank}(R_1)$ implies $R_0 S R_1$ is suboptimal:

- If $\text{Rank}(S) < \text{Rank}(R_0)$ then $\text{Cost}(R_0 S R_1) > \text{Cost}(S R_0 R_1)$.
- If $\text{Rank}(S) > \text{Rank}(R_1)$ then $\text{Cost}(R_0 S R_1) > \text{Cost}(R_0 R_1 S)$.

$\text{Rank}(R_{0:1}) > \text{Rank}(R_2)$ implies $R_{0:1} S R_1$ is suboptimal.

$\text{Rank}(R_{0:2}) > \text{Rank}(R_3)$ implies $R_{0:2} S R_3$ is suboptimal. Etc.

Justifies combining $R_0 \cdots R_i$ into $R_{0:i}$ for the rest of algorithm

Optimality of the IKKBZ Algorithm: Overview

Let $R_1 R_2 \ldots R_n$ be the sequence returned by IKKBZ.

Does it suffices to prove $\text{Rank}(R_1) \leq \text{Rank}(R_2) \leq \cdots$?

[1] If we also prove $\text{Rank}(R_1 R_2 R_3) \leq \text{Rank}(R_4 R_5)$, etc.

Optimality of the IKKBZ Algorithm: Overview

Let $R_1 R_2 \ldots R_n$ be the sequence returned by IKKBZ.

Does it suffices to prove $\text{Rank}(R_1) \leq \text{Rank}(R_2) \leq \cdots$?

No! That only proves local optimality[1]

[1] If we also prove $\text{Rank}(R_1 R_2 R_3) \leq \text{Rank}(R_4 R_5)$, etc.

Optimality of the IKKBZ Algorithm: Overview

Let $R_1 R_2 \ldots R_n$ be the sequence returned by IKKBZ.

Does it suffices to prove $\text{Rank}(R_1) \leq \text{Rank}(R_2) \leq \cdots$?

No! That only proves local optimality[1]

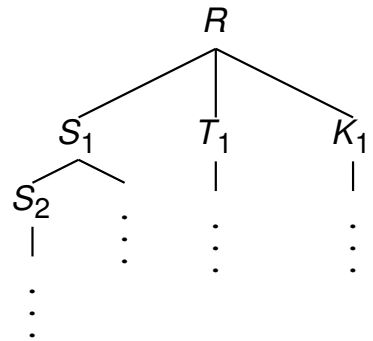
Instead we need to prove this:

If **Opt** is the optimal sequence, then IKKBZ returns **Opt**

[1] If we also prove $\text{Rank}(R_1 R_2 R_3) \leq \text{Rank}(R_4 R_5)$, etc.

Optimality of the IKKBZ Algorithm: Proof 1/2

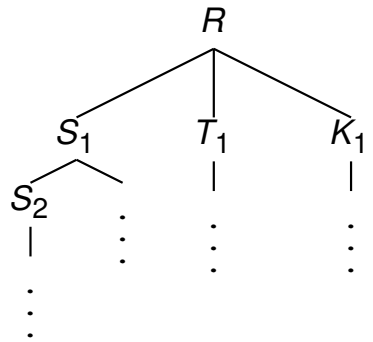
Theorem Fix R as the tree root, and let $\mathbf{Opt}_R$ be the optimal starting at this root. Then IKKBZ on root R returns $\mathbf{Opt}_R$.



Optimality of the IKKBZ Algorithm: Proof 1/2

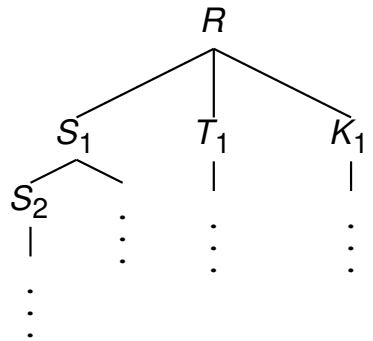
Theorem Fix R as the tree root, and let $\mathbf{Opt}_R$ be the optimal starting at this root. Then IKKBZ on root R returns $\mathbf{Opt}_R$.

Notice that $\mathbf{Opt}_R$ must start with R (why?).
IKKBZ also returns a sequence that starts with R.



Optimality of the IKKBZ Algorithm: Proof 1/2

Theorem Fix R as the tree root, and let $\mathbf{Opt}_R$ be the optimal starting at this root. Then IKKBZ on root R returns $\mathbf{Opt}_R$.



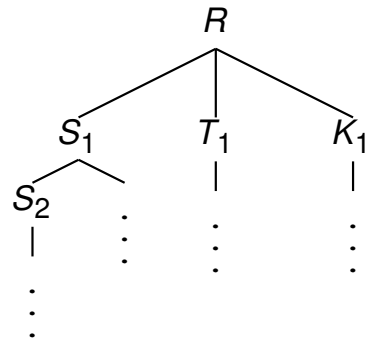
Notice that $\mathbf{Opt}_R$ must start with R (why?).
IKKBZ also returns a sequence that starts with R.

Let $\mathbf{Opt}_R \cap \mathcal{T}$ be the subsequence consisting of relations in subtree $\mathcal{T}$.

E.g. if $\mathbf{Opt}_R = RT_1S_1T_3K_1S_4S_2T_2\cdots$ then $\mathbf{Opt} \cap \text{Tree}(S_1) = S_1S_4S_2\cdots$

Optimality of the IKKBZ Algorithm: Proof 1/2

Theorem Fix R as the tree root, and let $\mathbf{Opt}_R$ be the optimal starting at this root. Then IKKBZ on root R returns $\mathbf{Opt}_R$.



Notice that $\mathbf{Opt}_R$ must start with R (why?).
IKKBZ also returns a sequence that starts with R.

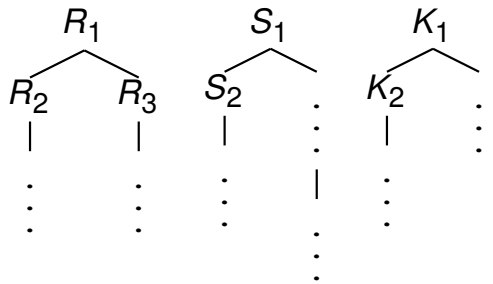
Let $\mathbf{Opt}_R \cap \mathcal{T}$ be the subsequence consisting of relations in subtree $\mathcal{T}$.

E.g. if $\mathbf{Opt}_R = RT_1S_1T_3K_1S_4S_2T_2\cdots$ then $\mathbf{Opt} \cap \text{Tree}(S_1) = S_1S_4S_2\cdots$

Subsequence Lemma: For every subtree $\mathcal{T}$ of the root R, $\mathbf{Opt}_R \cap \mathcal{T}$ is the optimal sequence $\mathbf{Opt}_{\mathcal{T}}$ of $\mathcal{T}$

Proof of the Subsequence Lemma

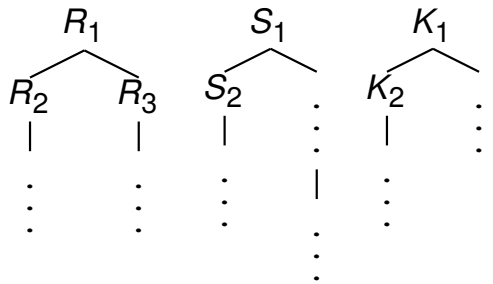
Forest $F = \{\mathcal{T}_1, \mathcal{T}_2, \ldots\}$ (set of ordered trees)



Claim Let $\mathbf{Opt}_F$ be optimal topological order of F.
Then, for all $\mathcal{T} \in F$, $\mathbf{Opt}_F \cap \mathcal{T}$ is optimal sequence $\mathbf{Opt}_\mathcal{T}$ of $\mathcal{T}$.

Proof of the Subsequence Lemma

Forest $F = \{\mathcal{T}_1, \mathcal{T}_2, \ldots\}$ (set of ordered trees)



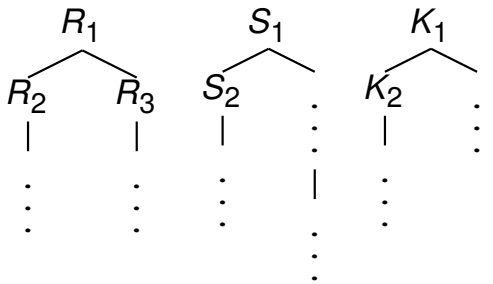
Claim Let $\mathbf{Opt}_F$ be optimal topological order of F.
Then, for all $\mathcal{T} \in F$, $\mathbf{Opt}_F \cap \mathcal{T}$ is optimal sequence $\mathbf{Opt}_\mathcal{T}$ of $\mathcal{T}$.

Proof by induction on the number of nodes in the forest.

Let the first node of $\mathbf{Opt}_F$ be S_1: $\boxed{\mathbf{Opt}_F = S_1\,\mathbf{Rest}}$

Proof of the Subsequence Lemma

Forest $F = \{\mathcal{T}_1, \mathcal{T}_2, \ldots\}$ (set of ordered trees)



Claim Let $\mathbf{Opt}_F$ be optimal topological order of F.
Then, for all $\mathcal{T} \in F$, $\mathbf{Opt}_F \cap \mathcal{T}$ is optimal sequence $\mathbf{Opt}_\mathcal{T}$ of $\mathcal{T}$.

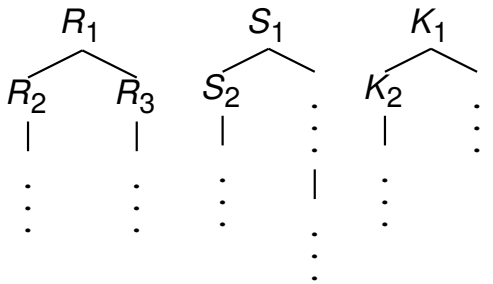
Proof by induction on the number of nodes in the forest.

Let the first node of $\mathbf{Opt}_F$ be S_1: $\boxed{\mathbf{Opt}_F = S_1\,\mathbf{Rest}}$

- S_1 is the root of some tree $\mathcal{T} \in F$ why?

Proof of the Subsequence Lemma

Forest $F = \{\mathcal{T}_1, \mathcal{T}_2, \ldots\}$ (set of ordered trees)



Claim Let $\mathbf{Opt}_F$ be optimal topological order of F.
Then, for all $\mathcal{T} \in F$, $\mathbf{Opt}_F \cap \mathcal{T}$ is optimal sequence $\mathbf{Opt}_\mathcal{T}$ of $\mathcal{T}$.

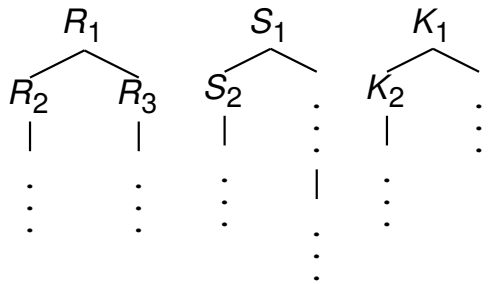
Proof by induction on the number of nodes in the forest.

Let the first node of $\mathbf{Opt}_F$ be S_1: $\boxed{\mathbf{Opt}_F = S_1\,\mathbf{Rest}}$

- S_1 is the root of some tree $\mathcal{T} \in F$ — why?
- and **Rest** is optimal topological order of $F - \{S_1\}$ — why?

Proof of the Subsequence Lemma

Forest $F = \{\mathcal{T}_1, \mathcal{T}_2, \ldots\}$ (set of ordered trees)



Claim Let $\mathbf{Opt}_F$ be optimal topological order of F.
Then, for all $\mathcal{T} \in F$, $\mathbf{Opt}_F \cap \mathcal{T}$ is optimal sequence $\mathbf{Opt}_\mathcal{T}$ of $\mathcal{T}$.

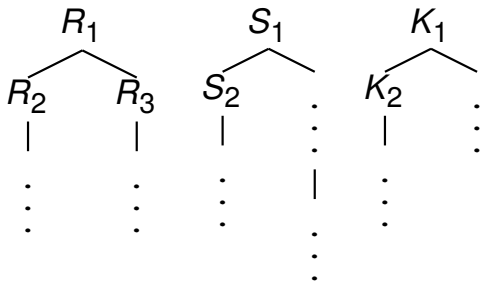
Proof by induction on the number of nodes in the forest.

Let the first node of $\mathbf{Opt}_F$ be S_1: $\boxed{\mathbf{Opt}_F = S_1\mathbf{Rest}}$

- S_1 is the root of some tree $\mathcal{T} \in F$ why?
- and **Rest** is optimal topological order of $F - \{S_1\}$ why?
  Because $\text{Cost}(\mathbf{Opt}_F) = \text{Cost}(S_1\mathbf{Rest}) = \text{Cost}(S_1) + \texttt{Est}(S_1)\text{Cost}(\mathbf{Rest})$
  Improving $\text{Cost}(\mathbf{Rest})$ would improve $\text{Cost}(\mathbf{Opt}_F)$.

Proof of the Subsequence Lemma

Forest $F = \{\mathcal{T}_1, \mathcal{T}_2, \ldots\}$ (set of ordered trees)



Claim Let $\mathbf{Opt}_F$ be optimal topological order of F.
Then, for all $\mathcal{T} \in F$, $\mathbf{Opt}_F \cap \mathcal{T}$ is optimal sequence $\mathbf{Opt}_\mathcal{T}$ of $\mathcal{T}$.

Proof by induction on the number of nodes in the forest.

Let the first node of $\mathbf{Opt}_F$ be S_1: $\boxed{\mathbf{Opt}_F = S_1\mathbf{Rest}}$

- S_1 is the root of some tree $\mathcal{T} \in F$ why?
- and **Rest** is optimal topological order of $F - \{S_1\}$ why?
  Because $\text{Cost}(\mathbf{Opt}_F) = \text{Cost}(S_1\mathbf{Rest}) = \text{Cost}(S_1) + \texttt{Est}(S_1)\text{Cost}(\mathbf{Rest})$
  Improving Cost(**Rest**) would improve $\text{Cost}(\mathbf{Opt}_F)$.

Lemma follows from induction hypothesis on $F - \{S_1\}$.

Optimality of the IKKBZ Algorithm: Proof 2/2

IKKBZ on root R returns $\mathbf{Opt}_R$.

Optimality of the IKKBZ Algorithm: Proof 2/2

IKKBZ on root R returns $\mathbf{Opt}_R$.

Assume for simplicity that R has two children: $\mathcal{T}_1, \mathcal{T}_2$

Optimality of the IKKBZ Algorithm: Proof 2/2

IKKBZ on root R returns $\mathbf{Opt}_R$.

Assume for simplicity that R has two children: $\mathcal{T}_1, \mathcal{T}_2$

By induction, IKKBZ computes their optimal order:

$$\mathbf{Opt}_{\mathcal{T}_1} = S_1 S_2 \cdots$$
$$\mathbf{Opt}_{\mathcal{T}_2} = T_1 T_2 \cdots$$

Optimality of the IKKBZ Algorithm: Proof 2/2

IKKBZ on root R returns $\mathbf{Opt}_R$.

Assume for simplicity that R has two children: $\mathcal{T}_1, \mathcal{T}_2$

By induction, IKKBZ computes their optimal order:

$$\mathbf{Opt}_{\mathcal{T}_1} = S_1 S_2 \cdots$$
$$\mathbf{Opt}_{\mathcal{T}_2} = T_1 T_2 \cdots$$

By the Subsequence Lemma:

$$\mathbf{Opt}_R \cap \mathcal{T}_1 = S_1 S_2 \cdots$$
$$\mathbf{Opt}_R \cap \mathcal{T}_2 = T_1 T_2 \cdots$$

Optimality of the IKKBZ Algorithm: Proof 2/2

IKKBZ on root R returns $\mathbf{Opt}_R$.

Assume for simplicity that R has two children: $\mathcal{T}_1, \mathcal{T}_2$

By induction, IKKBZ computes their optimal order:

$$\mathbf{Opt}_{\mathcal{T}_1} = S_1 S_2 \cdots$$
$$\mathbf{Opt}_{\mathcal{T}_2} = T_1 T_2 \cdots$$

By the Subsequence Lemma:

$$\mathbf{Opt}_R \cap \mathcal{T}_1 = S_1 S_2 \cdots$$
$$\mathbf{Opt}_R \cap \mathcal{T}_2 = T_1 T_2 \cdots$$

By the Key Lemma, if $S_{i:j}$ was combined by the algorithm, then $\mathbf{Opt}_R$ will not have any T_k between $S_i \cdots S_j$.

By ASI, $\mathbf{Opt}_R$ is obtained by merging their ranks.

Discussion

- IKKBZ finds optimal plan in polynomial time, but only under the ASI assumption on Cost.
- Can still use IKKBZ as a heuristics, setting $\text{Rank}(S) = \frac{\texttt{Est}(S)-1}{\text{Cost}(S)}$.
- Searches only for left-linear plans. This is useful for the linearization used to optimize very large joins: next.

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

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.

Medium Query

- 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

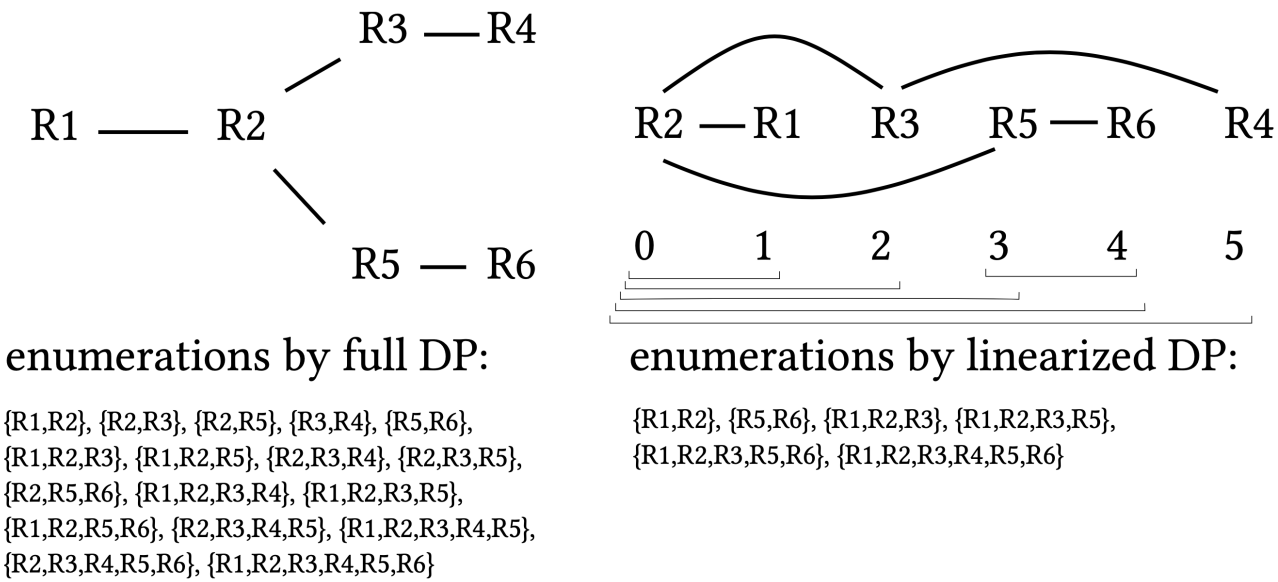


Figure 4: Example for Search Space Linearization

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

Use IKKBZ to find a good linearization order.

LinearizedDP: a simplified version of DPccp, runtime $O(n^3)$.

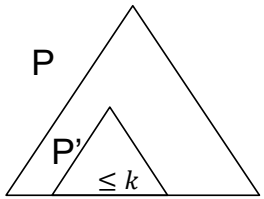
Discussion

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

Large Query

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

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

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.

Haffner, I. and Dittrich, J. (2023).

Efficiently computing join orders with heuristic search.
Proc. ACM Manag. Data, 1(1):73:1–73:26.

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.