CSEP 544: Lecture 09

Advanced Query Processing
Parallel Query Evaluation
Outline

• Optimal Sequential Algorithms

• Semijoin Reduction

• Optimal Parallel Algorithms
Semijoin: Review of Basics

\[
R \Join_{C} S = \Pi_{A_1, \ldots, A_n} (R \Join_{C} S)
\]

Where \(A_1, \ldots, A_n\) are the attributes in \(R\)

Formally, \(R \Join_{C} S\) means this: retain from \(R\) only those tuples that have some matching tuple in \(S\)
Duplicates in \(R\) are preserved
Duplicates in \(S\) don’t matter
Application

Q(x, y, z, u, v, w) = R(x, y), S(y, z), T(z, u), K(u, v), L(v, w)

Intermediate relations of size up to $m^5$...
... But final answer can be as small as 0 (or 1...)
Application

\[ Q(x,y,z,u,v,w) = R(x,y), S(y,z), T(z,u), K(u,v), L(v,w) \]

Intermediate relations of size up to \( m^5 \)…
... But final answer can be as small as 0 (or 1...)

Solution: semi-join reduction

/* Forwards: */
S := S \bowtie R
T := T \bowtie S
K := K \bowtie T
L := L \bowtie K

/* then backwards: */
K := K \bowtie L
T := T \bowtie K
S := S \bowtie T
R := R \bowtie S
Application

\[ Q(x,y,z,u,v,w) = R(x,y), S(y,z), T(z,u), K(u,v), L(v,w) \]

Intermediate relations of size up to \( m^5 \) …
… But final answer can be as small as 0 (or 1…)

Solution: semi-join reduction

/* Forwards: */
\[
\begin{align*}
S & := S \bowtie R \\
T & := T \bowtie S \\
K & := K \bowtie T \\
L & := L \bowtie K
\end{align*}
\]

/* then backwards: */
\[
\begin{align*}
K & := K \bowtie L \\
T & := T \bowtie K \\
S & := S \bowtie T \\
R & := R \bowtie S
\end{align*}
\]

Next, compute the query in time \( O(|Input| + |Output|) \)
Acyclic Queries

• A query is *acyclic* if its relations can be placed in a tree such that the set of nodes that contain any variable form a connected set

• Yannakakis’ algorithm (from the 80’s): any acyclic query can be computed in time: $O(|\text{Input}| + |\text{Output}|)$
Acyclic query:
\[ Q(x,y,z,u,v,w,m) = R(x,y), S(y,z,u), T(y,zw), K(z,v), L(v,m) \]

Cyclic query:
\[ Q(x,y,z) = R(x,y), S(y,z), T(z,x) \]
Yannakakis’ Algorithm

• Step 1: semi-join reduction
  – Two sweeps: up and down

• Step 2: use the tree as query query plan, compute bottom up

• Note: this works also for queries with projections and/or aggregates (in class...)
Acyclic query:
\[ Q(x,y,z,u,v,w,m) = R(x,y), S(y,z,u), T(y,zw), K(z,v), L(v,m) \]
Acyclic query:
\[ Q(x, y, z, u, v, w, m) = R(x, y), S(y, z, u), T(y, z, w), K(z, v), L(v, m) \]
Acyclic query:
\[ Q(x, y, z, u, v, w, m) = R(x, y), S(y, z, u), T(y, w), K(z, v), L(v, m) \]
Acyclic query:
\[ Q(x,y,z,u,v,w,m) = R(x,y), S(y,z,u), T(y,zw), K(z,v), L(v,m) \]

Query plan:
\[ \bowtie R(x,y) \bowtie S(y,z,u) \bowtie T(y,zw) \bowtie K(z,v) \bowtie L(v,m) \]

Question: change the query plan to return only variables \( x, w, m \)

\[ Q(x,w,m) = R(x,y), S(y,z,u), T(y,zw), K(z,v), L(v,m) \]
Cyclic Queries:

1. Acyclic queries: computable in time $O(|input| + |output|)$ [Yannakakis’81]

2. Arbitrary queries: computable in time $O(|input|^{\text{fractional-tree-width}} + |output|)$

The fractional tree-width:
- Tree nodes may contain multiple relations
- FTW is the largest AGM bound of any node
In class: how do we compute this query

Optimal edge cover = $\rho^*$
AGM bound = $m^{\rho^*}$
Generic-join algorithm, time = $O(m^{\rho^*})$

$O(n + |Output|)$

ftw = max($\rho^*$) = 2

$O(|Input|^2 + |Output|)$
Outline

• Optimal Sequential Algorithms

• Semijoin Reduction

• Optimal Parallel Algorithms
Parallel Models

• Shared-Nothing: $p$ servers
• MapReduce: $p$ reducers

• Notice: in MR we can choose $p$, but, as we shall see, it’s better if $p$ is the number of physical servers
Overview

Computes a full conjunctive query in one round, by partial replication.

• Replication appears in [Ganguli’92]
• **Shares Algorithm:** [Afrati&Ullman’10]
  – For MapReduce
• **HyperCube Algorithm** [Beame’13,’14]
  – Same as in Shares
  – But different optimization/analysis
The Triangle Query

- **Input**: three tables
  \[ R(X, Y), \ S(Y, Z), \ T(Z, X) \]
  \[ |R| = |S| = |T| = m \] tuples

- **Output**: compute all triangles
  \[ \text{Triangles}(x, y, z) = R(x, y), \ S(y, z), \ T(z, x) \]
Triangles in One Round

- Place servers in a cube \( p = p^{1/3} \times p^{1/3} \times p^{1/3} \)
- Each server identified by \((i,j,k)\)

\[
\text{Triangles}(x,y,z) = R(x,y), \ S(y,z), \ T(z,x) \quad |R| = |S| = |T| = m \text{ tuples}
\]
### Triangles in One Round

#### Round 1:
- Send $R(x,y)$ to all servers $(h_1(x), h_2(y), \ast)$
- Send $S(y,z)$ to all servers $(\ast, h_2(y), h_3(z))$
- Send $T(z,x)$ to all servers $(h_1(x), \ast, h_3(z))$

#### Output:
- Compute locally $R(x,y) \bowtie S(y,z) \bowtie T(z,x)$

---

<table>
<thead>
<tr>
<th>$R$</th>
<th>$X$</th>
<th>$Y$</th>
<th>$T$</th>
<th>$Z$</th>
<th>$X$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>Alice</td>
<td>Fred</td>
<td>Alice</td>
<td>Jim</td>
<td>Alice</td>
</tr>
<tr>
<td>Jack</td>
<td>Jim</td>
<td>Jim</td>
<td>Alice</td>
<td>Jim</td>
<td>Alice</td>
</tr>
<tr>
<td>Fred</td>
<td>Jim</td>
<td>Fred</td>
<td>Alice</td>
<td>Jim</td>
<td>Alice</td>
</tr>
<tr>
<td>Carol</td>
<td>Alice</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Example Calculation:**

\[
\text{Triangles}(x,y,z) = R(x,y), S(y,z), T(z,x)
\]

\[
|R| = |S| = |T| = m \text{ tuples}
\]
Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

Input data = size $m$

Number of servers = $p$

Input (size=$m$) $\mathcal{O}(m/p)$

Server 1 . . . Server $p$ $\mathcal{O}(m/p)$
Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

Input data = size $m$

Number of servers = $p$

One round = Compute & communicate
Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

Input data = size $m$

Number of servers = $p$

One round = Compute & communicate

Algorithm = Several rounds

Input (size = $m$)

$O(m/p)$

Number of servers = $p$

One round = Compute & communicate

Algorithm = Several rounds

$O(m/p)$
Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

Input data = size $m$

Number of servers = $p$

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = $L$

Input (size=$m$)  

$O(m/p)$  

$\leq L$  

$\leq L$  

$\leq L$  

Round 1

Round 2

Round 3
Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

Input data = size \( m \)

Number of servers = \( p \)

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = \( L \)

Cost:
Load \( L \)
Rounds \( r \)
Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

Input data = size $m$

Number of servers = $p$

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = $L$

<table>
<thead>
<tr>
<th>Cost:</th>
<th>Naïve 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load $L$</td>
<td>$L = m$</td>
</tr>
<tr>
<td>Rounds $r$</td>
<td>1</td>
</tr>
</tbody>
</table>
Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

Input data = size $m$

Number of servers = $p$

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = $L$

<table>
<thead>
<tr>
<th>Cost:</th>
<th>Naïve 1</th>
<th>Naïve 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load $L$</td>
<td>$L = m$</td>
<td>$L = m/p$</td>
</tr>
<tr>
<td>Rounds $r$</td>
<td>1</td>
<td>$p$</td>
</tr>
</tbody>
</table>
Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

Input data = size m

Number of servers = p

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = L

<table>
<thead>
<tr>
<th>Cost:</th>
<th>Ideal</th>
<th>Naïve 1</th>
<th>Naïve 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load L</td>
<td>L = m/p</td>
<td>L = m</td>
<td>L = m/p</td>
</tr>
<tr>
<td>Rounds r</td>
<td>1</td>
<td>1</td>
<td>p</td>
</tr>
</tbody>
</table>
Massively Parallel Communication Model (MPC)

Extends BSP [Valiant]

Input data = size \( m \)

Number of servers = \( p \)

One round = Compute & communicate

Algorithm = Several rounds

Max communication load / round / server = \( L \)

<table>
<thead>
<tr>
<th>Cost:</th>
<th>Ideal</th>
<th>Practical ( \varepsilon \in (0,1) )</th>
<th>Naïve 1</th>
<th>Naïve 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load ( L )</td>
<td>( L = m/p )</td>
<td>( L = m/p^{1-\varepsilon} )</td>
<td>( L = m )</td>
<td>( L = m/p )</td>
</tr>
<tr>
<td>Rounds ( r )</td>
<td>1</td>
<td>( O(1) )</td>
<td>1</td>
<td>( p )</td>
</tr>
</tbody>
</table>
Data Complexity of the Load

• Query $Q$ is fixed

• Input: data $m$, servers $p$

• Load: $L = f(m, p)$

• We discuss one round only
A load of $L = \frac{m}{p}$ corresponds to linear speedup.

A load of $L = \frac{m}{p^{1-\varepsilon}}$ corresponds to sub-linear speedup.
Example: $\text{Join}(x,y,z) = R(x,y), S(y,z)$

Input: $R, S$
- Uniformly partitioned

Round 1: each server
- Sends record $R(x,y)$ to server $h(y) \mod p$
- Sends record $S(y,z)$ to server $h(y) \mod p$

Output: each server
- local join $R(x,y) \bowtie S(y,z)$

Assuming no skew

$L = O(m/p)$ w.h.p.
Example: Triangles

\[ \text{Triangles}(x,y,z) = R(x,y), S(y,z), T(z,x) \]

|R| = |S| = |T| = m tuples

Round 1:
- Send \( R(x,y) \) to all servers \((h_1(x), h_2(y), *)\)
- Send \( S(y,z) \) to all servers \((*, h_2(y), h_3(z))\)
- Send \( T(z,x) \) to all servers \((h_1(x), *, h_3(z))\)

Output:
- compute locally \( R(x,y) \bowtie S(y,z) \bowtie T(z,x) \)

**Theorem** If data has “no skew”, then HyperCube computes \textbf{Triangles} with load/server \( L = O(m/p^{2/3}) \) w.h.p.
Triangles$(x,y,z) = R(x,y), S(y,z), T(z,x)$

$|R| = |S| = |T| = 1.1M$

1.1M triples of Twitter data $\rightarrow$ 220k triangles; $p=64$

**local 1 or 2-step hash-join; local 1-step Leapfrog Trie-join (a.k.a. Generic-Join)**

**Wall clock time**

**Total CPU time**

**Number of tuples shuffled**

- 2 rounds hash-join
- 1 round broadcast
- 1 round hypercube
1.1M triples of Twitter data $\rightarrow$ 220k triangles; $p=64$

\[ \text{Triangles}(x, y, z) = R(x, y), S(y, z), T(z, x) \]

<table>
<thead>
<tr>
<th>shuffle</th>
<th>tuples sent</th>
<th>producer skew</th>
<th>consumer skew</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R(x, y) \rightarrow h(y)$</td>
<td>1,114,289</td>
<td>1</td>
<td>1.35</td>
</tr>
<tr>
<td>$S(y, z) \rightarrow h(y)$</td>
<td>1,114,289</td>
<td>1</td>
<td>1.72</td>
</tr>
<tr>
<td>$RS(x, y, z) \rightarrow h(z)$</td>
<td>50,862,578</td>
<td>20.8</td>
<td>1</td>
</tr>
<tr>
<td>$T(z, x) \rightarrow h(z)$</td>
<td>1,114,289</td>
<td>1</td>
<td>1.01</td>
</tr>
<tr>
<td>Total</td>
<td>54,205,445</td>
<td>N.A.</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

**Table 2: Load balance with regular shuffles in query Q1**

<table>
<thead>
<tr>
<th>shuffles</th>
<th>tuples sent</th>
<th>producer skew</th>
<th>consumer skew</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCS $R(x, y)$</td>
<td>4,457,156</td>
<td>1</td>
<td>1.05</td>
</tr>
<tr>
<td>HCS $S(y, z)$</td>
<td>4,457,156</td>
<td>1</td>
<td>1.05</td>
</tr>
<tr>
<td>HCS $T(z, x)$</td>
<td>4,457,156</td>
<td>1</td>
<td>1.05</td>
</tr>
<tr>
<td>Total</td>
<td>13,371,468</td>
<td>N.A.</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

**Table 3: Load balance with HyperCube shuffles in query Q1**
HperCube Algorithm for Full CQ

$$Q(x_1, \ldots, x_k) = S_1(\overline{x}_1), S_2(\overline{x}_2), \ldots, S_{\ell}(\overline{x}_{\ell})$$

- **Write:** $p = p_1 \times p_2 \times \ldots \times p_k$

- **Round 1:** send $S_j(x_{j1}, x_{j2}, \ldots)$ to all servers whose coordinates agree with $h_{j1}(x_{j1}), h_{j2}(x_{j2}), \ldots$

- **Output:** compute $Q$ locally

$p_i = \text{a “share”}$

independent hash functions
Computing Shares $p_1, p_2, \ldots, p_k$

$Q(x_1, \ldots, x_k) = S_1(\bar{x}_1), S_2(\bar{x}_2), \ldots, S_\ell(\bar{x}_\ell)$

Load/server from $S_j$:

$$L_j = \frac{m_j}{(p_{j1} \times p_{j2} \times \ldots)}$$

Optimization problem: find $p_1 \times p_2 \times \ldots \times p_k = p$

[Beame’13] linear opt:

Minimize $\max_j L_j$
Vertex Cover / Edge Packing

Hyper-graph: nodes $x_1, \ldots, x_k$, edges $S_1, \ldots, S_l$

**Def. A fractional vertex cover:** $k$ numbers $v_1, v_2, \ldots, v_k \geq 0$ such that for every hyperedge $S_j$: $\sum_i v_i \geq 1$

**Def. A fractional edge packing:** $l$ numbers $u_1, u_2, \ldots, u_l \geq 0$ such that for every node $x_i$: $\sum_j u_j \leq 1$

$$\min_{v_1, \ldots, v_k} \sum_i v_i \geq \max_{u_1, \ldots, u_l} \sum_j u_j = T^*$$
Optimal Load

\[ Q(x_1, \ldots, x_k) = S_1(\bar{x}_1), S_2(\bar{x}_2), \ldots, S_\ell(\bar{x}_\ell) \]

Suppose \( m_1 = \ldots = m_\ell = m \)

Optimal load: \( L = \frac{m}{p^{1/\tau}} \)  
Optimal speedup: \( 1/p^{1/\tau} \)

\begin{align*}
\text{Thm.} & \quad \text{For every } v_1, \ldots, v_k, \text{ there exists an algorithm for } Q \\
& \quad \text{with } L \leq m / p^{1/\tau} \text{ w.h.p.} \\
& \quad \text{on skew-fee instances, where } \tau = \sum_i v_i \\
\text{Proof:} & \quad \text{use shares } p_i = p^{v_i / \tau}
\end{align*}

\begin{align*}
\text{Thm.} & \quad \text{For every } u_1, \ldots, u_\ell, \text{ any algorithm for } Q \\
& \quad \text{has } L \geq m / p^{1/\tau} \text{ even on skew-fee instances, where } \tau = \sum_j u_j \\
\text{Proof:} & \quad \text{uses Friedgut’s inequality}
\end{align*}
Example

\[ Q(x,y,z) = R(x,y), S(y,z), T(z,x) \]

Suppose \( m_R = m_S = m_T = m \)

Fractional vertex packing:

\[
\begin{align*}
\min(v_R + v_S + v_T) \\
R: & \quad v_x + v_y \geq 1 \\
S: & \quad v_y + v_S \geq 1 \\
T: & \quad v_x + v_z \geq 1
\end{align*}
\]

Fractional edge cover:

\[
\begin{align*}
\max(w_R + w_S + w_T) \\
x: & \quad w_R + w_T \leq 1 \\
y: & \quad w_R + w_S \leq 1 \\
z: & \quad w_S + w_T \leq 1
\end{align*}
\]

Optimal load: \( L = \frac{m}{p^{2/3}} \)  
Optimal speedup: \( 1/p^{2/3} \)
Lessons So Far

• MPC model: cost = communication load + rounds

• **HyperCube**: rounds = 1, $L = \frac{m}{p^{1/\tau^*}}$ Sub-linear speedup
  Note: it only shuffles data! Still need to compute $Q$ locally.

• Strong optimality guarantee: any algorithm with better load $\frac{m}{p^s}$ reports only $\frac{1}{p^{s \times \tau^* - 1}}$ fraction of answers.

  Parallelism gets harder as $p$ increases!

• Total communication = $p \times L = m \times p^{1-1/\tau^*}$

  MapReduce model is wrong! It encourages many reducers $p$
Skew Matters

- If the database is skewed, the query becomes provably harder. We want to optimize for the common case (skew-free) and treat skew separately.

- Parallel processing different from sequential processing, were worst-case optimal algorithms (LFTJ, generic-join) are for arbitrary instances, skewed or not.
Skew Matters

- Join\((x,y,z) = R(x,y), S(y,z)\)

\[ L = \frac{m}{p} \]

- Suppose \(R, S\) are skewed, e.g. single value \(y\)

- The query becomes a cartesian product!

Product\((x,z) = R(x), S(z)\)

\[ L = \frac{m}{p^{1/2}} \]

Let's examine skew...
All You Need to Know About Skew

Hash-partition a bag of $m$ data values to $p$ bins

**Fact 1** Expected size of any one fixed bin is $m/p$

**Fact 2** Say that database is *skewed* if some value has degree $> m/p$. Then some bin has load $> m/p$

**Fact 3** Conversely, if the database is *skew-free* then max size of all bins $= O(m/p)$ w.h.p.

Join: if $\forall$ degree $< m/p$ then $L = O(m/p)$ w.h.p

Triangles: if $\forall$ degree $< m/p^{1/3}$ then $L = O(m/p^{2/3})$ w.h.p

In general: if $\forall$ degree $< m/p^{vi/\tau^*}$ then $L = O(m/p^{1/\tau^*})$ w.h.p

Hiding log $p$ factors
AGM Bound + Optimal Alg.

Q(x,y,z) = R(x,y), S(y,z), T(z,x)

(Generalized) fractional vertex packing:

\[
\max(v_R + v_S + v_T)
\]

R: \(v_x + v_y \leq \log|R|\)
S: \(v_y + v_S \leq \log|S|\)
T: \(v_x + v_z \leq \log|T|\)

(Generalized) fractional edge cover:

\[
\min(w_R \log|R| + w_S \log|S| + w_T \log|T|)
\]

x: \(w_R + w_T \geq 1\)
y: \(w_R + w_S \geq 1\)
z: \(w_S + w_T \geq 1\)

Thm. For any feasible \(v_R, v_S, v_T\):

\[
\log|Q| \geq \text{objective} \geq m^{v_x} \times m^{v_y} \times m^{v_z}
\]

Proof. “Free” instance

\[
R(x,y) = [m^{v_x}] \times [m^{v_y}]
\]
\[
S(y,z) = [m^{v_y}] \times [m^{v_z}]
\]
\[
T(z,x) = [m^{v_x}] \times [m^{v_z}]
\]

Optimal edge cover = \(\rho^*\)
AGM bound = \(m^{\rho^*}\)
Generic-join algorithm, time = O\((m^{\rho^*})\)

Proof. (use Friedgut’s inequality)

Thm. For any feasible \(w_R, w_S, w_T\):

\[
\log|Q| \leq \text{objective} \leq |R|^{w_R} \times |S|^{w_S} \times |T|^{w_T}
\]
The AGM Inequality

\[ Q(x_1, \ldots, x_k) = S_1(\bar{x}_1), S_2(\bar{x}_2), \ldots, S_{\ell}(\bar{x}_{\ell}) \]

\[ m_1 = \ldots = m_\ell = m \]

**Fact.** Any MPC algorithm satisfies \( r \times L \geq m / p^{1/\rho^*} \)
where \( r = \# \) rounds, \( L = \) load/server

**Informal proof:**
- AGM lower bound: there exists db s.t. \( |Q| = m^{\rho^*} \)
- Each server receives subset \( |S_i| \leq r \times L \)
- AGM upper bound: server reports \( \leq (r \times L)^{\rho^*} \) answers
- All \( p \) servers report \( \leq p \times (r \times L)^{\rho^*} \) answers, = \( m^{\rho^*} \)
Lessons so Far

• Skew affects communication dramatically
  – E.g. Join from linear $m/p$ to $m/p^{1/2}$

• Analysis differs:
  – w/o skew: $L = m / p^{1/\tau^*}$ fractional vertex cover
  – w/ skew: $L \geq m / p^{1/\rho^*}$ fractional edge cover

• Focus on skew-free databases.
  Handle skewed values as a residual query.
Statistics

- So far: all relations have same size $m$
- In reality, we know their sizes = $m_1, m_2, \ldots$

**Q1**: What is the optimal choice of shares?
**Q2**: What is the optimal load $L$?

Will answer **Q2**, giving closed formula for $L$. Will answer **Q1** indirectly, by showing that HyperCube takes advantage of statistics.
2-way product \( Q(x,y) = S_1(x) \times S_2(y) \)

Shares \( p = p_1 \times p_2 \)

\[ L = \max\left(\frac{m_1}{p_1}, \frac{m_2}{p_2}\right) \]

Minimal \( L \):

\[ L = \left(\frac{m_1 \cdot m_2}{p}\right)^{\frac{1}{2}} \]

t-way product: \( Q(x_1,\ldots,x_u) = S_1(x_1) \times \cdots \times S_t(x_t) \):

\[ L = \left(\frac{m_1 \cdot m_2 \cdots m_t}{p}\right)^{\frac{1}{t}} \]
L for arbitrary Query Q

\[ Q(x_1, \ldots, x_k) = S_1(\bar{x}_1), S_2(\bar{x}_2), \ldots, S_\ell(\bar{x}_\ell) \]

Relations sizes = \( m_1, m_2, \ldots \) Then, for any 1-round algorithm

**Fact** For any integral edge packing \( S_{j_1}, S_{j_2}, \ldots, S_{j_t} \):

\[ L \geq \left( \frac{m_{j_1} \cdot m_{j_2} \cdots m_{j_t}}{\text{p}} \right)^{\frac{1}{t}} \]

\( t = \text{size of packing} \)

**Theorem: [Beame’14]**

(1) For any fractional edge packing \( u_1, \ldots, u_\ell \)

\[ L \geq L(u) = \left( \frac{m_1^{u_1} \cdot m_2^{u_2} \cdots m_\ell^{u_\ell}}{\text{p}} \right)^{\frac{1}{u_1 + u_2 + \cdots + u_\ell}} \]

(2) The optimal load of the HyperCube algorithm is \( \max_u L(u) \)
Example

Triangles \((x,y,z) = R(x,y), S(y,z), T(z,x)\)

<table>
<thead>
<tr>
<th>Edge packing</th>
<th>(\left( \frac{m_1 u_1 \cdot m_2 u_2 \cdot m_3 u_3}{p} \right)^{\frac{1}{u_1+u_2+u_3}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(u_1, u_2, u_3)</td>
<td></td>
</tr>
</tbody>
</table>
Example

Triangles\((x,y,z) = R(x,y), S(y,z), T(z,x)\)

<table>
<thead>
<tr>
<th>Edge packing</th>
<th>(\left( \frac{\mathbf{m}_1 \cdot \mathbf{m}_2 \cdot \mathbf{m}_3}{\mathbf{m}_1 \cdot \mathbf{m}_2 \cdot \mathbf{m}_3} \right)^{1/3} / \frac{1}{\mathbf{m}_1 + \mathbf{m}_2 + \mathbf{m}_3} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1/2, 1/2, 1/2)</td>
<td>((\mathbf{m}_1 \mathbf{m}_2 \mathbf{m}_3)^{1/3} / \mathbf{p}^{2/3})</td>
</tr>
</tbody>
</table>
Example

Triangles(x,y,z) = R(x,y), S(y,z), T(z,x)

<table>
<thead>
<tr>
<th>Edge packing ( u_1, u_2, u_3 )</th>
<th>( \left( \frac{m_1^{u_1} \cdot m_2^{u_2} \cdot m_3^{u_3}}{p} \right)^{\frac{1}{u_1+u_2+u_3}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{1}{2}, \frac{1}{2}, \frac{1}{2} )</td>
<td>( (m_1 m_2 m_3)^{1/3} / p^{2/3} )</td>
</tr>
<tr>
<td>( 1, 0, 0 )</td>
<td>( m_1 / p )</td>
</tr>
</tbody>
</table>
### Example

Triangles\((x,y,z) = R(x,y), S(y,z), T(z,x)\)

<table>
<thead>
<tr>
<th>Edge packing</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>(u_1, u_2, u_3)</td>
<td>(\left( \frac{m_1^{u_1} \cdot m_2^{u_2} \cdot m_3^{u_3}}{p} \right) \frac{1}{u_1+u_2+u_3})</td>
</tr>
<tr>
<td>1/2, 1/2, 1/2</td>
<td>((m_1 m_2 m_3)^{1/3} / p^{2/3})</td>
</tr>
<tr>
<td>1, 0, 0</td>
<td>(m_1 / p)</td>
</tr>
<tr>
<td>0, 1, 0</td>
<td>(m_2 / p)</td>
</tr>
<tr>
<td>0, 0, 1</td>
<td>(m_3 / p)</td>
</tr>
</tbody>
</table>

\(L = \) the largest of these four values.
Example

Triangles(x,y,z) = R(x,y), S(y,z), T(z,x)

<table>
<thead>
<tr>
<th>Edge packing</th>
<th>( \left( \frac{m_1^{u_1} \cdot m_2^{u_2} \cdot m_3^{u_3}}{p} \right)^{1/3} )</th>
<th>( \frac{1}{u_1+u_2+u_3} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2, 1/2, 1/2</td>
<td>( (m_1 m_2 m_3)^{1/3} / p^{2/3} )</td>
<td></td>
</tr>
<tr>
<td>1, 0, 0</td>
<td>( m_1 / p )</td>
<td></td>
</tr>
<tr>
<td>0, 1, 0</td>
<td>( m_2 / p )</td>
<td></td>
</tr>
<tr>
<td>0, 0, 1</td>
<td>( m_3 / p )</td>
<td></td>
</tr>
</tbody>
</table>

\( L = \) the largest of these four values.

Assuming \( m_1 > m_2, m_3 \)
- When \( p \) is small, then \( L = \frac{m_1}{p} \).
- When \( p \) is large, then \( L = \frac{(m_1 m_2 m_3)^{1/3}}{p^{2/3}} \).
Fact 1 \( L = \left[ \text{geometric-mean of } m_1, m_2, \ldots \right] / p^{1/\Sigma u_j} \)

Fact 2 As \( p \) increases, speedup degrades.
\[ 1/p^{1/\Sigma u_j} \rightarrow 1/p^{1/\tau^*} \]

Fact 3 If \( m_j < m_k/p \), then \( u_j = 0 \).
Intuitively: broadcast the small relations \( S_j \)
Summary

• Traditional query plans are sub-optimal for complex, cyclic queries
• Real data/applications: keys or UDF’s make traditional plans suboptimal even for acyclic queries
• Novel algorithm are now emerging, especially for large-scale data analytics