Join Algorithms

Qiu, Yuan and Zhang, Haoqian
Contents

1. Join algorithms in the lecture
   a. Nested-Loop Join
   b. Sort-merge Join
   c. Hash Join

2. Preliminaries
   a. Join as Hypergraphs
   b. The AGM bound for output size

3. More Join algorithms
   b. Generic Join algorithm.................................2013
   c. Hypercube algorithm.................................2011
   d. Degree-based algorithms............................2015+
1. Binary Join algorithms revisited

- **Nested-Loop Join**
  - Do a “for” loop for each tuple in each relation.
  - Each time read $O(M)$ tuples from $R_1$ and $O(M)$ from $R_2$.
  - Cost of Join: $O\left(\frac{N_1 N_2 M}{M^2 B}\right)$
  - This can be generalized to arbitrary join $R_1 \Join R_2 \Join \cdots \Join R_n$
  - Cost of Join (General case): $O\left(\frac{\prod_{i=1}^{n} N_i M}{M^n B}\right)$

- **Sort-Merge Join**
  - Sort both relations on the Join attribute.
  - Can only be applied to binary joins.

- **Hash Join**
  - Hash the tuples using the Join attribute.
  - The tuples to be joined together will “meet” in the same bucket.
  - There is a generalization of this in the Map-Reduce setting.
2. Join Operator

- **Natural Join (∇):** the result of \( R ∇ S \) is the set of all combinations of tuples in R and S that are equal on their common attributes.

<table>
<thead>
<tr>
<th>Employee</th>
<th>Dept</th>
<th>Employee ↯ Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Harry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sally</td>
</tr>
<tr>
<td></td>
<td></td>
<td>George</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Harriet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Finance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sales</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Production</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Harry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sally</td>
</tr>
<tr>
<td></td>
<td></td>
<td>George</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Harriet</td>
</tr>
</tbody>
</table>

- **Semijoin (∩, ⋈):** the result of \( R ⋈ S \) is the set of all tuples in R for which there is a tuple in S that is equal on their common attributes.

<table>
<thead>
<tr>
<th>Employee</th>
<th>Dept</th>
<th>Employee ↩ Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Harry</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sally</td>
</tr>
<tr>
<td></td>
<td></td>
<td>George</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Harriet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sales</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Production</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sally</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Harriet</td>
</tr>
</tbody>
</table>
2.a Join as Hypergraphs

Join result = subgraphs with specific structure

- L3: $R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,D)$
- C3: $R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(A,C)$
- T3: $R_0(A,B,C) \bowtie R_1(A,D_1) \bowtie R_2(B,D_2) \bowtie R_3(C,D_3)$

Vertices: values

Hyperedges: tuples
2.a Query as Hypergraphs

- L3: $R1(A,B) \bowtie R2(B,C) \bowtie R3(C,D)$
- C3: $R1(A,B) \bowtie R2(B,C) \bowtie R3(A,C)$
- T3: $R0(A,B,C) \bowtie R1(A,D1) \bowtie R2(B,D2) \bowtie R3(C,D3)$

Vertices: attributes
Hyperedges: relations
2.b The AGM Bound

- **Edge Cover problem:**

Given (hyper)graph $G = (V,E)$, assign real weights $w_1, \ldots, w_m \geq 0$ to each edge, s.t. for any vertex $x \in V : \sum_{e_i \ni x} w_i \geq 1$

The weights are called the Edge Covering numbers.

- **The AGM bound:**

Let $Q$ be a full join query $(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n)$. Let $w_i$ be the Edge Covering numbers for its corresponding hypergraph. The AGM bound says the output size $|OUT|$ is bounded by:

$$|OUT| \leq \prod_{i=1}^{n} |R_i|^{w_i}$$
2.b Examples of AGM bound

Assume all the relations have size N.

- L3: $R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(C,D)$
  
  \[ w_1 = 1, w_2 = 0, w_3 = 1, |\text{OUT}| \leq N^2 \]

- C3: $R_1(A,B) \bowtie R_2(B,C) \bowtie R_3(A,C)$
  
  \[ w_1 = w_2 = w_3 = 0.5, |\text{OUT}| \leq N^{1.5} \]

- T3: $R_0(A,B,C) \bowtie R_1(A,D_1) \bowtie R_2(B,D_2) \bowtie R_3(C,D_3)$
  
  \[ w_0 = 0, w_1 = w_2 = w_3 = 1, |\text{OUT}| \leq N^3 \]
3.a Yannakakis Algorithm

Query with running time $O(IN+OUT)$

- **IN**: Number of tuples in all relations
- **OUT**: Number of Join results

Algorithm:

- **Semi-join phase**: $O(IN)$
  - Bottom-up: from leaves to root, semijoin each relation with each of its children
  - Top-down: from root to leaves, semijoin each relation with its parent

- **Join phase**: $O(OUT)$
  - Join in arbitrary order
3.a Yannakakis Algorithm (Example)
3.a Generalized Yannakakis Algorithm

Yannakakis Algorithm only works for acyclic joins.

For cyclic queries, it is also possible to do some decompositions as well.

Example: Barbell query
3.a Generalized Yannakakis Algorithm

Generalized Hypergraph Decomposition:

Transform a hypergraph $G=(V, E)$ to a tree $T=(V', E')$ such that:

1. Each tree node $v'$ contains some relations and exactly their attributes.
2. All the edges in $E$ are covered.
3. All nodes $v'$ that contain a particular $v$ form a subgraph.
3.a Generalized Yannakakis Algorithm

For each tree node $v'$, there is an optimal edge cover number $w(v')$.

The fractional hypertree width (fhw) is the maximum of all the $w(v')$’s, so that by AGM bound, the output size from any node is bounded by $O(N^{fhw})$.

After the computation is done for each node, the remaining is simply run original Yannakakis algorithm, which now has running time $O(N^{fhw} + OUT)$.
3.a Generalized Yannakakis Algorithm

Another example of Generalized Hypergraph Decomposition

The running time of Generalized Yannakakis Algorithm is $\mathcal{O}(N^{1.5} + OUT)$
3.a Generalized Yannakakis Algorithm

Remarks:

The discussion about Yannakakis Algorithm is based on Internal Memory Model.

It has been extended to External Memory Model. However, even for binary join, the running time \( \tilde{O}\left(\frac{OUT}{B}\right) \) is not optimal if we do not need to write all outputs to the disk.

Note that even Nested-Loop Join is better!

\[
O\left(\frac{N_1 N_2 M}{M^2 B}\right) = O\left(\frac{OUT M}{M B}\right)
\]
3.b Generic Join

Generalized Yannakakis Algorithm cannot deal with the “core” of cyclic queries.

Consider Triangle query R(A, B), S(B, C), T(A, C):

Fact 1: Suppose |R|=|S|=|T|=N, then the AGM bound for triangle query is

\[ \text{OUT} = O(N^{1.5}) \]

Fact 2: Any pairwise join plan (e.g. sort-merge join) runs in

\[ \Omega(N^2) \]
3.b Generic Join

Hard instance for binary join plans:

Each attribute has value 0,1,...,N

Each relation has 2N tuples: (0,i) and (i,0) where i=1,...,N

There is no triangle because there is no (0,0) or (i,j) edge.

But if we join any two relations, there will be:

$N^2$ tuples (i, 0, j) for i=1,...,N, j=1,...,N

$N$ tuples (0,i,0) for i=1,...,N
3.b Generic Join

The algorithm works as follows:

```
Algorithm 2 Computing $Q_\Delta$ by delaying computation.
Input: $R(A, B), S(B, C), T(A, C)$ in sorted order

1: $Q \leftarrow \emptyset$
2: $L_A \leftarrow \pi_A R \cap \pi_A T$
3: For each $a \in L_A$
4:     $L_a^b \leftarrow \pi_B \sigma_{A=a} R \cap \pi_B S$
5:     For each $b \in L_a^b$
6:         $L_c^{a,b} \leftarrow \pi_C \sigma_{B=b} S \cap \pi_C \sigma_{A=a} T$
7:     For each $c \in L_c^{a,b}$
8:         Add $(a, b, c)$ to $Q$
9: Return $Q$
```

Consider $a=0$:

$\pi_R^A=[0,1,2]$  $\pi_T^A=[0,1,2]$  $L^A=[0,1,2]$

$\pi_R^B=[1,2]$  $\pi_S^B=[0,1,2]$  $L^B=[1,2]$

Consider $b=1$:

$\pi_S^C=[0]$  $\pi_T^C=[1,2]$  $L^C=[]$
3.b Generic Join

Remarks:

1. The generalized version basically has more nested for loops.
2. This is a worst-case optimal algorithm whose running time is bounded by AGM bound.
3. The key feature is that the running time of intersection is $O(\cdot)$ of the smaller set.
4. For general queries, this is usually combined with Generalized Yannakakis algorithm to provide $O(N^{fhw} + OUT)$ running time.
5. It may be sub-optimal when OUT is small (usually the case in real life examples).
3.c Hypercube algorithm (shares)

Consider the triangle join again, in a parallel computation setting with \( p = 64 \) processors.

Similar to a “Hash Join”, now we have 3 join attributes. So we hash each one.

With 64 processors, each attribute can have \( p_A = p_B = p_C = \sqrt[3]{64} = 4 \) buckets.

e.g. For tuple \( t = (a,b) \) in \( R \), hash it to all cells \([h(a), h(b), *]\)

Tuples in the blue processor has: \( h(a) = 1, h(b) = 3, h(c) = 4 \), so \( 3 \cdot (N/16) \) tuples.

Processor load: There are 4*4 buckets for each relation, each processor gets

\[
\frac{|R|}{p_A p_B} + \frac{|S|}{p_B p_C} + \frac{|T|}{p_C p_D} = \frac{N}{p^{2/3}} \text{ tuples.}
\]
3.d Degree-based Algorithms

In External Memory Join algorithms, we may have huge intermediate join results that we have to write to the disk.

Example: OUT = constant. But if we do (R join S) join T (as Yannakakis may choose), we have to write $N^2$ tuples to the disk!

Idea: This attribute value “b” is “heavy” in R (meaning a lot of tuples). So we do the other part first (BC Join CD), then go back and join with AB.
Conclusion

We have introduced several important join algorithms:

1. Yannakakis: The most classic and fundamental algorithm.
4. Degree-based: The focus of recent research.
References:


