Skew Strikes Back: New Developments in the Theory of Join Algorithms

Hung Q. Ngo, Christopher Ré, Atri Rudra

Presenter: Ayush Kanodia
Advisor: S. Sudarshan

Indian Institute of Technology Bombay

ashu@cse.iitb.ac.in

April 12, 2015

Overview I

1. Introduction

2. Motivation
   - The Triangle Query
   - The Triangle Query and the Relational Join
   - Why traditional join plans are suboptimal

3. Alternative Join Algorithms
   - Algorithm 1: The power of two choices
   - Algorithm 2: Delaying the computation
   - Approaches to Join Processing
   - Bridging this Gap
   - Key Takeaways

Overview II

4. The AGM bound
   - Description
   - The tightest AGM bound
   - Illustrations
   - Conjunctive queries with simple functional dependencies
   - A worst-case optimal join algorithm

5. Using the AGM bound
   - Proof Part 1 : Query Decomposition Lemma
   - Proof Part 2 : An Inductive Proof of the AGM inequality

6. Open Questions and Conclusion

Introduction

- This paper is a survey on recent theoretical advances in algorithm design for relational joins, one of the best studied problems in database systems
- In spite of this, the textbook (as well as a large component of the research literature) description of join processing is suboptimal - *In the worst case sense*
- This survey describes recent results on join algorithms that have provable worst case optimality runtime guarantees

The Triangle Query

- The *Triangle Query* helps us understand much of the progress in this direction - a query which has gained popularity off late

  Suppose that one is given a graph with N edges, how many distinct triangles can there be in the graph?

  - $O(N^3)$?
  - $O(N^2)$? As a Triangle is indexed uniquely by any two of its sides
  - Lesser?
- The correct non trivial bound is $O(N^{3/2})$

The Triangle Query and the Relational Join

- The triangle query can be viewed as a relational join problem. For a given query and a graph, construct three relations $R(A, B)$, $S(B, C)$, $T(C, A)$, and every pair of vertices (i, j) such that there is an edge connecting them exists in each of the three relations.
- The resulting number of triangles in the graph is given by the natural join of R, S and T
- Each triangle in the original graph will show up six times in the relational join due to symmetricity
- We now consider the following natural join query - $Q_\Delta = R(A, B) \bowtie S(B, C) \bowtie T(C, A)$

Why traditional join plans are suboptimal

- **Assumption:** We have very *nice* indices on the relations that we are joining, and all of them are in memory. This assumption is to ease understanding, and may be relaxed in reality.
- The traditional method to evaluate this join query would be one of the three following pairwise joins.

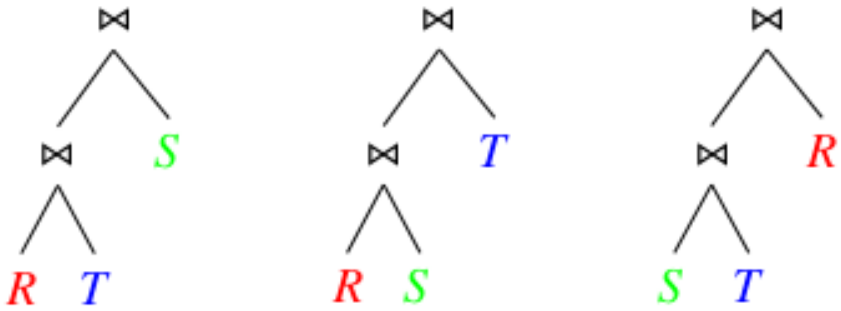

Figure : The three pairwise joins

Why traditional join plans are suboptimal

- Now consider the following database instance

$$R = \{a_0\} \times \{b_0, \ldots, b_m\} \cup \{a_0, \ldots, a_m\} \times \{b_0\}$$

$$S = \{b_0\} \times \{c_0, \ldots, c_m\} \cup \{b_0, \ldots, b_m\} \times \{c_0\}$$

$$T = \{a_0\} \times \{c_0, \ldots, c_m\} \cup \{a_0, \ldots, a_m\} \times \{c_0\}$$

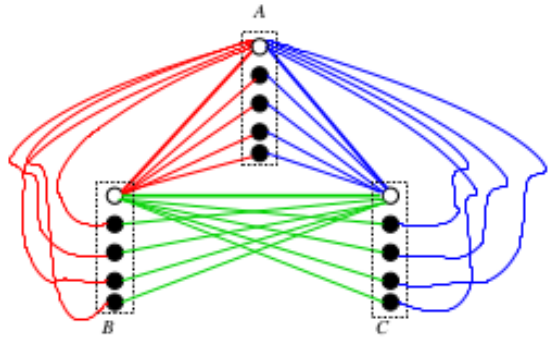

Figure : The database instance which is a counter example for the optimality of pair wise joins, and an illustration for m = 4. The pairs connected by red/blue/green edges from the tuples in R/S/T respectively. Note that in case each relation has $N = 2m + 1 = 9$ tuples and there are $3m + 1 = 13$ output tuples in Q_Δ. Any pair-wise join however has size $m^2 + m = 20$

Algorithm 1: The power of two choices

- The root cause for the large amount of time taken for the above computation is that a_0 has high degree, and is an instance of skew.
- To cope with this, we introduce a new idea - that of dealing with nodes of high and low skew using different join techniques.
- The first goal is to understand when some value has high skew. For each a_i define

$$Q_\Delta[a_i] := \Pi_{B,C}(\sigma_{A=a_i}(Q_\Delta)) \tag{1}$$

and we call a_i heavy if

$$|\sigma_{A=a_i}(R \bowtie T)| \geq |Q_\Delta[a_i]| \tag{2}$$

Algorithm 1: The power of two choices

- Since

$$|\sigma_{A=a_i}(R \bowtie S)| = |\sigma_{A=a_i}(R)||\sigma_{A=a_i}(S)| \tag{3}$$

  We can easily compute the LHS of the previous equation. For the RHS however, we need Q_Δ, but since $Q_\Delta[a_i] \subseteq S$, we can use $|S|$ as a proxy for $Q_\Delta[a_i]$
- We now describe the two choices themselves
  1. Compute $\sigma_{A=a_i}(R) \bowtie \sigma_{A=a_i}(T)$ and filter the results by probing against S, or
  2. Consider each tuple $(b, c) \in S$ and check if $(a_i, b) \in R$ and $(a_i, c) \in T$
- The algorithm is : We choose (i) when a_i is light and (ii) if a_i is heavy

Algorithm 1: The power of two choices

Algorithm 1 Computing $Q_\triangle$ with power of two choices.

Input: $R(A,B), S(B,C), T(A,C)$ in sorted order

```
1:  Q△ ← ∅
2:  L ← π_A(R) ∩ π_A(T)
3:  For each a ∈ L do
4:      If |σ_{A=a}R| · |σ_{A=a}T| ⩾ |S| then
5:          For each (b, c) ∈ S do
6:              If (a, b) ∈ R and (a, c) ∈ T then
7:                  Add (a, b, c) to Q△
8:      else
9:          For each b ∈ π_B(σ_{A=a}R) ∧ c ∈ π_C(σ_{A=a}T) do
10:             If (b, c) ∈ S then
11:                 Add (a, b, c) to Q△
12: Return Q
```

Figure : Computing Q_Δ with the power of two choices

Algorithm 1: The power of two choices

$$R = \{a_0\} \times \{b_0, \ldots, b_m\} \cup \{a_0, \ldots, a_m\} \times \{b_0\}$$

$$S = \{b_0\} \times \{c_0, \ldots, c_m\} \cup \{b_0, \ldots, b_m\} \times \{c_0\}$$

$$T = \{a_0\} \times \{c_0, \ldots, c_m\} \cup \{a_0, \ldots, a_m\} \times \{c_0\}$$

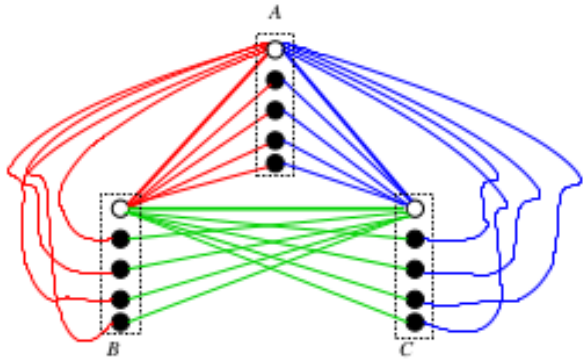

Figure : The database instance which is a counter example for the optimality of pair wise joins

- Example : Let us work through the motivating example that was given.

Algorithm 1: The power of two choices

- When we compute $Q_\Delta[a_0]$, we realise that a_0 is heavy and hence, we use option (ii) as described. Since here we need to scan tuples in S, this takes $O(m)$ time.
- On the other hand, when we want to compute $Q_\Delta[a_i]$ for $i \geq 1$, we realise these a_is are light, and so we take option (i). In these cases, the time taken is $O(1)$ since $|\sigma_{A=a_i}R| = |\sigma_{A=a_i}R| = 1$.
- As there are m such light a_i's, the algorithm overall takes $O(m)$ each on the heavy and light vertices and thus $O(m) = O(N)$ overall which is best possible since the output size is $\Theta(N)$.

Algorithm 1: The power of two choices

For the analysis of the algorithm, we refer to the paper itself. The time taken is proven to be $O(N^{3/2})$, which is the intended result.

Algorithm 2: Delaying the computation

- We now present a different algorithm which differentiates between heavy and light values $a_i \in A$ in a different way.
- Instead of estimating the heaviness of a_i directly, this algorithm "looks deeper" into what pair (b, c) can go along with the a_i in the output by computing c for each candidate b
- We compute the intersection $\pi_B(\sigma_{A=a_i} R) \cap \pi_B S$, thus looking at only the candidates b that can possibly participate with a_i
- Then, the candidate set (for each b) for c is $\pi_C(\sigma_{B=b} S) \cap \pi_C(\sigma_{A=a_i} T)$
- When a_i is really skewed toward the heavy side, the candidates b and c reduce the skew toward building up the final solution Q_Δ

Algorithm 2: Delaying the Computation

Algorithm 2 Computing Q_Δ 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 $a \in L_A$ **do**
4: $\quad L_B^a \leftarrow \pi_B \sigma_{A=a} R \cap \pi_B S$
5: $\quad$ **For** each $b \in L_B^a$ **do**
6: $\quad\quad L_C^{a,b} \leftarrow \pi_C \sigma_{B=b} S \cap \pi_C \sigma_{A=a} T$
7: $\quad\quad$ **For** each $c \in L_C^{a,b}$ **do**
8: $\quad\quad\quad$ Add (a,b,c) to Q
9: **Return** Q

Figure : Computing Q_Δ by delaying the computation

Algorithm 2: Delaying the Computation

$$R = \{a_0\} \times \{b_0, \ldots, b_m\} \cup \{a_0, \ldots, a_m\} \times \{b_0\}$$

$$S = \{b_0\} \times \{c_0, \ldots, c_m\} \cup \{b_0, \ldots, b_m\} \times \{c_0\}$$

$$T = \{a_0\} \times \{c_0, \ldots, c_m\} \cup \{a_0, \ldots, a_m\} \times \{c_0\}$$

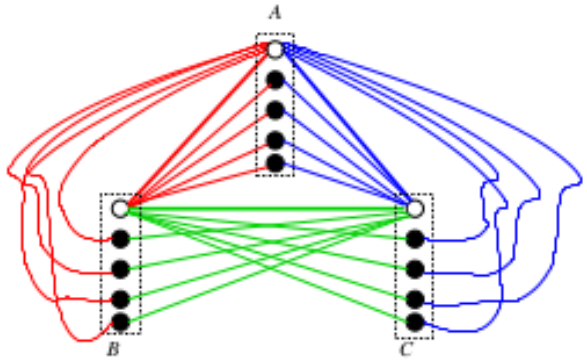

Figure : The database instance which is a counter example for the optimality of pair wise joins

- Example : Let us work through the motivating example that was given.

Algorithm 2: Delaying the Computation

- As observed earlier in using the *The power of two choices*, computing the intersection of two sorted sets takes time which is at most the *minimum* of the two sizes
- For a_0, we consider all $b \in \{b_0, b_1, ..., b_m\}$. When $b = b_0$, we have $\pi_C(\sigma_{B=b_0} S) = \pi_C(\sigma_{A=a_0} T) = \{c_0, ..., c_m\}$, so we output $m + 1$ triangles in total time $O(m)$.
- For the pairs (a_0, b_i), when $i \geq 1$, we have $|\sigma_{B=b_i} S| = 1$, and hence we spend $O(1)$ time on each such pair, for a total of $O(m)$ overall.

Algorithm 2: Delaying the Computation

- Now consider a_i for $i \geq 1$. In this case, $b = b_0$ is the only candidate. Further, for (a_i, b_0), we have $|\sigma_{A=a_i} T| = 1$, so we can handle each such a_i in $O(1)$ time leading to an overall runtime of $O(m)$.
- Thus on this example, this algorithm works in $O(N)$ time.
- For the full analysis of this algorithm, we will refer to the paper. It's worst case running time is exactly the same as that of Algorithm 1.
- What is remarkable is that both these algorithms follow exactly the same recursive structure and they are special cases of a generic worst-case optimal join algorithm

Traditional Approach 1

- One of the traditional approaches to join processing uses **structural properties** of the query
- The idea is to measure if the query being issued is 'acyclic' with respect to the database schema.
- When the query is acyclic, intuitively, we have *fast* algorithms for join processing

Traditional Approach 1

- Extensions of this idea expand the class of queries which can be evaluated in polynomial time.
- There are progressively more general notions of 'width' for a query, which intuitively measure how far a query is from being 'acyclic'.
- Roughly, these results state that if the corresponding notion of 'width' is bounded by a constant, then the query is 'tractable' (polynomial time solution algorithms exist).

Traditional Approach 2

- The algorithms which use the structural property of the query completely ignore the cardinality estimates of the relations involved in the join
- Structural approaches ignore the individual relation sizes, and summarizes them in a single number N, and as a result the runtime of these algorithms is $O(N^{(w+1)}logN)$, where w is the corresponding width measure.

Traditional Approach 2

- On the other hand, commercial RDBMSs seem to place little importance on the structure of the query and care only about the cardinality estimates
- Joins are processed by breaking them down into a series of two way joins - an approach described first in the seminal "System R".
- However, throwing away the structural information also comes at a cost, as we shall see later.

Bridging this Gap

- Atserias-Grohe-Marx (henceforth AGM) derived a tight bound on the output size of a join query as a function of individual input relation sizes and a general notion of query 'width'.
- AGM's bound takes into account both cardinality and structural information, and the bounds thus derived can be better than both cardinality and structural methods when the individual relation sizes vary, and when there is considerable skew.

Bridging this Gap

- Algorithms which follow the AGM bound have been developed very recently.
- Such an algorithm was first implemented in a commercial RDBMS system before its optimality properties were discovered.
- A key idea to the solution is to handle skew in a theoretically optimal fashion

Skew: The Devil's Own

- As we have seen, it is the old enemy of the database optimizer - skew, which is behind this connection
- There are two key messages to takeaway
  - The ideas that will henceforth be presented are an optimal way of avoiding skew. A theoretical basis for one family of techniques to tackle skew (by relating it to geometry) will be described.
  - We need to challenge the database dogma of doing "one join at a time", as is done in traditional DBMSs.

The AGM bound

- We now state the AGM bound
- The natural join question is as follows - We are given a collection of m relations. Each relation is over a collection of attributes.
- We use $\mathcal{V}$ to denote the set of attributes; let $n = |\mathcal{V}|$.
- The join query is modeled as a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where for each hyperedge $\mathcal{F} \in \mathcal{E}$ there is a relation R_F on attribute set F.
- Let $\mathbf{x} = (x_F)_{F \in \mathcal{E}}$ be any point in the following polyhedron - $\{\sum_{F:v \in F} x_F \geq 1, \forall v \in \mathcal{V}, \mathbf{x} \geq \mathbf{0}\}$
- Such a point $\mathbf{x}$ is called a *fractional edge cover* of the hypergraph $\mathcal{H}$

The AGM bound

- Then, the AGM bound states that

$$|Q| = |\bowtie_{F \in \mathcal{E}} R_F| \leq \Pi_{F \in \mathcal{E}} |R_F|^{x_F} \tag{4}$$

The tightest AGM bound

- The AGM bound depends on the relation sizes. To minimize the right hand side of the above equation, we can solve the following linear program

$$min\ \Sigma_{F \in \mathcal{E}}(log_2|R_F|).x_F$$

$$s.t.\ \Sigma_{F:v \in F} x_f \geq 1, v \in \mathcal{V}, \mathbf{x} \geq \mathbf{0}$$

- From this formulation, we see that the objective function depends on the database instance $\mathcal{D}$ on which the query is applied.
- Further, we also note that the objective function depends on the structural properties of the query with respect to the database schema.

Illustrations

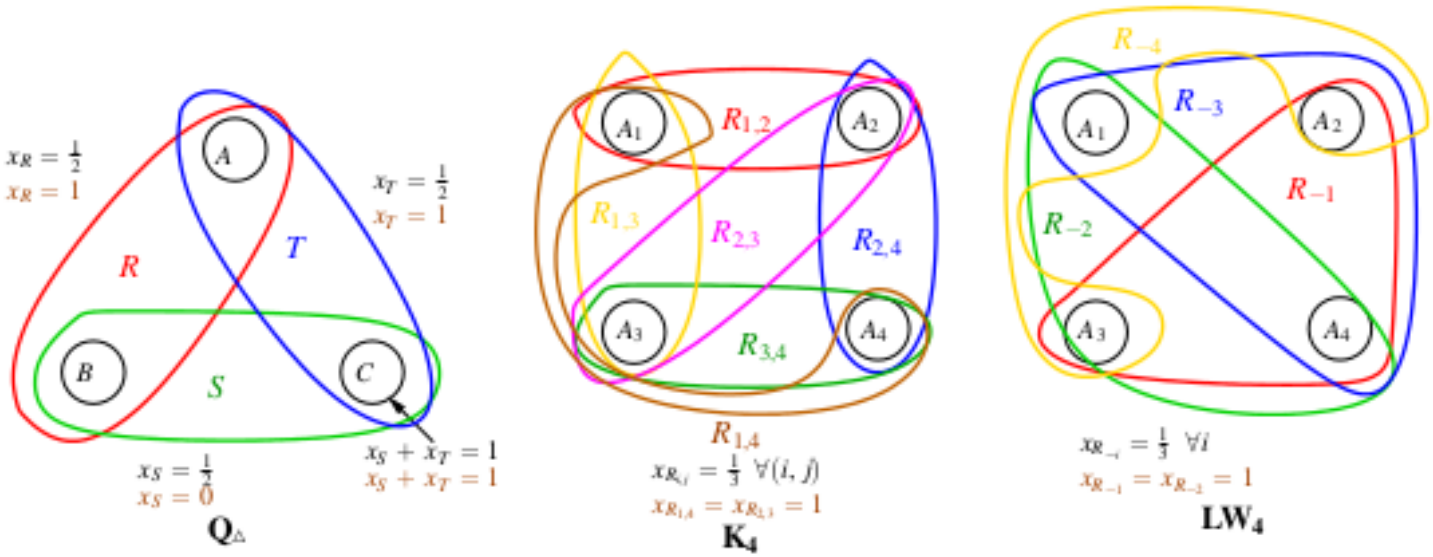

Figure : Some join queries and their covers

Conjuctive Queries

- **Projection** : Consider the following conjunctive query:
  $C_1 = R_0(W) \Leftarrow R(WX) \cap S(WY) \cap T(WZ)$
  - The AGM bound for this query is $O(N^3)$
  - However, since $R_0(W) \subseteq \pi_W(R) \bowtie \pi_W(S) \bowtie \pi_W(T)$, we can adapt the AGM bound to these relations, to get $O(N)$ as the new bound

Conjuctive Queries

- **Repeated Variables** : Consider the query
  $C_2 = R_0(WY) \Leftarrow R(WW) \cap S(WY) \cap T(YY)$
  - In this case we can replace $R(WW)$ and $T(YY)$ by keeping all tuples $(t_1, t_2) \in R$ for which $t_1 = t_2$, and similarly for T. This is now turned into the query
    $C_2' = R'(W) \bowtie S(WY) \bowtie T'(Y)$
  - For this query, $x_{R'} = x_{T'} = 0$ and $x_s = 1$ is a fractional cover and the AGM bound is hence $O(N)$

Functional Dependencies

- **Introducing the Chase:** Consider the following query $C_3 = R_0(WXY) \Leftarrow R(WX) \cap R(WW) \cap S(XY)$
- We can replace $R(WW)$ with $R'(W)$, and obtain a bound of $O(N^2)$
- Let $R = \{(i,i)|i \in [N/2]\} \cup (i,0)|i \in [N/2]$ and $S = \{(0,j)|j \in [N]\}$
- Every tuple $(i,0,j)$ for $i \in [N/2]$, $j \in [N]$ is in the output - $O(N^2)$

Functional Dependencies

- However, if we have that the first attribute in R is its key, then we can infer that (w, x, y) is an output tuple iff (w, x) and (w, w) are in R, and (x, y) are in S. We know that $x = w$.
- Hence, this is equivalent to
  $C_3' = R_0(WY) \Leftarrow R(WW) \cap S(WY)$
- The AGM bound for the output of this query is $O(N)$, and this transformation is a famous *chase* operation

A worst-case optimal join algorithm

- The proof of the AGM inequality leads to a natural formulation for a worst-case optimal join algorithm, which is depicted in the following figure

Algorithm 3 Generic-Join($\bowtie_{F\in\mathcal{E}} R_F$)

Input: Query Q, hypergraph $\mathcal{H} = (\mathcal{V},\mathcal{E})$

1: $Q \leftarrow \emptyset$
2: **If** $|\mathcal{V}| = 1$ **then**
3: return $\bigcap_{F\in\mathcal{E}} R_F$
4: Pick I arbitrarily such that $1 \leqslant |I| < |\mathcal{V}|$
5: $L \leftarrow$ Generic-Join$(\bowtie_{F\in\mathcal{E}_I} \pi_I(R_F))$
6: **For** every $\mathbf{t}_I \in L$ **do**
7: $Q[\mathbf{t}_I] \leftarrow$ Generic-Join$(\bowtie_{F\in\mathcal{E}_J} \pi_J(R_F \ltimes \mathbf{t}_I))$
8: $Q \leftarrow Q \cup \{\mathbf{t}_I\} \times Q[\mathbf{t}_I]$
9: **Return** Q

Figure : A worst-case optimal join algorithm based on the recursive proof of the AGM inequality

Query Decomposition Lemma

- We define $\mathcal{E}_I := \{F \in \mathcal{E} | F \cap I \neq \emptyset\}$
- *Query Decomposition Lemma*: Let $Q = \bowtie_{F \in \mathcal{E}} R_F$ be a natural join query represented by a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, and x be any fractional edge cover for $\mathcal{H}$. Let $\mathcal{V} = I \cup J$ be an arbitrary partition of V such that $1 \leq |I| \leq |\mathcal{V}|$; and, $L = \bowtie_{F \in \mathcal{E}_I} \pi_I(R_F)$.
- Then

$$\sum_{t_I \in L} \prod_{F \in \mathcal{E}_J} |R_F \ltimes \mathbf{t}_I|^{x_F} \leq \prod_{F \in \mathcal{E}} |R_F|^{x_F} \tag{5}$$

- For the complete proof as well as the inequality, we will refer to the paper

An Inductive Proof of the AGM inequality

- We induct over the number of attributes
- In the base case $|\mathcal{V}| = 1$, and we are computing the join of $|\mathcal{E}|$ unary relations. Let x be a fractional edge cover for this instance. Then,

$$
\begin{aligned}
|\bowtie_{F\in\mathcal{E}} R_F| &\leq min_{F\in\mathcal{E}}|R_F| \\
&\leq (min_{F\in\mathcal{E}}|R_F|)^{\Sigma_{F\in\mathcal{E}}x_F} \\
&= \prod_{F\in\mathcal{E}} (min_{F\in\mathcal{E}}|R_F|)^{x_F} \\
&\leq \prod_{F\in\mathcal{E}} |R_F|^{x_F}
\end{aligned}
$$

An Inductive Proof of the AGM inequality

- **Inductive Step**: Now assume $n = |\mathcal{V} \geq 2$. Let $\mathcal{V} = I \cup J$ be any partition of $\mathcal{V}$ such that $1 \leq |I| \leq |\mathcal{V}|$. Define $L = \bowtie_{F \in \mathcal{E}_I} \pi_I(R_F)$ as in the query decomposition lemma. For each tuple $\mathbf{t}_I \in L$ we define a new join query

$$Q[\mathbf{t}_I] := \bowtie_{F \in \mathcal{E}_J} \pi_J(R_F \ltimes \mathbf{t}_I)$$

- Then, the original query Q is

$$Q = \cup_{\mathbf{t}_I \in L}(\{\mathbf{t}_I\} x Q[\mathbf{t}_I])$$

- The vector $(x_F)_{F \in \mathcal{E}_J}$ is a fractional edge cover for the hypergraph of $Q[\mathbf{t}_I]$. Hence, the induction hypothesis gives us

$$|Q[\mathbf{t}_I]| \leq \prod_{F \in \mathcal{E}_J} |\pi_J(R_F \ltimes \mathbf{t}_I)|^{x_F} = \prod_{F \in \mathcal{E}_J} |R_F \ltimes \mathbf{t}_I|^{x_F}$$

An Inductive Proof of the AGM inequality

- Therefore, using the last two equations and the AGM inequality, we obtain

$$|Q| = \sum_{\mathbf{t}_I \in L} |Q[\mathbf{t}_I]| \leq \prod_{F \in \mathcal{E}} |R_F|^{x_F}$$

Open Questions and Conclusion

- This survey is concluded with two open questions.
- Can the algorithmic ideas presented in this paper gain runtime efficiency in database systems? On the one hand, this survey depicts asymptotic improvements in join algorithms, but on the other there are several decades of engineering refinements and research contributions in the traditional domain
- Worst-case results may only give information and efficiency gains for pathological instances. This leads to a push towards more refined measures of complexity of a natural join query. Current complexity measures do not give much insight into the use of indices, as well as the average case. Can one design an adaptive join algorithm whose runtime depends on the 'difficulty' of the input instance in some sense (instead of the input size, as is now).

Thank You