On Join Sampling and the Hardness of Combinatorial Output-Sensitive Join Algorithms

Based on the work of Shiyuan Deng, Shangqi Lu, and Yufei Tao (PODS 2023)

Presented by Jinchao Huang

Available at https://doi.org/10.1145/3584372.3588666

INTRODUCTION & MOTIVATION

WHAT IS A JOIN OPERATION?

- A fundamental operation in relational databases.
- Combines tuples from a set Q of multiple relations based on common attribute values.
- Example: Suppose $Q = \{R_{AB}, R_{BC}, R_{CA}\}$, then $Join(Q)$ is the set of all combinations of (a, b, c) where (a, b) is in R_{AB}, (b, c) is in R_{BC}, and (c, a) is in R_{CA}.

R_{AB}

A	B
0	1
0	2
1	1
2	1

R_{BC}

B	C
1	1
1	2
2	0
2	1

R_{CA}

C	A
0	1
1	0
2	1
2	2

Join(Q)

A	B	C
0	1	1
0	2	1
1	1	2
2	1	2

THE CHALLENGE: THE OUTPUT SIZE EXPLOSION

Joins are a primary performance bottleneck in database systems.

- The size of the join result, denoted OUT, can be massive.
- The theoretical upper bound on the output size is given by the AGM bound.
- In the worst case, for an input of size $\text{IN} = \sum |R|$:

$$\text{OUT} = \Omega(\text{IN}^{\rho^*})$$

  where ρ^* is the *fractional edge covering number* of the join query, a constant ≥ 1.
- For a simple 3-relation join, ρ^* can be 1.5. For more complex joins, it can be much larger.
- Even just writing the output can take $\Omega(\text{IN}^{\rho^*})$ time, which is prohibitive for large IN.

THE CHALLENGE: THE OUTPUT SIZE EXPLOSION

Joins are a primary performance bottleneck in database systems.

- The size of the join result, denoted OUT, can be massive.
- The theoretical upper bound on the output size is given by the AGM bound.
- In the worst case, for an input of size IN = $\sum |R|$:

$$\text{OUT} = \Omega(\text{IN}^{\rho^*})$$

  where ρ^* is the *fractional edge covering number* of the join query, a constant ≥ 1.
- For a simple 3-relation join, ρ^* can be 1.5. For more complex joins, it can be much larger.
- Even just writing the output can take $\Omega(\text{IN}^{\rho^*})$ time, which is prohibitive for large IN.

MOTIVATION: WHY SAMPLE FROM A JOIN?

Many applications do not need the full join result. A small set of random samples is often sufficient.

- **Approximate Query Processing:** Estimate aggregates like `SUM(sales)` or `AVG(price)` over a large join result quickly.
- **Data Exploration & Visualization:** Get a quick "feel" for the data distribution without waiting for the full join.
- **Machine Learning:** Use samples as a training set for models.
- **Fair Representative Reporting:** Select a few diverse tuples to represent the overall distribution.

The Join Sampling Problem

Design a data structure to support *sampling query*, which extracts a uniform sample from the join result Join(Q). Additionally, the sample returned by each query must be independent.

MOTIVATION: WHY SAMPLE FROM A JOIN?

Many applications do not need the full join result. A small set of random samples is often sufficient.

- **Approximate Query Processing:** Estimate aggregates like `SUM(sales)` or `AVG(price)` over a large join result quickly.
- **Data Exploration & Visualization:** Get a quick "feel" for the data distribution without waiting for the full join.
- **Machine Learning:** Use samples as a training set for models.
- **Fair Representative Reporting:** Select a few diverse tuples to represent the overall distribution.

The Join Sampling Problem

Design a data structure to support *sampling query*, which extracts a uniform sample from the join result $\text{Join}(Q)$. Additionally, the sample returned by each query must be independent.

STATE OF THE ART VS. THE CONTRIBUTION

Let IN be the input size, OUT be the output size, and ρ^* be the fractional edge covering number.

- **State of the Art** [Chen and Yi, ICDT'20]:
  - After $\tilde{O}(\text{IN})$ preprocessing, a sample can be drawn in time:

$$\tilde{O}(\text{IN}^{\rho^*+1}/\max\{1, \text{OUT}\})$$

  - They posed it as an open problem to remove the extra factor of IN.
- **The Result** [Deng, Lu, and Tao, PODS'23]:
  - An $\tilde{O}(\text{IN})$-space fully dynamic data structure that supports tuple insertions/deletions in $\tilde{O}(1)$ time and draws a sample in time:

$$\tilde{O}(\text{IN}^{\rho^*}/\max\{1, \text{OUT}\}) \quad \text{w.h.p.}$$

  - A justification for the $O(IN^{\rho^*})$ running time of the worst-case optimal join evaluation algorithms even when $OUT << \text{IN}^{\rho^*}$.

STATE OF THE ART VS. THE CONTRIBUTION

Let IN be the input size, OUT be the output size, and ρ^* be the fractional edge covering number.

- **State of the Art** [Chen and Yi, ICDT'20]:
  - After $\tilde{O}(\text{IN})$ preprocessing, a sample can be drawn in time:

$$\tilde{O}(\text{IN}^{\rho^*}+1 / \max\{1, \text{OUT}\})$$

  - They posed it as an open problem to remove the extra factor of IN.
- **The Result** [Deng, Lu, and Tao, PODS'23]:
  - An $\tilde{O}(\text{IN})$-space fully dynamic data structure that supports tuple insertions/deletions in $\tilde{O}(1)$ time and draws a sample in time:

$$\tilde{O}(\text{IN}^{\rho^*} / \max\{1, \text{OUT}\}) \quad \text{w.h.p.}$$

  - A justification for the $O(IN^{\rho^*})$ running time of the worst-case optimal join evaluation algorithms even when $OUT << \text{IN}^{\rho^*}$.

OUTLINE OF THE TALK

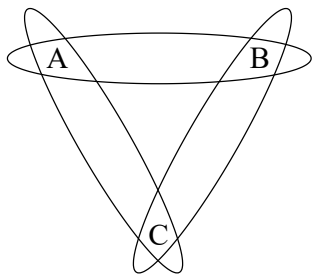
1. **Preliminaries:** The AGM Bound
   - A powerful tool for analyzing the maximum size of a join result.
2. **Core Technique:** The AGM Split Theorem
   - A geometric approach to recursively partition the data space.
3. **The Sampling Algorithm**
   - How to use the split theorem to build a fast sampling algorithm.
4. **Hardness Results & Conclusion**
   - Why the result is likely the best possible for combinatorial algorithms.

PRELIMINARIES: THE AGM BOUND

THE SCHEMA GRAPH AND FRACTIONAL EDGE COVERS

We can model a join query Q as a hypergraph $G = (\mathcal{X}, \mathcal{E})$.

- **Vertices** $\mathcal{X}$: The set of all attributes in the join (e.g., A, B, C).
- **Hyperedges** $\mathcal{E}$: The schema of each input relation (e.g., $\{A, B\}$, $\{B, C\}$, $\{C, A\}$).

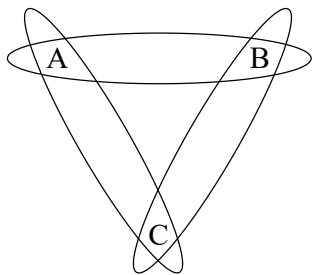


A **fractional edge cover** assigns a weight $W(e) \geq 0$ to each edge $e \in \mathcal{E}$ such that for every vertex X, the sum of weights of edges covering X is at least 1.

THE SCHEMA GRAPH AND FRACTIONAL EDGE COVERS

We can model a join query Q as a hypergraph $G = (\mathcal{X}, \mathcal{E})$.

- **Vertices** $\mathcal{X}$: The set of all attributes in the join (e.g., A, B, C).
- **Hyperedges** $\mathcal{E}$: The schema of each input relation (e.g., $\{A, B\}$, $\{B, C\}$, $\{C, A\}$).



A **fractional edge cover** assigns a weight $W(e) \geq 0$ to each edge $e \in \mathcal{E}$ such that for every vertex X, the sum of weights of edges covering X is at least 1.

THE AGM BOUND

Theorem (AGM Bound, Atserias et al. 2008)

For any fractional edge cover W, the size of the join result is bounded by:

$$|Join(Q)| \leq AGM_W(Q) := \prod_{e \in \mathcal{E}} |R_e|^{W(e)}$$

where R_e is the relation corresponding to edge e.

- This gives an instance-specific upper bound on the output size.
- To get a bound in terms of the total input size IN, we can minimize the total weight $\sum W(e)$.
- Let $\rho^* = \min_W \sum_{e \in \mathcal{E}} W(e)$ be the **fractional edge covering number**.

Corollary

$|Join(Q)| \leq IN^{\rho^*}$. (Remark: This bound is known to be tight in the worst case.)

THE AGM BOUND

Theorem (AGM Bound, Atserias et al. 2008)

For any fractional edge cover W, the size of the join result is bounded by:

$$|Join(Q)| \leq AGM_W(Q) := \prod_{e \in \mathcal{E}} |R_e|^{W(e)}$$

where R_e is the relation corresponding to edge e.

- This gives an instance-specific upper bound on the output size.
- To get a bound in terms of the total input size IN, we can minimize the total weight $\sum W(e)$.
- Let $\rho^* = \min_W \sum_{e \in \mathcal{E}} W(e)$ be the **fractional edge covering number**.

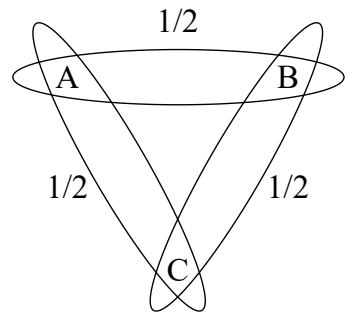
Corollary

$|Join(Q)| \leq IN^{\rho^*}$. *(Remark: This bound is known to be tight in the worst case.)*

EXAMPLE: AGM BOUND FOR A TRIANGLE JOIN

Consider the triangle join $R_{AB} \bowtie R_{BC} \bowtie R_{AC}$.

- We need to find weights w_{AB}, w_{BC}, w_{AC} such that:
  - For attribute A: $w_{AB} + w_{AC} \geq 1$
  - For attribute B: $w_{AB} + w_{BC} \geq 1$
  - For attribute C: $w_{BC} + w_{AC} \geq 1$
- To minimize the sum $w_{AB} + w_{BC} + w_{AC}$, we can set: $w_{AB} = w_{BC} = w_{AC} = 1/2$
- The fractional edge covering number is $\rho^* = 1/2 + 1/2 + 1/2 = 1.5$.
- The AGM bound is $|\text{Join}(Q)| \leq |R_{AB}|^{0.5} |R_{BC}|^{0.5} |R_{AC}|^{0.5}$.
- The worst-case output size is $\text{OUT} = O(\text{IN}^{1.5})$.



CORE TECHNIQUE: THE AGM SPLIT THEOREM

A GEOMETRIC VIEWPOINT & BOX-INDUCED SUB-JOINS

- We view each result tuple as a point in d-dimensional space $\mathbb{N}^d$.
- A **box** B is a d-dimensional rectangle that acts as a filter.
- The box induces a **sub-join** $Q(B)$ by keeping only the input tuples that fall within the box.

Example: Box-Induced Sub-Join

Consider a 3D box for attributes (A, B, C): $B = [0,1] \times [1,1] \times [1,2]$.

$R_{AB}(B)$

A	B
0	1
~~0~~	~~2~~
1	1
~~2~~	~~1~~

$R_{BC}(B)$

B	C
1	1
1	2
~~2~~	~~0~~
~~2~~	~~1~~

$R_{CA}(B)$

C	A
~~0~~	~~1~~
1	0
2	1
~~2~~	~~2~~

Join(Q(B))

A	B	C
0	1	1
~~0~~	~~2~~	~~1~~
1	1	2
~~2~~	~~1~~	~~2~~

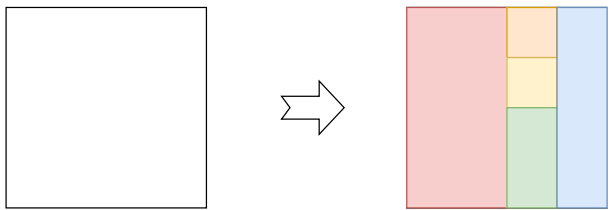
- We can still compute the AGM bound for sub-joins, which we denote as $\text{AGM}_W(B) := \text{AGM}_W(Q(B))$.

THE AGM SPLIT THEOREM

Theorem (AGM Split Theorem)

Fix an arbitrary fractional edge cover W, and assume the availability of count and median oracles. Given any box B with $AGM_W(B) \geq 2$, we can find in $\tilde{O}(1)$ time a set $\mathcal{C}$ of at most $2d + 1$ smaller boxes having these nice properties:

1. ***Partition:*** *The boxes in $\mathcal{C}$ are disjoint and their union is B. Formally, $B = \bigsqcup_{B' \in \mathcal{C}} B'$.*
2. ***Shrink:*** *For each $B' \in \mathcal{C}$, $AGM_W(B') \leq \frac{1}{2} AGM_W(B)$.*
3. ***Conserve:*** *$\sum_{B' \in \mathcal{C}} AGM_W(B') \leq AGM_W(B)$.*



THE SPLITTING STRATEGY

The split proceeds recursively, one dimension at a time, based on a lemma guaranteeing that any partition created by slicing a box along a single axis satisfies the ‘Conserve’ property (3).

- **Step 1: Find a split point.** Find the z_1 in the domain of X_1 that creates a "balanced" partition.
  - We choose the smallest z_1 such that the "left" part of the box has an AGM bound $\leq \frac{1}{2}\mathrm{AGM}_W(B)$. As a result, the "right" part also has an AGM bound $\leq \frac{1}{2}\mathrm{AGM}_W(B)$.
- **Step 2: Partition and Recurse.** This splits B into three pieces:
  - B_{left} and B_{right}: These are guaranteed to be "small" and are added to $\mathcal{C}$.
  - B_{mid}: A thin "slice" where the first attribute is fixed to z_1. We recursively apply the same process to this slice using the next attribute, X_2.
- This process is very fast, taking $\tilde{O}(d) = \tilde{O}(1)$ time per split.

THE SPLITTING STRATEGY

The split proceeds recursively, one dimension at a time, based on a lemma guaranteeing that any partition created by slicing a box along a single axis satisfies the ‘Conserve’ property (3).

- **Step 1: Find a split point.** Find the z_1 in the domain of X_1 that creates a "balanced" partition.
  - We choose the smallest z_1 such that the "left" part of the box has an AGM bound $\leq \frac{1}{2}\text{AGM}_W(B)$. As a result, the "right" part also has an AGM bound $\leq \frac{1}{2}\text{AGM}_W(B)$.
- **Step 2: Partition and Recurse.** This splits B into three pieces:
  - B_{left} and B_{right}: These are guaranteed to be "small" and are added to $\mathcal{C}$.
  - B_{mid}: A thin "slice" where the first attribute is fixed to z_1. We recursively apply the same process to this slice using the next attribute, X_2.
- This process is very fast, taking $\tilde{O}(d) = \tilde{O}(1)$ time per split.

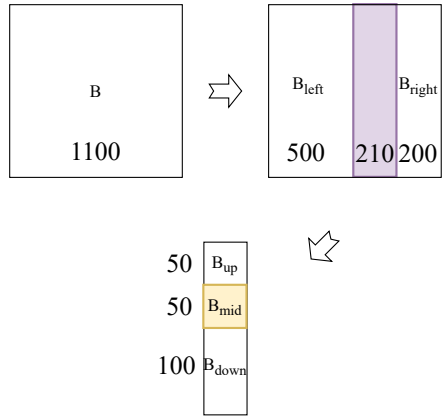
THE SPLITTING STRATEGY

The split proceeds recursively, one dimension at a time, based on a lemma guaranteeing that any partition created by slicing a box along a single axis satisfies the ‘Conserve’ property (3).

- **Step 1: Find a split point.** Find the z_1 in the domain of X_1 that creates a "balanced" partition.
  - We choose the smallest z_1 such that the "left" part of the box has an AGM bound $\leq \frac{1}{2}\text{AGM}_W(B)$. As a result, the "right" part also has an AGM bound $\leq \frac{1}{2}\text{AGM}_W(B)$.
- **Step 2: Partition and Recurse.** This splits B into three pieces:
  - B_{left} and B_{right}: These are guaranteed to be "small" and are added to $\mathcal{C}$.
  - B_{mid}: A thin "slice" where the first attribute is fixed to z_1. We recursively apply the same process to this slice using the next attribute, X_2.
- This process is very fast, taking $\tilde{O}(d) = \tilde{O}(1)$ time per split.

VISUALIZING THE SPLIT (2D EXAMPLE)

1. **Start with Box B.**
2. **Split on X_1 at z_1.**
   - B_{left} and B_{right} are final.
   - B_{mid} is the puple slice.
3. **Recurse on B_{mid}.**
4. **Split B_{mid} on X_2 at z_2.**
   - This creates B_{up}, B_{down}, and the final point B_{mid}.



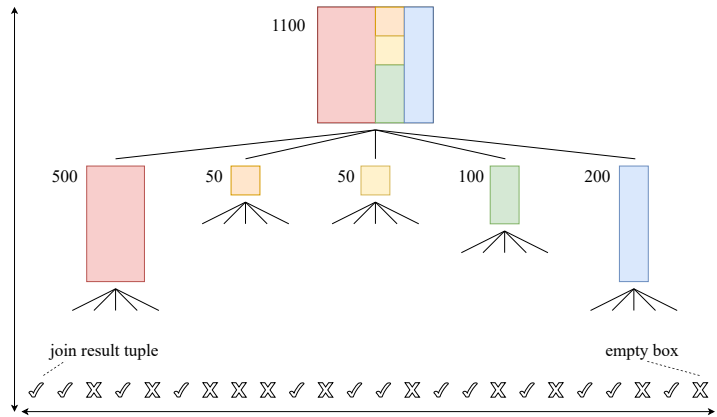
The final set of children for B would be $\{B_{left}, B_{right}, B_{up}, B_{down}, B_{mid}\}$.

THE SAMPLING ALGORITHM

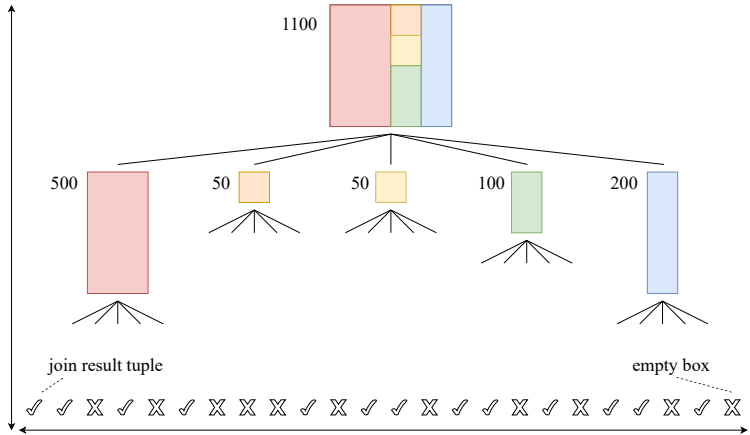
THE CONCEPTUAL AGM SPLIT TREE

We can use the AGM Split Theorem to get a conceptual decomposition tree.

- **Root:** The entire attribute space $\mathbb{N}^d$.
- **Children:** An internal box B is split into its children using the theorem.
- **Leaves:** Boxes with an AGM bound less than 2.



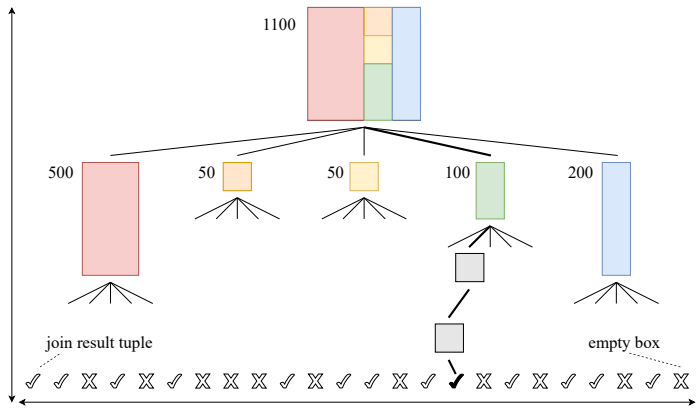
THE CONCEPTUAL AGM SPLIT TREE (CONT.)



- The height of this tree is $O(\log \mathrm{IN})$ because the AGM bound is halved at each level.
- We cannot afford to build this tree explicitly; it could have $\Omega(\mathrm{IN}^{\rho^*})$ leaves.

THE ALGORITHM IDEA: A RANDOM WALK ON A CONCEPTUAL TREE

- The AGM Split Theorem defines a conceptual AGM Split Tree.
- We cannot build this tree explicitly; it could have $\Omega(\text{IN}^{\rho^*})$ leaves.
- **Solution:** We don't build the tree. Instead, we generate a *single random path* on the fly.



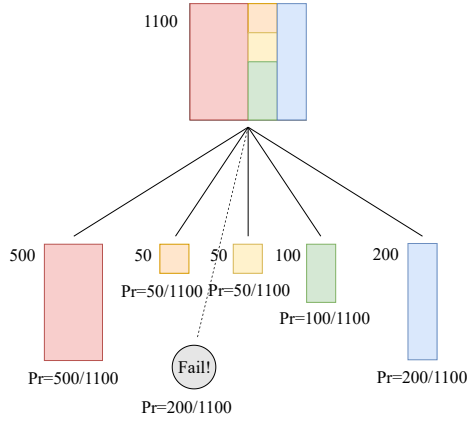
STEP 1: TRAVERSING THE TREE

At each internal node, we perform a weighted random choice to pick the next box in the path.

1. Split the current box B into children $\{B'_1, B'_2, \dots\}$.
2. Assign each child B'_j a probability:

$$\Pr[\text{choose } B'_j] = \frac{\mathrm{AGM}_W(B'_j)}{\mathrm{AGM}_W(B)}$$

3. Randomly select one child based on these weights to continue the walk.
4. There's a chance of "failure" if $\sum \mathrm{AGM}_W(B'_j) < \mathrm{AGM}_W(B)$.

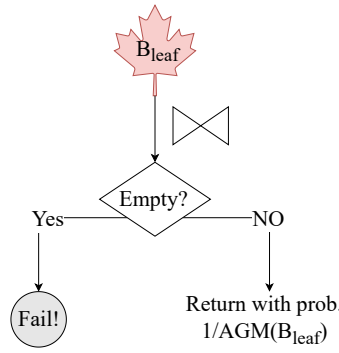


This process repeats until a leaf node is reached. Each step takes $\tilde{O}(1)$ time.

STEP 2: ARRIVING AT A LEAF

A leaf is a box B_{leaf} with $\text{AGM}_W(B_{leaf}) < 2$. It contains at most one join result tuple.

1. Compute the sub-join for B_{leaf}.
2. If it's empty, the attempt fails.
3. If it contains a single tuple u:
   - We perform one final probabilistic check.
   - Return u with probability $1/\text{AGM}_W(B_{leaf})$.
   - Otherwise, the attempt fails.



This final step ensures that every tuple in the entire join result has the exact same probability of being selected.

ANALYSIS OF THE ALGORITHM

- **Correctness:** The probability of the random walk reaching a specific leaf box B_{leaf} is $\text{AGM}_W(B_{leaf})/\text{AGM}_W(Q)$. The final step cancels out the $\text{AGM}_W(B_{leaf})$ term.
  - This means any given result tuple u is returned with probability exactly $1/\text{AGM}_W(Q)$. All result tuples are equally likely!
- **Success Probability:** The total probability of returning *any* tuple is:

$$\sum_{u \in \text{Join}(Q)} \Pr[\text{return } u] = \sum_{u \in \text{Join}(Q)} \frac{1}{\text{AGM}_W(Q)} = \frac{\text{OUT}}{\text{AGM}_W(Q)}$$

- **Running Time:**
  - A single run of the algorithm takes $\tilde{O}(1)$ time.
  - We expect to run it $\text{AGM}_W(Q)/\text{OUT}$ times to get one sample.
  - By choosing the optimal W, the total time to get a sample is $\tilde{O}(\text{IN}^{\rho^*}/\text{OUT})$ w.h.p.

HARDNESS RESULTS & CONCLUSION

HOW GOOD IS THE RESULT?

We achieved a sampling time of $\tilde{O}(\text{IN}^{\rho^*}/\text{OUT})$.

- When $\text{OUT} = \Omega(\text{IN}^{\rho^*})$, the time is $\tilde{O}(1)$, which is clearly optimal.
- But what if OUT is very small, e.g., OUT = 1? The time is $\tilde{O}(\text{IN}^{\rho^*})$. Can we do better?

The Hardness Question

Can a *combinatorial* algorithm achieve sampling time $\tilde{O}(\text{IN}^{\rho^*-\epsilon}/\text{OUT})$ for some constant $\epsilon > 0$, at least when $1 \leq \text{OUT} \leq \text{IN}^{\epsilon}$?

- A combinatorial algorithm is one that does not use techniques like fast matrix multiplication. All known practical join algorithms are combinatorial.

HOW GOOD IS THE RESULT?

We achieved a sampling time of $\tilde{O}(\text{IN}^{\rho^*}/\text{OUT})$.

- When $\text{OUT} = \Omega(\text{IN}^{\rho^*})$, the time is $\tilde{O}(1)$, which is clearly optimal.
- But what if OUT is very small, e.g., OUT = 1? The time is $\tilde{O}(\text{IN}^{\rho^*})$. Can we do better?

The Hardness Question

Can a *combinatorial* algorithm achieve sampling time $\tilde{O}(\text{IN}^{\rho^*-\epsilon}/\text{OUT})$ for some constant $\epsilon > 0$, at least when $1 \leq \text{OUT} \leq \text{IN}^{\epsilon}$?

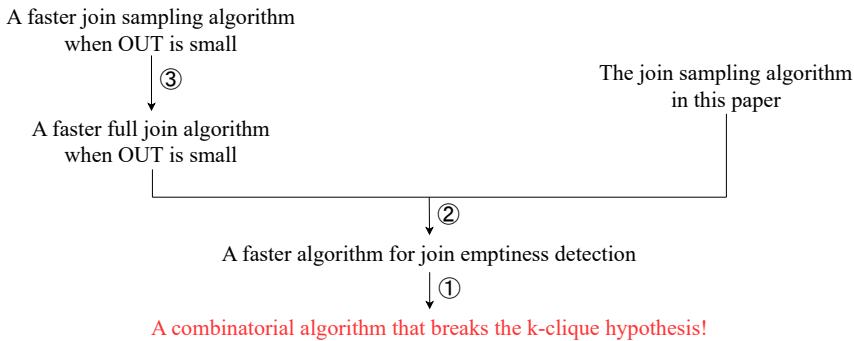
- A combinatorial algorithm is one that does not use techniques like fast matrix multiplication. All known practical join algorithms are combinatorial.

CONNECTION TO THE COMBINATORIAL K-CLIQUE HYPOTHESIS

Definition (Combinatorial k-Clique Hypothesis)

No combinatorial algorithm can detect if an n-vertex graph has a k-clique in $O(n^{k-\epsilon})$ time for any constant $\epsilon > 0$. This is a widely believed conjecture in fine-grained complexity.

We show a chain of reductions:



CONCLUSION

- We presented a dynamic data structure for the join sampling problem.
  - Achieves a sampling time of $\tilde{O}(\mathrm{IN}^{\rho^*}/\mathrm{OUT})$ w.h.p.
  - Supports dynamic updates in $\tilde{O}(1)$ time.
- The core technical contribution is the **AGM Split Theorem**, which enables a geometric, recursive decomposition of the problem space.
- We established a **hardness result**, showing that the sampling algorithm and the worst-case optimal join algorithm are likely the best possible for the class of combinatorial algorithms, by linking further improvements to the combinatorial k-clique hypothesis.
- This work not only solves a long-standing open problem but also deepens our understanding of the fundamental limits of join computation.

Thank you!