Fast Join Project Query Evaluation using Matrix Multiplication

Minsi Lu

Week12: Nov 17, 2025

Introduction | Why do we care about Join-Project queries?

- There are many efficient algorithms that achieve worst-case optimal runtime for full join queries
- However, there is not much support for queries involve projections
  - Standard approach: Compute full join → Project → Deduplicate
  - Inefficient if the Full join results much larger than final result
  - Critical since many tasks can be formulated as Join-Project queries

Introduction | Example 1

- Consider relation R(x,y) of size N denotes that x and y are friends.
- We wish to enumerate all users pairs who have at least one friend in common
- This task can be captured by the query $Q^{¨}(x, z) = R(x,y), R(z,y)$

```
SELECT DISTINCT R1.x, R2.x
FROM R1 as R, R2 as R
WHERE R1.y = R2.y
```

- Suppose that the graph contains a constant number of communities and the users are spread evenly across them, Each community has $O(\sqrt{N})$ users
- Full join ouput: $\Theta(N^{3/2})$, Final output: $\Theta(N)$

Introduction | Other Applications

- Set Similarity Join (Entity Matching, Recommender Systems)
  - Find all set pairs with ≥c common elements
  - Previous best: $O(|D|^{2-1/c} \cdot |OUT|^{1/2c})$, near to $O(|D|^2)$ as the c increase
- Set Containment Join
  - Find all pairs where one set contains another
- Both tasks can be answered using the query

```
SELECT R1.x, R2.x count(*)
FROM R1 as R, R2 as R
WHERE R1.y = R2.y
GROUP BY R1.x, R2.x
```

Introduction | Other Applications

- Graph Analytics(Collaborative Networks)
  - Co-authorship graphs:V(x,y) = R(x,p), R(y,p)
  - Batched boolean queries: "Do authors a_1 and a_2 share papers? "

Problem Setting | Definitions

- 2-path query

$$\ddot{Q}(x, z) = R(x, y), S(z, y)$$

- Star Join

$$Q_k^{\star}(x_1, x_2, \ldots, x_k) = R_1(x_1, y), R_2(x_2, y), \ldots, R_k(x_k, y)$$

- Key Metrics
  - |D| = max(|R|, |S|) - input size
  - |OUT| - projected output size
  - |OUT⋈| - full join size (before projection)

Problem Setting | Definitions

- Set Similarity (SSJ): given two families of sets R(x,y) and S (z,y). R(x,y) means set x contains element y, S (z,y) means set z contains element y, c ≥ 1.

$$\{(a, b) \mid |\pi_y(\sigma_{x=a}(R)) \cap \pi_y(\sigma_{z=b}(S))| \geq c\}$$

When c = 1, SSJ = 2-path query

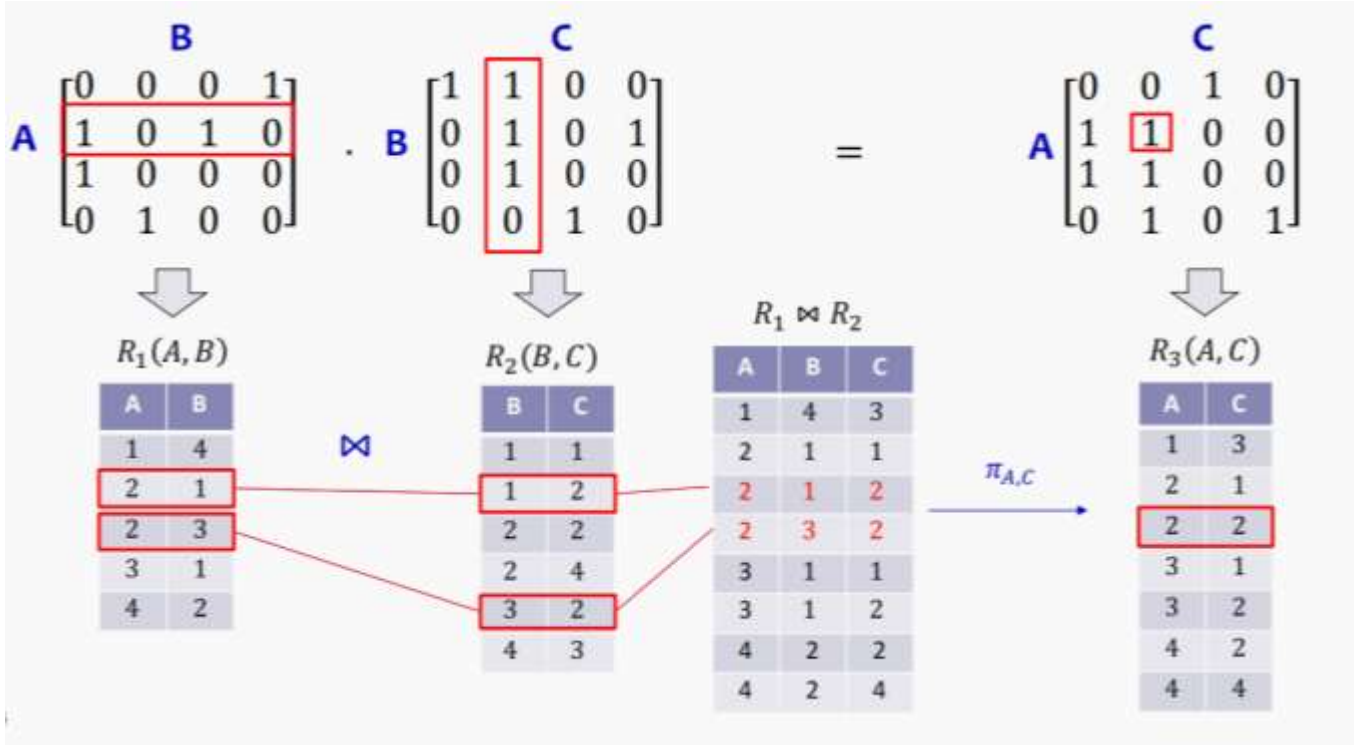
- Set Containment (SCJ):

$$\{(a, b) \mid \pi_y(\sigma_{x=a}(R)) \subseteq \pi_y(\sigma_{z=b}(S))\}$$

- Complexity Model:
  - Uniform-cost RAM model: Data values and pointers are constant size
  - Data complexity: Query is fixed, complexity measured in database size

Problem Setting | Matrix Multiplication

- view the 2-path query as a matrix computation over the boolean field



Problem Setting | Matrix Multiplication

- Complexity
  - For matrices U×V and V×W:

$$O(UVW\beta^{\omega-3})$$

    where β = min{U, V, W}
  - Special case (ω = 2):

$$O(UVW/\beta)$$

Problem Setting | The Baseline

- Worst-Case Optimal Joins (WCOJ)

- Any CQ Q with fractional edge cover ρ^* can be computed in $O(|D|^{\rho^*})$
  - For star join Q_k^*: $O(|D|^k)$
  - Problem: Oblivious to actual |OUT|!

- Combinatorial Improvement (Amossen & Pagh 2009)
  - Q^k computable in $O(|D| \cdot |OUT|^{1-1/k})$ For star join Q_k^* $O(|D|^k)$
  - For k=2: $O(|D| \cdot |OUT|^{1/2})$

Problem Setting | The Baseline

- Worst-Case Optimal Joins (WCOJ)

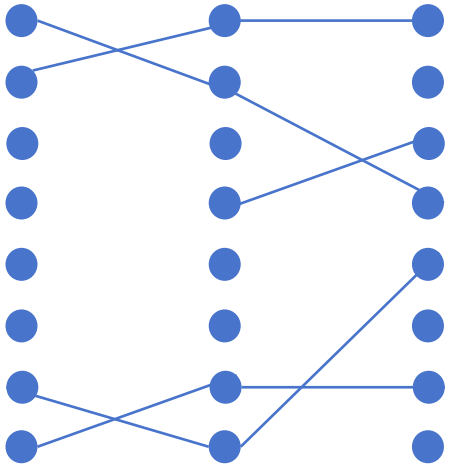
- Any CQ Q with fractional edge cover ρ^* can be computed in $O(|D|^{\rho^*})$
  - For star join Q_k^*: $O(|D|^k)$
  - Problem: Oblivious to actual |OUT|!

- Combinatorial Improvement (Amossen & Pagh 2009)
  - Q^k computable in $O(|D| \cdot |OUT|^{1-1/k})$ For star join Q_k^* $O(|D|^k)$
  - For k=2: $O(|D| \cdot |OUT|^{1/2})$

Join-Project | Key Ideas

When does WCOJ and matrix multiplication work well?

R_1

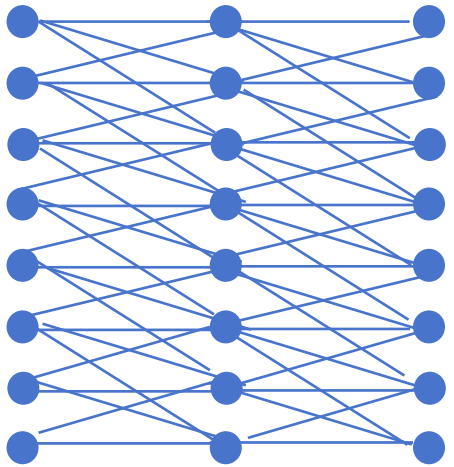


Graph vertices have low degree(i.e sparse)

WCOJ good

MM bad

R_2



Graph vertices have large degree(i.e dense)

WCOJ bad

MM good

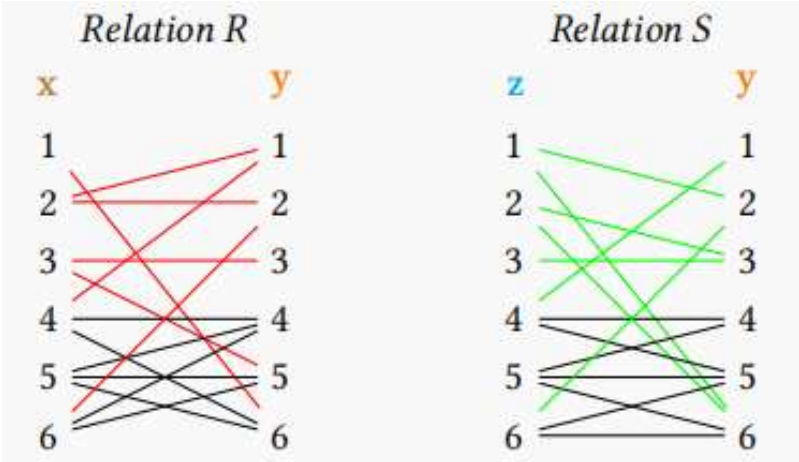
Join-Project | Assumptions

- know the output size |OUT| (drop this assumption later)
- removed any tuples that do not contribute to the query result (linear time)
- In 2 path query, assume that $N_S \leq N_R$

Join-Project | The 2-Path Query Algorithm

- Partition the relations based on degree of vertices to achieve best of both worlds

$$R^- = \{R(a,b) \mid |\sigma_{x=a}R(x,y)| \leq \Delta_2 \text{ or } |\sigma_{y=b}S(z,y)| \leq \Delta_1\}$$
$$S^- = \{S(c,b) \mid |\sigma_{z=c}S(z,y)| \leq \Delta_2 \text{ or } |\sigma_{y=b}S(z,y)| \leq \Delta_1\}$$



R- R+

S- S+

$(R^- \bowtie S) \cup (R \bowtie S^-)$, WOCJ
$R^+ \bowtie S^+$, MM

Join-Project | The 2-Path Query Algorithm

Algorithm 1: Computing $\pi_{xz} R(x,y) \bowtie S(z,y)$

1 $R^- \leftarrow \{R(a,b) \mid |\sigma_{x=a} R(x,y)| \leq \Delta_2$ or $|\sigma_{y=b} S(z,y)| \leq \Delta_1\}$, $R^+ \leftarrow R \setminus R^-$

2 $S^- \leftarrow \{S(c,b) \mid |\sigma_{z=c} S(z,y)| \leq \Delta_2$ or $|\sigma_{y=b} S(z,y)| \leq \Delta_1\}$, $S^+ \leftarrow S \setminus S^+$

3 $T \leftarrow (R^- \bowtie S) \cup (R \bowtie S^-)$ /* use wcoj */

4 $M_1(x,y) \leftarrow R^+$ adj matrix, $M_2(y,z) \leftarrow S^+$ adj matrix

5 $M \leftarrow M_1 \times M_2$ /* matrix multiplication */

6 $T \leftarrow T \cup \{(a,c) \mid M_{ac} > 0\}$

7 **return** T

WOCJ for sparse component:

- $|OUT\bowtie|$ bounded by $N_s \cdot \Delta_1 + |OUT| \cdot \Delta_2$
  - Why?
  - Each light y-value contributes $\leq \Delta_1$ tuples
  - Each output tuple has x-degree $\leq \Delta_2$

- Runtime:
  - $O(N_R + N_s + |OUT\bowtie|)$
  - $= O(N_R + N_s \cdot \Delta_1 + |OUT| \cdot \Delta_2)$

Join-Project | The 2-Path Query Algorithm

Algorithm 1: Computing $\pi_{xz} R(x,y) \bowtie S(z,y)$

1 $R^- \leftarrow \{R(a,b) \mid |\sigma_{x=a} R(x,y)| \leq \Delta_2$ or $|\sigma_{y=b} S(z,y)| \leq \Delta_1\}$, $R^+ \leftarrow R \setminus R^-$

2 $S^- \leftarrow \{S(c,b) \mid |\sigma_{z=c} S(z,y)| \leq \Delta_2$ or $|\sigma_{y=b} S(z,y)| \leq \Delta_1\}$, $S^+ \leftarrow S \setminus S^+$

3 $T \leftarrow (R^- \bowtie S) \cup (R \bowtie S^-)$ /* use wcoj */

4 $M_1(x,y) \leftarrow R^+$ adj matrix, $M_2(y,z) \leftarrow S^+$ adj matrix

5 $M \leftarrow M_1 \times M_2$ /* matrix multiplication */

6 $T \leftarrow T \cup \{(a,c) \mid M_{ac} > 0\}$

7 **return** T

MM for dense component:

- M_1: $N_R/\Delta_2 \times N_s/\Delta_1$ (for R^+)
- M_2: $N_s/\Delta_1 \times N_s/\Delta_2$ (for S^+)
- Cost: $M(N_R/\Delta_2, N_s/\Delta_1, N_s/\Delta_2)$

Join-Project | The 2-Path Query Algorithm

Total Cost

$$N_R + N_S\,\Delta_1 + |\mathsf{OUT}|\Delta_2 + M\Big(\frac{N_R}{\Delta_2}, \frac{N_S}{\Delta_1}, \frac{N_S}{\Delta_2}\Big) + C$$

For $\omega = 2$ (theoretical optimal MM), assume $N_R = N_s = N$

$$f(\Delta_1, \Delta_2) = N + N \cdot \Delta_1 + |\mathsf{OUT}| \cdot \Delta_2 + \frac{N^2}{\Delta_2 \min\{\Delta_1, \Delta_2\}}$$

while ensuring $1 \le \Delta 1, \Delta 2 \le N$.

If $\Delta 1 > \Delta 2$, we can always improve the solution by decreasing the value of $\Delta 1$ to $\Delta 2$. So impose the constraint $1 \le \Delta 1 \le \Delta 2 \le N$

Join-Project | The 2-Path Query Algorithm

- Case 1: $|OUT| \leq N$
  - Optimal: $\Delta_1 = |OUT|^{1/3}$, $\Delta_2 = N/|OUT|^{2/3}$
  - Runtime: $O(N + N \cdot |OUT|^{1/3})$
- Case 2: $|OUT| > N$
  - Optimal: $\Delta_1 = \Delta_2 = (2N^2/(N + |OUT|))^{1/3}$
  - Runtime: $O(N^{2/3} \cdot |OUT|^{2/3})$
- Lemma 3. Assuming that the exponent in matrix multiplication is $\omega = 2$, the query $Q^{\cdot\cdot}$ can be computed in time

$$O(|D| + |D|^{2/3} \cdot |OUT|^{1/3} \cdot \max\{|D|, |OUT|\}^{1/3})$$

- Compare to baseline: $O(|D| \cdot |OUT|^{1/2})$

Join-Project | The 2-Path Query Algorithm Optimization

- Two Key issue:
  - How to estimate |OUT|
    - $|dom(x)| \leq |OUT| \leq \min\{|dom(x)|^2, |OUT\bowtie|\}$
    - $|OUT\bowtie| \leq N \cdot \sqrt{|OUT|}$ -> $OUT >= (|OUT\bowtie|/N)^2$
    - Paper suggests: geometric mean of $\max\{|dom(x)|, (|OUT\bowtie|/N)^2\}$ and $\min\{|dom(x)|^2, |OUT\bowtie|\}$
  - How to estimate MM cost

if $|OUT_{\bowtie}| \leq 20 \cdot N$ **then**
  use worst-case optimal join algorithm

Rationale:
- MM has overhead (matrix construction, memory allocation)
- Only beneficial when avoiding large intermediate results
- Threshold of 20× is empirically determined

Join-Project | The 2-Path Query Algorithm Optimization

- How to estimate MM cost

Algorithm 3: Cost Based Optimizer

Output: degree threshold Δ_1, Δ_2

1 *Estimate full join result* $|\mathsf{OUT}_{\bowtie}|$ *and* $|\mathsf{OUT}|$
2 **if** $|\mathsf{OUT}_{\bowtie}| \leq 20 \cdot N$ **then**
3 | *use worst-case optimal join algorithm*
4 $\mathsf{t}_{\mathsf{light}} \leftarrow |\mathsf{OUT}_{\bowtie}|, \mathsf{t}_{\mathsf{heavy}} \leftarrow 0, \mathsf{prev}_{\mathsf{light}} \leftarrow \infty, \mathsf{prev}_{\mathsf{heavy}} \leftarrow 0, \Delta_1 = N$
5 **while** true **do**
6 | $\mathsf{prev}_{\mathsf{light}} \leftarrow \mathsf{t}_{\mathsf{light}}, \mathsf{prev}_{\mathsf{heavy}} \leftarrow \mathsf{t}_{\mathsf{heavy}}$
7 | $\mathsf{prev}_{\Delta_1} \leftarrow \Delta_1, \mathsf{prev}_{\Delta_2} \leftarrow \Delta_2$
8 | $\Delta_1 \leftarrow (1-\epsilon)\Delta_1, \Delta_2 \leftarrow N \cdot \Delta_1 / |\mathsf{OUT}|$
9 | $\mathsf{t}_{\mathsf{light}} \leftarrow T_I \cdot \mathsf{sum}(y_{\Delta_1}) + \cdot T_I \cdot \mathsf{sum}(x_{\Delta_2}) +$
10 | $T_m \cdot |\mathbf{dom}(x)| + T_s \cdot \mathsf{cdfx}(y_{\Delta_1}) \cdot |\mathbf{dom}(x)|$
11 | $u, v, w \leftarrow$ #heavy x, y, z values using $\mathsf{count}(w_\delta)$
12 | $\mathsf{t}_{\mathsf{heavy}} \leftarrow \hat{M}(u, v, w, co) + T_m \cdot (u \cdot v + u \cdot w)$
13 | **if** $\mathsf{prev}_{\mathsf{light}} + \mathsf{prev}_{\mathsf{heavy}} \leq \mathsf{t}_{\mathsf{light}} + \mathsf{t}_{\mathsf{heavy}}$ **then**
14 | | **return** $\mathsf{prev}_{\Delta_1}, \mathsf{prev}_{\Delta_2}$

Symbol	Description
T_s	avg time for sequential access
T_m	avg time for allocating 32 bytes of memory
co	number of cores available
$\hat{M}(u, v, w, co)$	estimate of time required to multiply matrices of dimension $u \times v$ and $v \times w$ using co cores
T_I	avg time for random access and insert

Join-Project | The 2-Path Query Algorithm Optimization

- How to estimate MM cost

Challenge: M̃(u,v,w,co) is hardware and library dependent!
Solution: Precomputed Lookup Table
1.Offline phase: Benchmark M̃(p,p,p,co) for:
   1. p ∈ {1000, 2000, 3000, ..., 20000}
   2. co ∈ {1, 2, 3, 4, 5}

2.Online phase: Given arbitrary (u,v,w,co):
   1. Find nearest p in table
   2. Extrapolate using scaling rules
   3. Example: M̃(1500, 8000, 1500, 4) ≈ interpolate from nearby values

Why this works:
•MM performance scales predictably within ranges
•Don't need exhaustive table (memory efficient)
•Captures hardware-specific optimizations (AVX, SIMD, cache effects)

Join-Project | Start Query

- Partition

$$R_i^- = \{R_i(a,b) \mid |\sigma_{x_i=a} R_i(x_i,y)| \le \Delta_2\}$$
$$R_i^\diamond = \{R_i(a,b) \mid |\sigma_{y=b} R_j(x_j,y)| \le \Delta_1, \text{ for each } j \in [k] \setminus i\}$$
$$R_i^+ = R_i \setminus (R_i^- \cup R_i^\diamond)$$

Light x_i values

y-values light in ALL other relations

Heavy values only

- Algorithm:
  - 1. Compute joins with R^-_j using WCOJ (for each j), $O(|OUT| \cdot \Delta 2)$
  - 2. Compute joins with $R\diamond_j$ using WCOJ (for each j), $O(N \cdot \Delta_1^{k-1})$
  - 3. Matrix multiplication for R^+_1, ..., R^+_k

Join-Project | Start Query

Star Join Matrix Construction

For k relations, create two matrices:

Matrix V: $(N/\Delta_2)^{\lceil k/2 \rceil} \times N/\Delta_1$

$$V_{(a_1, a_2, \ldots a_{\lceil k/2 \rceil}), b} = \begin{cases} 1, & (a_1, b) \in R_1, \ldots, (a_{\lceil k/2 \rceil}, b) \in R_{\lceil k/2 \rceil} \\ 0, & \text{otherwise} \end{cases}$$

Matrix W: $(N/\Delta_2)^{\lfloor k/2 \rfloor} \times N/\Delta_1$

$$W_{(a_{\lceil k/2 \rceil + 1} \ldots a_k), b} = \begin{cases} 1, & (a_{\lceil k/2 \rceil + 1}, b) \in R_{\lceil k/2 \rceil + 1}, \ldots, (a_k, b) \in R_k \\ 0, & \text{otherwise} \end{cases}$$

Total Cost:

$$N \cdot \Delta_1^{k-1} + |\mathsf{OUT}| \cdot \Delta_2 + M\left(\left(\frac{N}{\Delta_2}\right)^{\lceil k/2 \rceil}, \frac{N}{\Delta_1}, \left(\frac{N}{\Delta_2}\right)^{\lfloor k/2 \rfloor}\right)$$

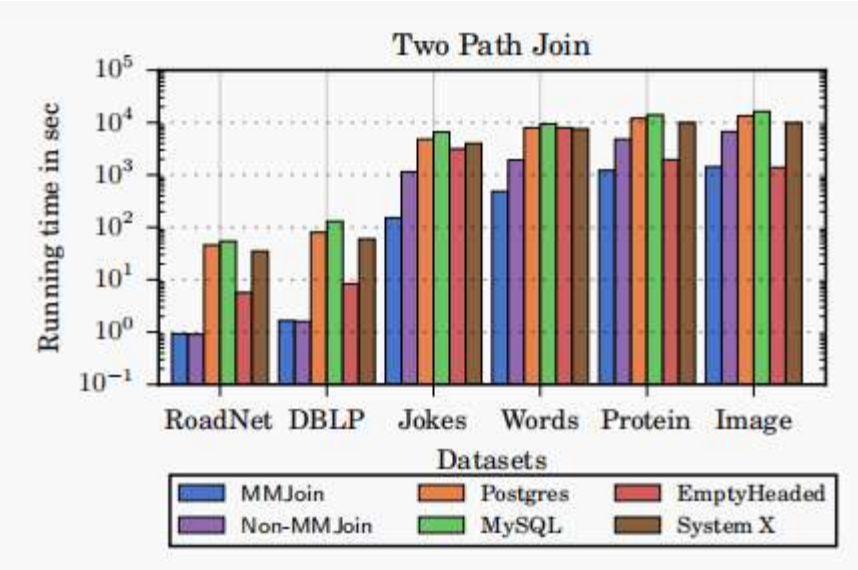
Experiments

- on 150GB RAM, Intel Xeon CPU E5 with 20 cores
  - use AVX and OpenMp for vectorized, multicore execution support
  - Intel MKL is the underiying linear algebra library
  - Use 6 datasets with different characteristics

Dataset	$\|R\|$	No. of sets	$\|\mathbf{dom}\|$	Avg set size	Min set size	Max set size
DBLP	10M	1.5M	3M	6.6	1	500
RoadNet	1.5M	1M	1M	1.5	1	20
Jokes	400M	70K	50K	5.7K	130	10K
Words	500M	1M	150K	500	1	10K
Protein	900M	60K	60K	15K	50	50K
Image	800M	70K	50K	11.4K	10K	50K

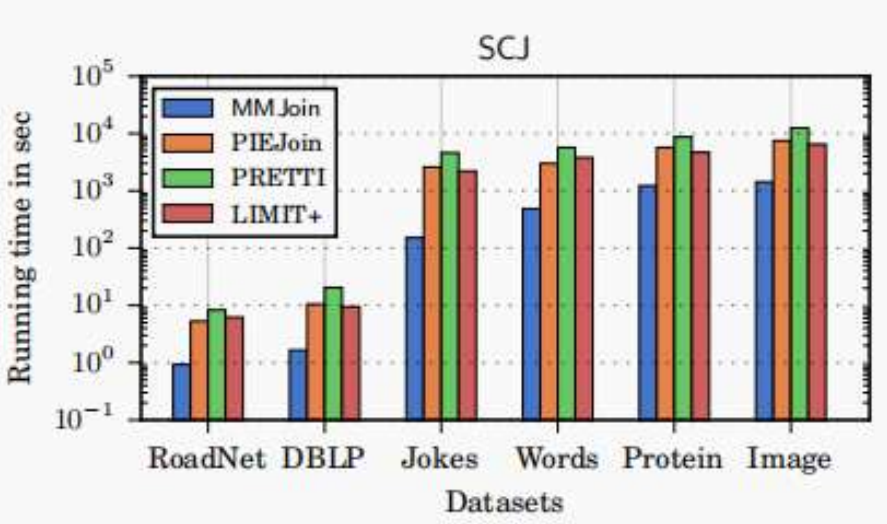
Table 2: Dataset Characteristics

Experiments



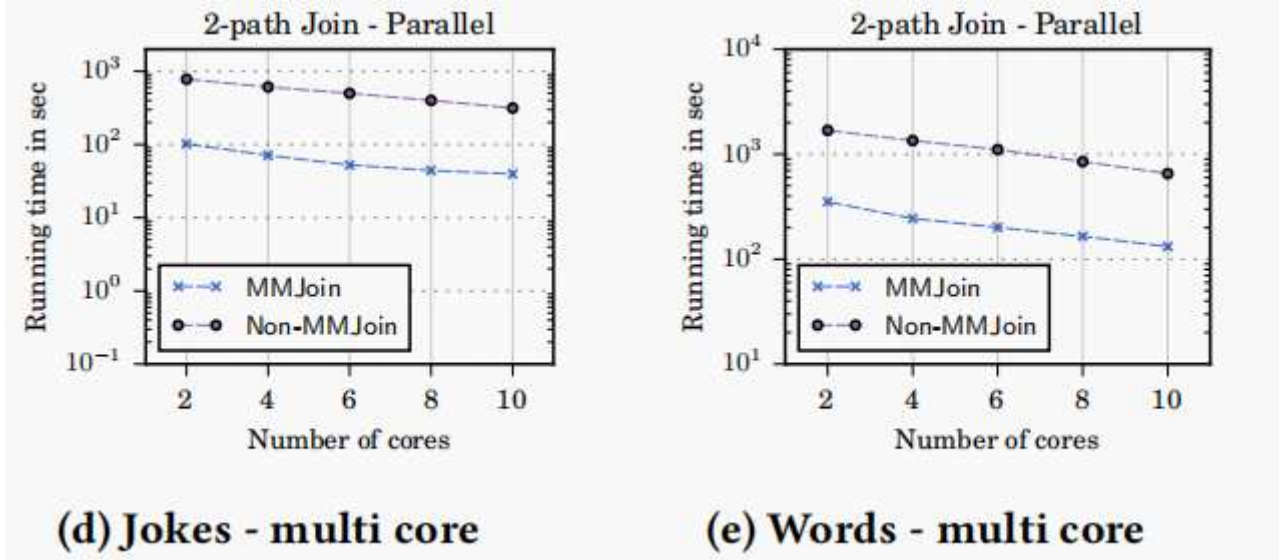
- MMjoin is up to two orders of magnitude Faster
- Matrix multiplication avoids materializing large intermediate results
- Intel MKL is highly optimized for vectorized execution

Experiments



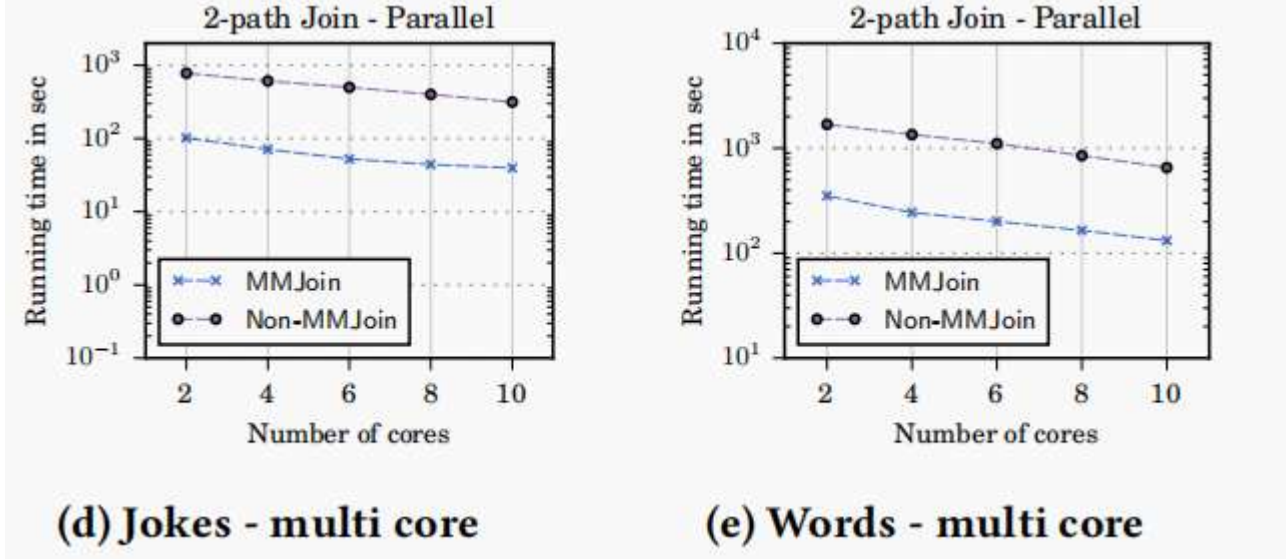
- MMjoin is s faster that SCJ algorithrms on dense datasets
- Matrix multiplication allows for coordination-free parallelization

Experiments



(d) Jokes - multi core

(e) Words - multi core

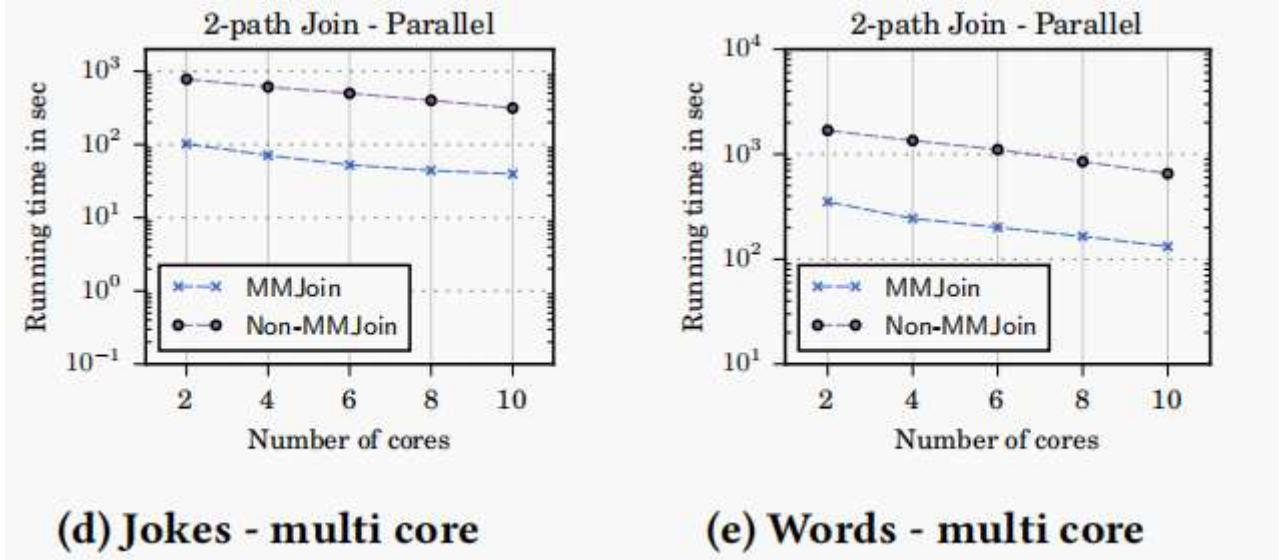


(d) Jokes - multi core

(e) Words - multi core

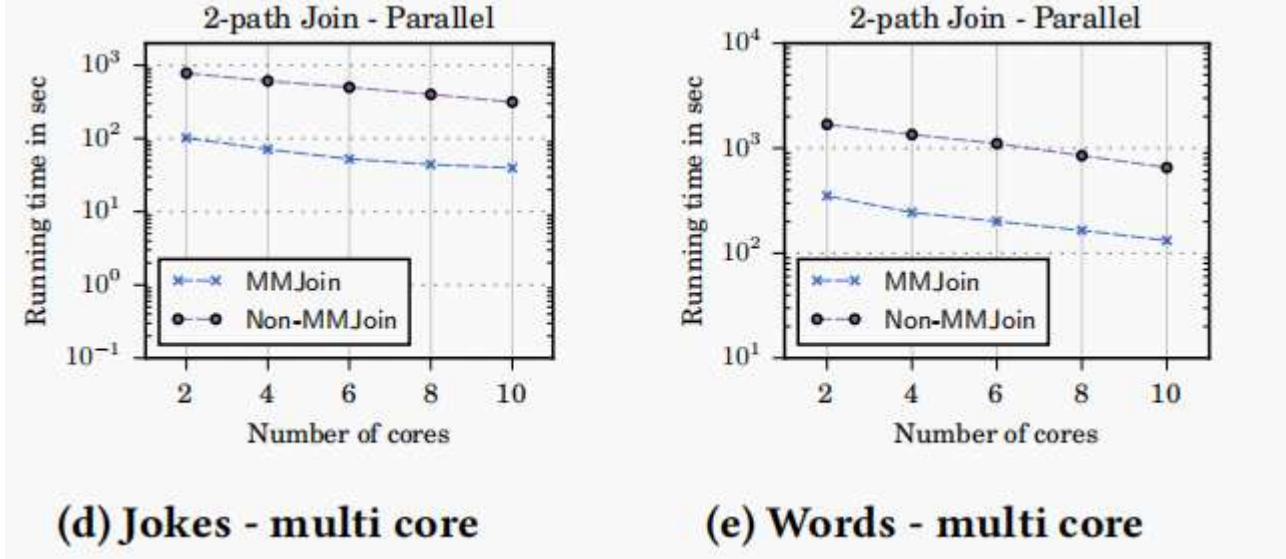
- SSJ

Experiments



(d) Jokes - multi core

(e) Words - multi core



(d) Jokes - multi core

(e) Words - multi core

- SSJ

Reference

- Deep, Shaleen, Xiao Hu, and Paraschos Koutris. "Fast join project query evaluation using matrix multiplication." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020.
- Amossen, Rasmus Resen, and Rasmus Pagh. "Faster join-projects and sparse matrix multiplications." Proceedings of the 12th International Conference on Database Theory. 2009.
- Xiao Hu. "Output-Optimal Algorithms for Join-Aggregate Queries." CS848: Advanced Database Systems, University of Waterloo, Lecture slides, Sep 22, 2025.

Thank you!