Join Algorithms - New Developments

- New breed of join algorithms
  - NPRJ, MineSweeper, Leapfrog Triejoin (LFTJ)
- Worst-case optimal join (Ngo, PODS 12), meet the AGM bound
  - Example: \( R(A,B) \bowtie S(B,C) \Rightarrow T(A,C) \)
  - Pairwise: \( O(N^3) \), AGM: \( O(N^{3/2}) \)
- \( N \)=#tuples

Significant improvements over traditional approaches

Leapfrog Triejoin [Veldhuizen, ICDT'14]

- Relations as tries
- Sequentially match variables
- Variable order can greatly affect the performance
- Small memory footprint – no intermediate results are constructed

Reusing Intermediate Results

- **Problem:** Recurring joins repeatedly reconstructed in LFTJ

![Graph with reusing intermediate results](image)

- By Caching only 5% of the intermediate results, \(^*80\%\) accesses can be reused

Why Naïve Caching Fails

- **Goal:** Compute join query by dynamically using available memory to speedup the query
- State of the art: materialize full intermediate results or none of them
- Which attribute should be used for caching in the next query?
  
  \( Q = R(x_1, x_2) \bowtie R(x_2, x_3) \bowtie R(x_3, x_4) \bowtie R(x_3, x_4) \)

- **Strategy:**
  - Build a Tree Decomposition (TD) of the query
  - Variable order – pre-order over TD
  - Caching over adhesions (parent-child intersections)
  - Execute LFTJ as usual

Cached Trie Join Execution Example

\[ Q = R(x_1, x_2) \bowtie R(x_2, x_3) \bowtie R(x_3, x_4) \bowtie R(x_4, x_5) \]

Trie structure:

Evaluation caches: \( C_{TD_1} = \{ (1,1), (1,2), (1,3), (2,1) \} \)

Counting caches: \( C_{TD_2} = \{ (1,1), (1,2), (2,1) \} \)

Numerical Experiments

- **Problem:** graph properties do not capture caching effectiveness
  - Solution:
    - Produce a large number of TDs (algorithm in the paper)
    - Cost function for selecting the best TD
      - e.g. cache dimension, skew, variable ordering [Chu, SIGMOD'15]

Summary

1. Extend trie join with caching
   - Worst-case optimal
   - Speedup vs memory consumption
2. Heuristic approach to enumerating TDs
3. Thorough experimental study
   - Evaluation and counting
   - State of the art comparison
   - Effect of cache properties

Comparison to the State of the Art

Limited Cache Size Results

\*3.5X slowdown on average for caching only 10% of all intermediate results