Optimal Join Algorithms meet Top-$k$

Nikolaos Tziavelis, Wolfgang Gatterbauer, Mirek Riedewald
Northeastern University, Boston

Part 3: Ranked Enumeration

Slides: https://northeastern-datalab.github.io/topk-join-tutorial/
DOI: https://doi.org/10.1145/3318464.3383132
Data Lab: https://db.khoury.northeastern.edu

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 4.0 International License. See https://creativecommons.org/licenses/by-nc-sa/4.0/ for details.
Outline tutorial

• Part 1: Top-$k$ (Wolfgang): ~20min
• Part 2: Optimal Join Algorithms (Mirek): ~30min
• Part 3: Ranked enumeration over Joins (Nikolaos): ~40min
  – Ranked Enumeration
  – Top-1 Result for Path Queries
  – From Top-1 to Any-$k$
    • Anyk-Part
    • Anyk-Rec
  – Beyond Path Queries
  – Ranking Function
  – Open Problems
Ranked Enumeration Example

<table>
<thead>
<tr>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>$A_2$</td>
<td>$w_1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

select $A_1, A_2, A_3, A_4,$
$w_1 + w_2 + w_3$ as weight
from $R_1, R_2, R_3$
where $R_1.A_1 = R_2.A_1$
and $R_2.A_2 = R_3.A_2$
order by weight
limit k any-k

<table>
<thead>
<tr>
<th>Rank-1</th>
<th>Rank-2</th>
<th>Rank-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 0, 2, 3, 17)</td>
<td>(2, 0, 2, 3, 18)</td>
<td>(3, 0, 2, 3, 19)</td>
</tr>
</tbody>
</table>

...
Ranked Enumeration: Problem Definition

RAM Cost Model: $TT(k) = \text{Time-to-}k^{th}\ \text{result}$

- $TTF = \text{Time-to-First} = TT(1)$
- Delay
- $TTL = \text{Time-to-Last} = TT(|\text{out}|)$

"Any-k"

Anytime algorithms + Top-k

Most important results first (ranking function on output tuples, e.g. sum of weights)

All results eventually returned
No need to set $k$ in advance
**Top-k**

- middleware cost model (# accesses)
- small result size; wish: $O(k)$
- return only $k$-best results

**Optimal Join Algorithms**

**Any-k**

- ranking function
- most important results first
- incremental computation
- conjunctive queries
- query decompositions
- minimize intermediate results

**RAM cost model**

- return all results; wish: $O(r), r > n$

- all results are equally important
Resorting to other paradigms

- **Using Top-\(k\):**
  - Most top-\(k\) join algorithms can be adapted to support ranked enumeration (\(k\) is usually not a hard requirement)
  - But different cost model, huge intermediate results

- **Using (Optimal) Join Algorithms:**
  - Batch computation of full output then sort
  - Good TTL, Bad TTF

How do we push the sorting into the join?
Unranked Enumeration

Related problem: enumerate join results in no particular order

What if we have projections?
[Bagan+ 07]: “Free-connex” acyclic queries
  • Linear pre-processing
  • Constant delay

Unranked Enumeration vs Ranked Enumeration

Challenge: return the output tuples in the right order

Our focus: ranking, no projections

Pre-processing: (1, 1, 3) vs (3, 2, 1)
Conceptual Roadmap

Join Problems:
- Top-1 Path Queries
- Top-1 Conjunctive Queries
- Union of Tree-DP (UT-DP)
- Any-k UT-DP over selective dioids

Tropical semiring (min, +)

Optimization:
- Paths/Serial
  - DP
  - Any-k DP

Ranked Enumeration:
- Cyclic/General
  - Top-1 Conjunctive Queries
  - Union of Tree-DP (UT-DP)
  - Any-k UT-DP

Any-k UT-DP over selective dioids
Outline tutorial

• Part 1: Top-$k$ (Wolfgang): ~20min
• Part 2: Optimal Join Algorithms (Mirek): ~30min
• Part 3: Ranked enumeration over Joins (Nikolaos): ~40min
  – Ranked Enumeration
  – Top-1 Result for Path Queries
    – From Top-1 to Any-$k$
      • Any$k$-Part
      • Any$k$-Rec
    – Beyond Path Queries
  – Ranking Function
  – Open Problems
Top-1 result

- **Idea**: Modify the bottom-up phase of Yannakakis to propagate the minimum weight
  - (min, +) operators in each step
  - Top-1 result can be constructed with one top-down traversal
Top-1 result: Example

\[ R_1 \]
\[
\begin{array}{ccc}
A_1 & A_2 & w_1 \\
1 & 0 & 1 \\
2 & 0 & 2 \\
3 & 0 & 3 \\
4 & 1 & 4 \\
\end{array}
\]

\[ R_2 \]
\[
\begin{array}{ccc}
A_2 & A_3 & w_2 \\
0 & 1 & 5 \\
0 & 1 & 7 \\
0 & 1 & 8 \\
0 & 2 & 6 \\
\end{array}
\]

\[ R_3 \]
\[
\begin{array}{ccc}
A_3 & A_4 & w_3 \\
1 & 1 & 20 \\
1 & 2 & 40 \\
2 & 3 & 10 \\
2 & 4 & 30 \\
\end{array}
\]
Top-1 result: Example

Nodes = Tuples
Edges = Joining pairs
Labels = Weights
Top-1 result: Example

Bottom-up
Each node passes on the minimum total weight it can reach.
Each node passes on the minimum total weight it can reach.

\[
\min(20,40) + 5 = 25
\]
Top-1 result: Example

Each node passes on the minimum total weight it can reach.
Each node passes on the minimum total weight it can reach

Minimum result weight = 17
Top-1 result: Example

Follow the winning edges

Top-down for Top-1 result
Top-1 result & DP

Rank-1 algorithm for path queries = (Serial) Dynamic Programming

Overlapping Subproblems

Subproblem from tuple “1”

Subproblem from tuple “5”

Subproblem
Minimum achievable weight starting from \( r_i \in R_i \)
Rank-1 algorithm for path queries = (Serial) Dynamic Programming

Principle of Optimality
An optimal solution must contain optimal solutions (to subproblems)

Edges = Decisions (Dependencies)

Nodes = States (Subproblems)

Relations = Stages (Independent problems)

Edges = Decisions (Dependencies)

Principle of Optimality
An optimal solution must contain optimal solutions (to subproblems)
DP Equi-join State Space

Total time = #Edges = $O(n^2 \ell)$
DP Equi-join State Space

Equivalent to the “messages” of Yannakakis

Transform the state space (at most one incoming/outgoing edge per tuple)

Total time = #Edges = $O(n \ell)$

Linear in the size of the database
Connection to Factorized Databases

\[ \begin{align*}
R_1 \rightarrow & \quad A_2 \rightarrow \quad R_2 \rightarrow \quad A_3 \rightarrow \quad R_3 \\
1 \quad & A_2 = 0 \quad 5 \quad & A_3 = 1 \quad 20 \\
2 \quad & \quad 7 \quad & \quad 40 \\
3 \quad & \quad 8 \quad & \quad 10 \\
4 \quad & \quad 6 \quad & \quad 30
\end{align*} \]

[Olteanu+ 16]: Conditional independence of the non-joining attributes given the joining attribute value

[Olteanu, Schleich. Factorized databases. SIGMOD Record'06](https://doi.org/10.1145/3003665.3003667)
Outline tutorial

• Part 1: Top-$k$ (Wolfgang): $\sim$20min
• Part 2: Optimal Join Algorithms (Mirek): $\sim$30min
• Part 3: Ranked enumeration over Joins (Nikolaos): $\sim$40min
  – Ranked Enumeration
  – Top-1 Result for Path Queries
  – From Top-1 to Any-$k$
    • Any-$k$-Part
    • Any-$k$-Rec
  – Beyond Path Queries
  – Ranking Function
  – Open Problems
DP as a Shortest Path Problem

- DP computation equivalent to finding the shortest path in a graph

Note: We ignore the artificial intermediate nodes for simplicity
K-Shortest Paths

• How do we find the $k^{th}$ best solution to a DP problem?
  - Rank-1 DP solution $\Rightarrow$ shortest path
  - Rank-$k$ DP solution $\Rightarrow$ $k^{th}$ shortest path
K-Shortest Paths

• Two major approaches for computing the $k^{th}$ shortest path in a directed acyclic multi-stage graph

• Anyk-Part
  - Partition the solution space

• Anyk-Rec
  - Recursively compute the lower-rank paths from all nodes (suffixes)
Outline tutorial

• Part 1: Top-\(k\) (Wolfgang): \(~20\text{min}\)
• Part 2: Optimal Join Algorithms (Mirek): \(~30\text{min}\)
• Part 3: Ranked enumeration over Joins (Nikolaos): \(~40\text{min}\)
  – Ranked Enumeration
  – Top-1 Result for Path Queries
  – From Top-1 to Any-k
  • Anyk-Part
  • Anyk-Rec
  – Beyond Path Queries
  – Ranking Function
  – Open Problems
Lawler-Murty Procedure

[Lawler 72]: generic procedure for ranked enumeration

• Repeatedly partitions the solution space
• Applicable to a wide range of problems
• Generalization of an earlier algorithm of [Murty 68]

Original Space

Disjoint Subspaces

Variables

Values

Best solution

Fixed

Available

[Lawler 72] Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. Management Science’72
https://doi.org/10.1287/mnsc.18.7.401

What can the 2\textsuperscript{nd} best path be?
Option 1: Deviate in the first stage
Option 2: Keep the first decision
Deviate in the second stage
Option 3: Keep the first and second decisions Deviate in the third stage
Partition the solution space into 3 disjoint subspaces (subgraphs)
Compute the best solution in each subspace
2\textsuperscript{nd} best = winner among the 3

(18)  
(26)  
(37)
Rank-$k$ path

- In general, maintain a global **Priority Queue**
  - Pop to find winner
  - Partition winner further
Anyk-Part: Default

• How do we find the best solution in each subspace?
• Default approach: Shortest path algorithm from scratch
Anyk-Part: Default

• How do we find the best solution in each subspace?
• Default approach: Shortest path algorithm from scratch
How do we find the best solution in each subspace?

Default approach: Shortest path algorithm from scratch

$O(n \ell)$ per new subspace
[Kimelfeld+ 06]:

• Ranked enumeration with delay linear in the size of the database
• Does not fully exploit the structure of the problem

Successor: given a prefix and a decision, what is the next best decision we can make?

Can we calculate the Top-1 weight of each subspace faster?

Fixed prefix: Same as previous solution

Choose “successor” of previous decision

Reach the terminal optimally
Anyk-Part: Exploiting the DP structure

- **Fixed prefix:** Same as previous solution
- **Computed from DP bottom-up**
- **Choose** "successor" of previous decision
- **Reach the terminal optimally**
Anyk-Part Variants

- We already know the minimum weight we can get from choosing each decision
- We just need to compare them to find the “successor”
- Do some pre-processing after DP bottom-up to accomplish that
  - 4 different variants

What is the successor of 6?
Anyk-Part Variant 1: “All”

[Yang+ 18]:

- The solutions will be compared by the global priority queue anyway, so insert all of them as potential successors
- But delay will again be linear in the size of the database
Anyk-Part Variant 2: “Eager”

- Invest more into pre-processing to get a lower delay
- Sort the decisions and find the true successor
Anyk-Part Variant 3: “Lazy”

- Sorting = wasted effort if enumeration is stopped early

[Chang+ 15]:
- sort incrementally with a priority queue (per node)
- store order for future reuse

Anyk-Part Variant 4: “Take2”

• We want to lower both preprocessing time and delay

[Tziavelis+ 20]:
• Build a heap (binary tree) in linear time
• Heap order gives only two potential successors (asymptotically same as one)

https://doi.org/10.14778/3397230.3397250
Anyk-Part Complexity

- \( O(\ell n) \) same as DP bottom-up
- \( O(k \log k) \) same as sorting \( k \) objects
- \( O(k \ell) \) needed to enumerate each result

\[
\begin{array}{|c|c|}
\hline
\text{Algorithm} & \text{TT}(k) \\
\hline
\text{TAKE2} & \mathcal{O}(\ell n + k(\log k + \ell)) \\
\text{LAZY} & \mathcal{O}(\ell n + k(\log k + \ell + \log n)) \\
\text{EAGER} & \mathcal{O}(\ell n \log n + k(\log k + \ell)) \\
\text{ALL} & \mathcal{O}(\ell n + k(\log k + \ell n)) \\
\hline
\end{array}
\]

(*) assuming constant-time lookup with hashing
Outline tutorial

• Part 1: Top-$k$ (Wolfgang): ~20min
• Part 2: Optimal Join Algorithms (Mirek): ~30min
• Part 3: Ranked enumeration over Joins (Nikolaos): ~40min
  – Ranked Enumeration
  – Top-1 Result for Path Queries
  – From Top-1 to Any-$k$
    • Anyk-Part
    • Anyk-Rec
  – Beyond Path Queries
  – Ranking Function
  – Open Problems
Anyk-Rec: Motivation

**Principle of Optimality (DP)**

If \( \Pi_1(s) \) begins with node \( r \) then
\[
\Pi_1(s) = s \circ \Pi_1(r)
\]

**Generalized Principle of Optimality**

If \( \Pi_k(s) \) begins with node \( r \) then
\[
\Pi_k(s) = s \circ \Pi_j(r) \text{ for some } j \leq k
\]

\( \Pi_k(s) = k^{th} \) shortest path from node \( s \)

Martins, Pascoal, Santos. A new improvement for a K shortest paths algorithm. Investigação Operacional’01  
Anyk-Rec: Example

Idea: Store ordering of lower-rank suffixes and reuse it as much as possible

For each node (e.g. 1) we want to compute the ranking of paths-suffixes

\[ \Pi_1(1) \]
\[ \Pi_2(1) \]
\[ \Pi_3(1) \]
Anyk-Rec: Example

PQ

<table>
<thead>
<tr>
<th></th>
<th>1 \circ \Pi_1(5) [25]</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1 \circ \Pi_1(7) [27]</td>
</tr>
<tr>
<td>8</td>
<td>1 \circ \Pi_1(8) [28]</td>
</tr>
<tr>
<td>6</td>
<td>1 \circ \Pi_1(6) [16]</td>
</tr>
</tbody>
</table>

Sorted List

Initially Empty

One entry per outgoing edge

Stores ordering of suffixes
Anyk-Rec: Example

<table>
<thead>
<tr>
<th>PQ</th>
<th>Sorted List</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>(1 \circ \Pi_1(5)) [25]</td>
<td></td>
</tr>
<tr>
<td>(1 \circ \Pi_1(7)) [27]</td>
<td></td>
</tr>
<tr>
<td>(1 \circ \Pi_1(8)) [28]</td>
<td></td>
</tr>
</tbody>
</table>

Pop

\(1 \circ \Pi_1(6)\) [16]

\(\Pi_1(6)\)
Anyk-Rec: Example

**PQ**

<table>
<thead>
<tr>
<th>6</th>
<th>1 \circ \Pi_1(6) [25]</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1 \circ \Pi_1(7) [27]</td>
</tr>
<tr>
<td>8</td>
<td>1 \circ \Pi_1(8) [28]</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Sorted List**

\[ \Pi_1(1) = 1 \circ \Pi_1(6) [16] \]

**Store**

\[ 1 \circ \Pi_1(6) [16] \]
Anyk-Rec: Example

PQ

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1 \circ \Pi_1(5) [25]</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1 \circ \Pi_1(7) [27]</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1 \circ \Pi_1(8) [28]</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1 \circ \Pi_2(6) [36]</td>
<td></td>
</tr>
</tbody>
</table>

Sorted List

\[ \Pi_1(1) = 1 \circ \Pi_1(6) [16] \]

Replace

1 \circ \Pi_2(6) [36]

Computed recursively

\[ \Pi_2(6) \]

\[ \Pi_1(6) \]
**Anyk-Rec: Example**

**PQ**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1 \circ \Pi_1(7)</td>
<td>[27]</td>
</tr>
<tr>
<td>7</td>
<td>1 \circ \Pi_1(8)</td>
<td>[28]</td>
</tr>
<tr>
<td>8</td>
<td>1 \circ \Pi_2(6)</td>
<td>[36]</td>
</tr>
</tbody>
</table>

**Sorted List**

\[ \Pi_1(1) = 1 \circ \Pi_1(6) \] [16]

**Pop**

1 \circ \Pi_1(5) [25]

**Diagram**

\[ \Pi_1(5) \]

1 \rightarrow 5, 7, 8, 6
Anyk-Rec: Example

\[
\begin{align*}
\text{Sorted List} & \quad \Pi_1(1) = 1 \circ \Pi_1(6) [16] \\
& \quad \downarrow \\
\Pi_2(1) & = 1 \circ \Pi_1(5) [25] \\
\end{align*}
\]

\[
\begin{align*}
\text{PQ} & \\
\begin{array}{l}
1 \circ \Pi_1(7) [27] \\
1 \circ \Pi_1(8) [28] \\
1 \circ \Pi_2(6) [36] \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
& \quad 1 \circ \Pi_1(5) [25] \\
& \quad \Pi_1(5) \\
\end{align*}
\]

\[
\begin{align*}
1 & \quad 5 \\
& \quad 7 \\
& \quad 8 \\
& \quad 6 \\
\end{align*}
\]
Anyk-Rec: Example

**PQ**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1 ( \circ ) ( \Pi_2(5) ) [45]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1 ( \circ ) ( \Pi_1(7) ) [27]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1 ( \circ ) ( \Pi_1(8) ) [28]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1 ( \circ ) ( \Pi_2(6) ) [36]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Sorted List**

\[ \Pi_1(1) = 1 \circ \Pi_1(6) [16] \]
\[ \Pi_2(1) = 1 \circ \Pi_1(5) [25] \]

**Replace**

\[ 1 \circ \Pi_2(5) [45] \]

**Computed recursively**
Anyk-Rec: Suffix Reusage

Later...

Reus\( \Pi_2(1) \) for all subsequent calls!
Anyk-Rec: Suffix Reusage

• In general, delay is higher than Anyk-Part
  - $O(\ell \log n)$ vs $O(\log k + \ell)$ of Take2

• But reusing computation may pay off in the end
  - If a lot of suffixes are shared, TTL can be faster than sorting!

• If join pattern = Cartesian product with $n^\ell$ results:
  - Anyk-Rec TTL: $O(n^\ell (\log n + \ell))$
  - Sorting the output: $O(n^\ell \log n \cdot \ell)$
More on the History of Anyk-Rec

- [Bellman+ 60]: Keep the $k$ best solutions per node
- [Dreyfus 69]: Recursive equations
- [Jiménez+ 99]: Top-down approach
- [Deep+ 19]: Application to conjunctive queries
- [Tziavelis+ 20]: Improved TTL guarantees


[Dreyfus 69] Dreyfus. An appraisal of some shortest-path algorithms. Operations research’69 https://doi.org/10.1287/opre.17.3.395


Overview

- Take2 has lower complexity over all instances
- But there are cases where the recursive approach wins for TTL

| Algorithm | TT\( (k) \) | TTL for \( |\text{out}| = \Omega(\ell n) \) | TTL for \( |\text{out}| = \Theta(n^\ell) \) |
|-----------|-------------|---------------------------|---------------------------|
| **RECURSIVE** | \( \mathcal{O}(\ell n + k \ell \log n) \) | \( \mathcal{O}(|\text{out}| \ell \log n) \) | \( \mathcal{O}(n^\ell (\log n + \ell)) \) |
| Take2      | \( \mathcal{O}(\ell n + k(\log k + \ell)) \) | \( \mathcal{O}(|\text{out}| (\log |\text{out}| + \ell)) \) | \( \mathcal{O}(n^\ell \cdot \ell \log n) \) |
| Lazy       | \( \mathcal{O}(\ell n + k(\log k + \ell + \log n)) \) | \( \mathcal{O}(|\text{out}| (\log |\text{out}| + \ell)) \) | \( \mathcal{O}(n^\ell \cdot \ell \log n) \) |
| Eager      | \( \mathcal{O}(\ell n \log n + k(\log k + \ell)) \) | \( \mathcal{O}(|\text{out}| (\log |\text{out}| + \ell)) \) | \( \mathcal{O}(n^\ell \cdot \ell \log n) \) |
| All        | \( \mathcal{O}(\ell n + k(\log k + \ell n)) \) | \( \mathcal{O}(|\text{out}| (\log |\text{out}| + \ell)) \) | \( \mathcal{O}(n^\ell \cdot \ell \log n) \) |
| Batch      | \( \mathcal{O}(\ell n + |\text{out}| (\log |\text{out}| + \ell)) \) | \( \mathcal{O}(|\text{out}| (\log |\text{out}| + \ell)) \) | \( \mathcal{O}(n^\ell \cdot \ell \log n) \) |

(*) assuming constant-time lookup with hashing
Some Experimental Results

- Anyk starts much faster than Batch
- Anyk-Rec also finishes faster than Batch

Tziavelis, Ajwani, Gatterbauer, Riedewald, Yang. Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries. PVLDB’20
https://doi.org/10.14778/3397230.3397250
Some Experimental Results

- Boolean (is there any result?) is the best we can do
- Anyk-Rec is getting faster when there are more opportunities for suffix reusage
- Anyk is only 2 times slower
Outline tutorial

• Part 1: Top-$k$ (Wolfgang): ~20min
• Part 2: Optimal Join Algorithms (Mirek): ~30min
• Part 3: Ranked enumeration over Joins (Nikolaos): ~40min
  – Ranked Enumeration
  – Top-1 Result for Path Queries
  – From Top-1 to Any-$k$
    • Any$k$-Part
    • Any$k$-Rec
  – Beyond Path Queries
  – Ranking Function
  – Open Problems
Acyclic Queries

If the query is acyclic, it can be represented by a join tree
Tree-DP

Stages of DP form a tree instead of a path (Tree-DP)

For Top-1, go bottom-up and choose decisions independently in each branch
Ranked Enumeration for Tree-DP

- **Anyk-Part:**
  - Serialize the stages and treat it like the path case
  - Complexity guarantees remain the same

- **Anyk-Rec:**
  - Apply the path algorithm in each branch
  - Difficulty: how do we combine the solutions from each branch?
  - Improved TTL only if the tree has significant depth


Cyclic Queries

- For cyclic queries, use tree decompositions
- Submodular width decompositions: union of acyclic queries

\[
\begin{array}{c}
\text{Acyclic 1} \\
O(n^{5/3})
\end{array}
\quad
\begin{array}{c}
\text{Acyclic 2} \\
O(n^{5/3})
\end{array}
\quad
\begin{array}{c}
\text{Acyclic 3} \\
O(n^{5/3})
\end{array}
\quad
\begin{array}{c}
\text{Acyclic 4} \\
O(n^{5/3})
\end{array}
\quad
\begin{array}{c}
\text{Acyclic 5} \\
O(n^{5/3})
\end{array}
\quad
\begin{array}{c}
\text{Acyclic 6} \\
O(n^{5/3})
\end{array}
\quad
\begin{array}{c}
\text{Acyclic 7} \\
O(n^{5/3})
\end{array}
\]
Ranked Enumeration for Cyclic Queries

- Straightforward to run any-k with a top-level Priority Queue
- $\text{TTF} = O(n^{5/3})$ for $Q_{6c}$ (same as Boolean query)
Outline tutorial

• Part 1: Top-$k$ (Wolfgang): ~20min
• Part 2: Optimal Join Algorithms (Mirek): ~30min
• Part 3: Ranked enumeration over Joins (Nikolaos): ~40min
  – Ranked Enumeration
  – Top-1 Result for Path Queries
  – From Top-1 to Any-$k$
    • Any$k$-Part
    • Any$k$-Rec
  – Beyond Path Queries
  – Ranking Function
  – Open Problems
What ranking functions can be supported?

So far (min, +). Can we substitute these operators with others?

1. We need to be able to do Dynamic Programming

2. The 1\textsuperscript{st} operator has to induce an order on the domain

\[
\begin{align*}
R_2 & \quad 5 \quad 25 \\
R_3 & \quad 20 \quad 40 \\
\min(20, 40) + 5 & = 25
\end{align*}
\]
Semirings

- Semiring \((W, \oplus, \otimes, 0, 1)\)
  1. \((W, \oplus, 0)\) is commutative monoid
  2. \((W, \otimes, 1)\) is monoid
  3. \(\otimes\) distributes over \(\oplus\): \((x \oplus y) \otimes z = (x \otimes z) \oplus (y \otimes z)\)
  4. 0 annihilates \(\otimes\): \(0 \otimes x = 0\)

- Examples
  1. \((R^\infty, \min, +, \infty, 0)\) “Tropical semiring”
  2. \(\{0,1\}, \lor, \land, 0, 1\) Boolean
  3. \((N, +, \cdot, 0, 1)\) Number of paths

---

Mohri. Semiring frameworks and algorithms for shortest-distance problems. JALC’02

No ordering  
(What would the 2\textsuperscript{nd} best solution be?)
Selective Dioids

- A selective dioid $(W, \oplus, \otimes, 0, 1)$ is a semiring with an additional property:
  - $\oplus$ is selective: $(x \oplus y = x) \lor (x \oplus y = y)$

- Selectivity of $\oplus$ gives us a total order on $W$:
  - $x \leq y$ iff $x \oplus y = x$
  - E.g. $x \leq y$ iff $\min(x, y) = x$

DP & Yannakakis

Yannakakis Bottom-up: DP over Boolean semiring ($\{0,1\}, \lor, \land, 0, 1$)

Any-k with Boolean semiring?
Equivalent to standard query evaluation of Yannakakis if we use “smarter” PQs (sorted lists of 0-1)

Dangling tuple

Yannakakis
Minimum SUM
Lexicographic Orders

$R_1 \rightarrow R_2 \rightarrow R_3$

Lexicographic Order

$R_2 \rightarrow R_1 \rightarrow R_3$

Results first weighted on $R_2$ then $R_1$ then $R_3$

(5, 1, 20) $\rightarrow$ (5, 1, 40) $\rightarrow$ (5, 2, 20) $\rightarrow$ ...

(5, 2, 20) $\rightarrow$ ...

(5, 1, 40) $\rightarrow$ ...

(5, 1, 20) $\rightarrow$ ...

...
Lexicographic Orders

$\mathbb{R}_1 \leq \mathbb{R}_2 \leq \mathbb{R}_3$

$W$: $\ell$-dimensional vectors

5 has input weight $(0, 5, 0)$

$\oplus$: lexicographic min

$(0, 5, 20) \oplus (0, 6, 10) = (0, 5, 20)$

$\otimes$: element-wise addition

$(0, 0, 20) \otimes (0, 5, 0) = (0, 5, 20)$
Outline tutorial

• Part 1: Top-\(k\) (Wolfgang): \(~20\text{min}\)
• Part 2: Optimal Join Algorithms (Mirek): \(~30\text{min}\)
• Part 3: Ranked enumeration over Joins (Nikolaos): \(~40\text{min}\)
  – Ranked Enumeration
  – Top-1 Result for Path Queries
  – From Top-1 to Any-k
    • Anyk-Part
    • Anyk-Rec
  – Beyond Path Queries
  – Ranking Function
  – Open Problems
Open Problems

• How does any-k interact with other relational operators?
  - Projections (drawing ideas from constant-delay enumeration)
  - Disjunctions
  - Groupings

• How does the query plan affect the performance of any-k algorithms? How would the database optimizer choose the best algorithm/join plan?

• Can we efficiently decompose every query into a union of disjoint trees?

• Can we prove results beyond the worst-case? (e.g. instance-optimality)

• Can we “push” the any-k functionality inside the bags of the tree decomposition instead of materializing them beforehand?