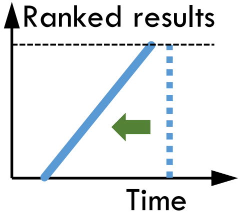


SIGMOD 2020 tutorial

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

Northeastern University Khoury College of Computer Sciences

DATALAB @Northeastern

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

R_1

A_1	A_2	w_1
1	0	1
2	0	2
3	0	3
4	1	4

R_2

A_2	A_3	w_2
0	1	5
0	1	7
0	1	8
0	2	6

R_3

A_3	A_4	w_3
1	1	20
1	2	40
2	3	10
2	4	30

```
select      A1, A2, A3, A4,
            w1 + w2 + w3 as weight
from        R1, R2, R3
where       R1.A1=R2.A1
   and      R2.A2=R3.A2
order by weight
```

~~limit k~~ any-k

<u>Rank-1</u>

(1, 0, 2, 3, 17) ⇒

<u>Rank-2</u>

(2, 0, 2, 3, 18) ⇒

<u>Rank-3</u>

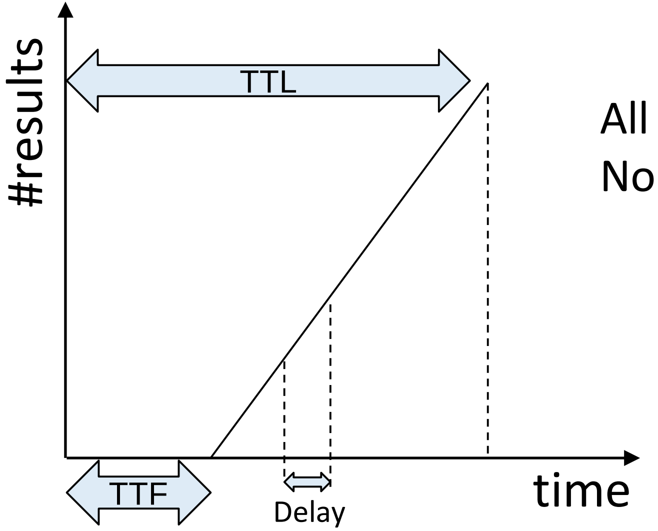
(3, 0, 2, 3, 19) ⇒ …

Ranked Enumeration: Problem Definition

"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

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

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

Top-k

Optimal Join Algorithms

Any-k

middleware
cost model
(# accesses)

small result size;
wish: $O(k)$

return only
k-best results

ranking function

most important
results first

RAM cost model

conjunctive queries

query
decompositions

minimize
intermediate
results

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

all results
are equally
important

incremental
computation

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

R_1

A_1	A_2
1	1
2	4
3	2

R_2

A_2	A_3
2	1
5	2
1	3

$\xrightarrow{\text{Pre-processing}}$ (1, 1, 3) $\xrightarrow{\text{Delay}}$ (3, 2, 1)

<u>What if we have projections?</u>

[Bagan+ 07]: “Free-connex” acyclic queries

- Linear pre-processing
- Constant delay

[Bagan+ 07] Bagan, Durand, Grandjean. On acyclic conjunctive queries and constant delay enumeration. CSL'07 https://doi.org/10.1007/978-3-540-74915-8_18

Unranked Enumeration vs Ranked Enumeration

<u>Challenge</u>: return the output tuples in the right order

R_1

A_1	A_2
1	1
2	4
3	2

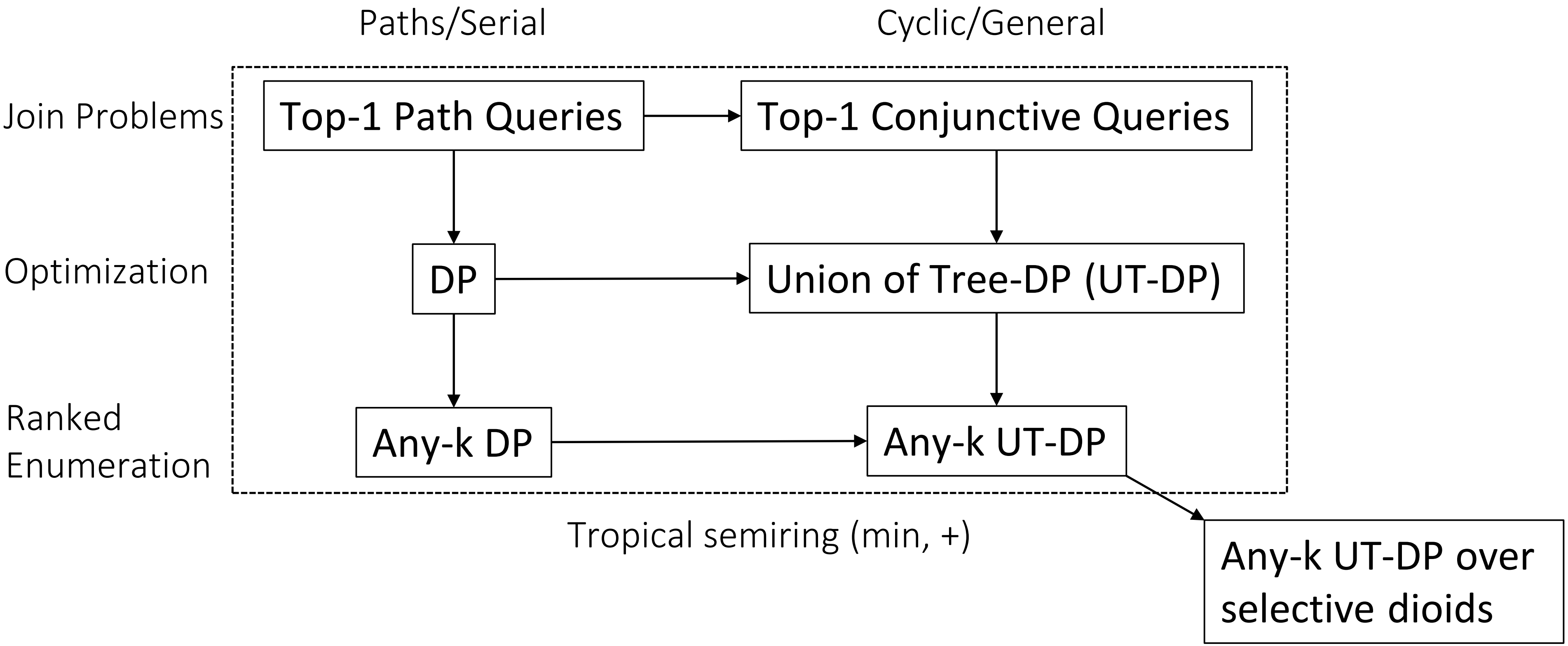
R_2

A_2	A_3
2	1
5	2
1	3

Pre-processing $\Longrightarrow$ (1, 1, 3) vs (3, 2, 1) ?

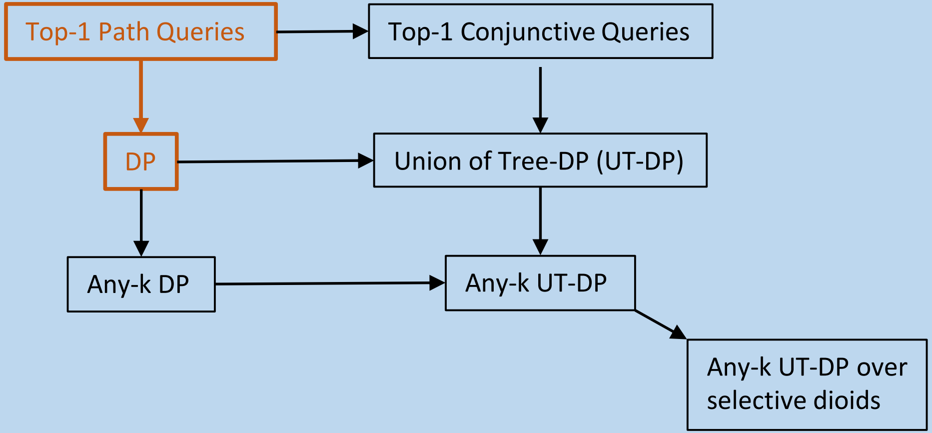
Our focus: ranking, no projections

Conceptual Roadmap



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



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

A_1	A_2	w_1
1	0	1
2	0	2
3	0	3
4	1	4

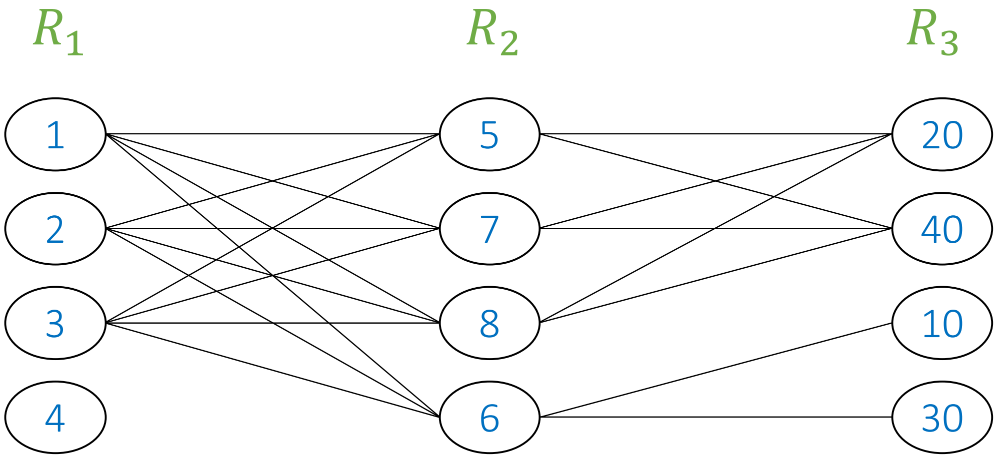
R_2

A_2	A_3	w_2
0	1	5
0	1	7
0	1	8
0	2	6

R_3

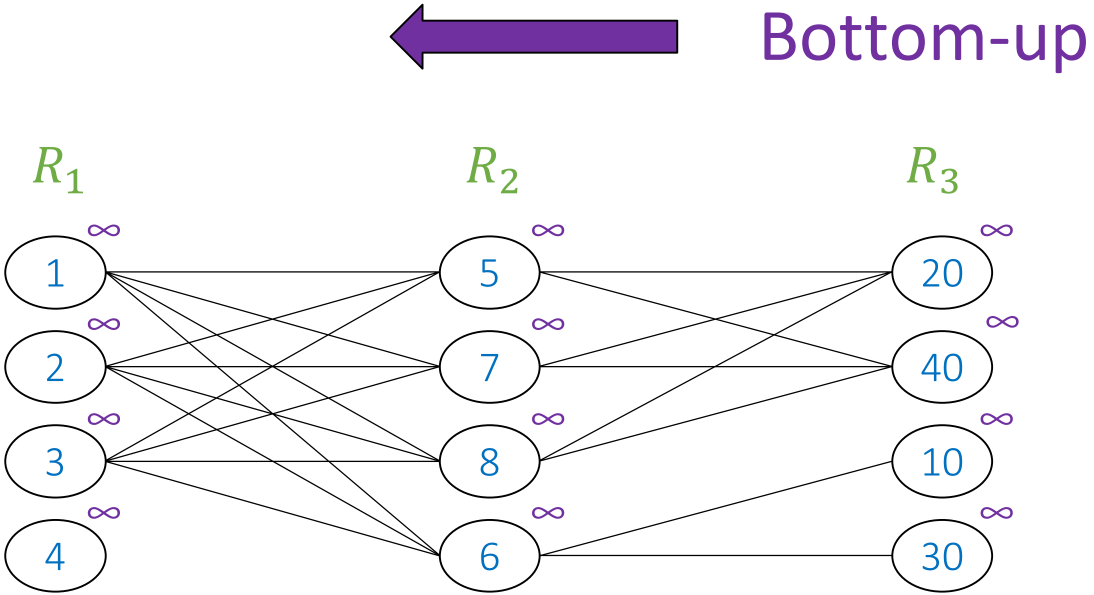
A_3	A_4	w_3
1	1	20
1	2	40
2	3	10
2	4	30

Top-1 result: Example

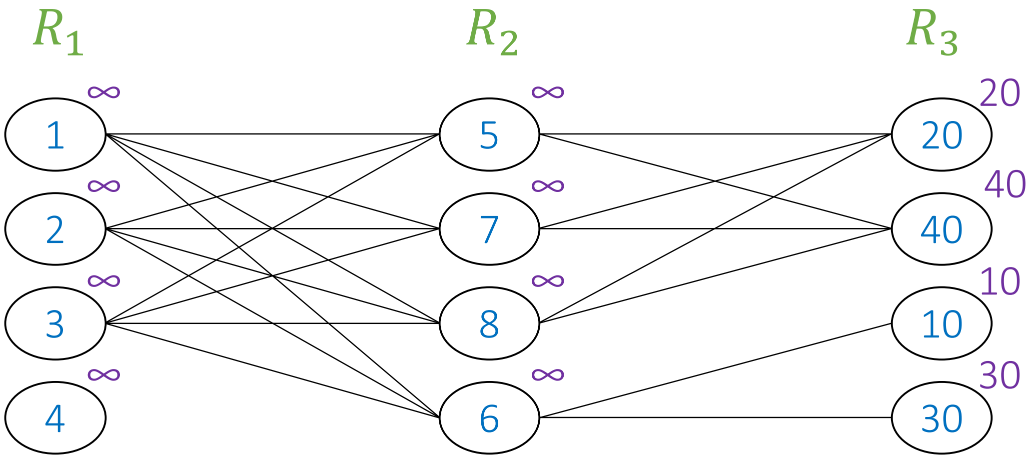


Nodes = Tuples
Edges = Joining pairs
Labels = Weights

Top-1 result: Example

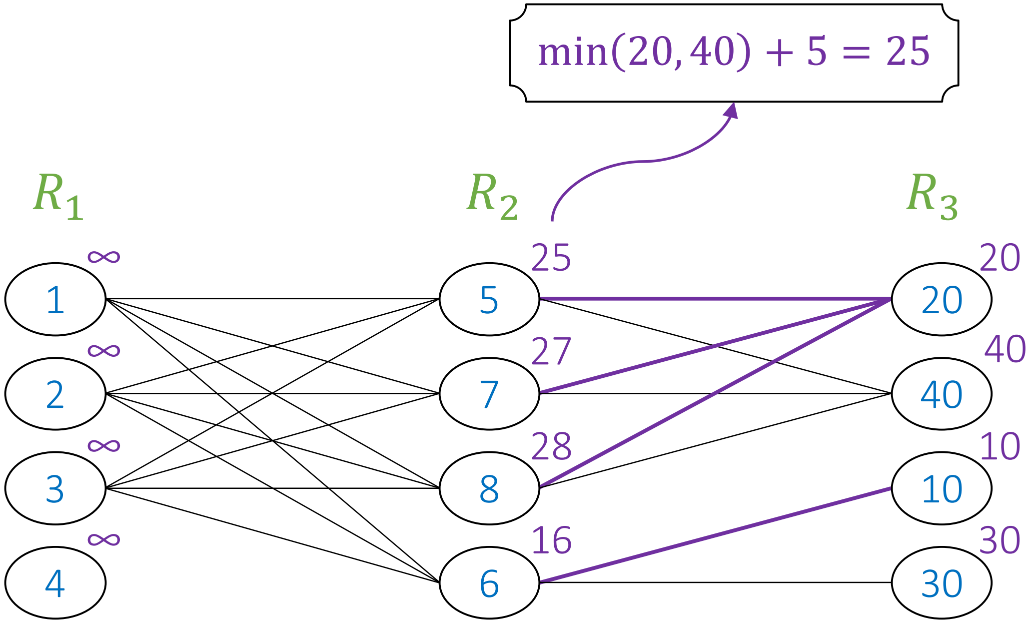


Top-1 result: Example



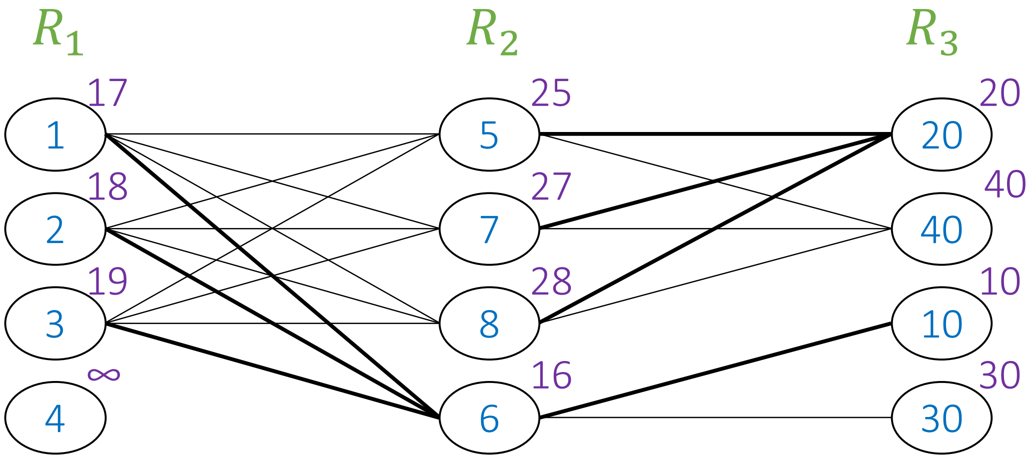
Each node passes on the minimum total weight it can reach

Top-1 result: Example



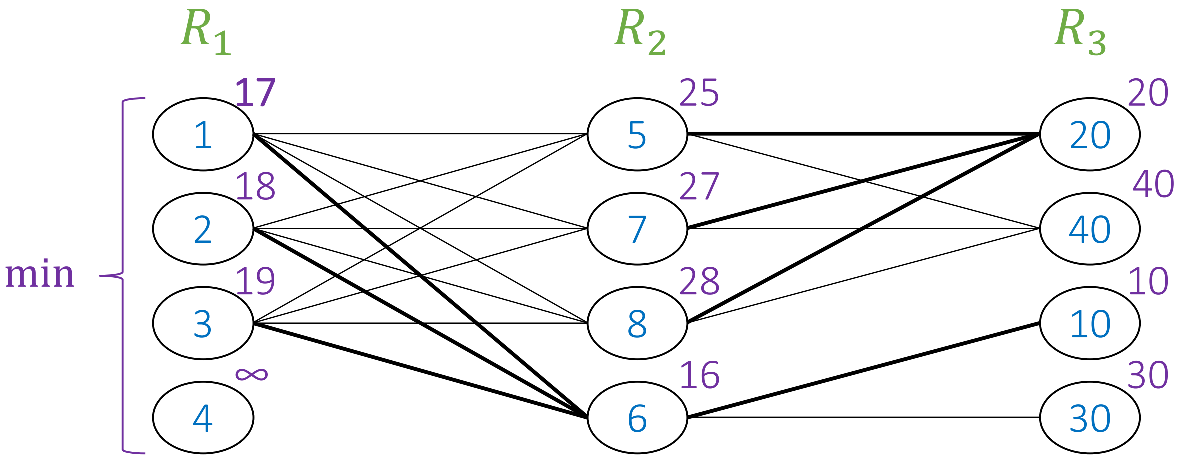
Each node passes on the minimum total weight it can reach

Top-1 result: Example



Each node passes on the minimum total weight it can reach

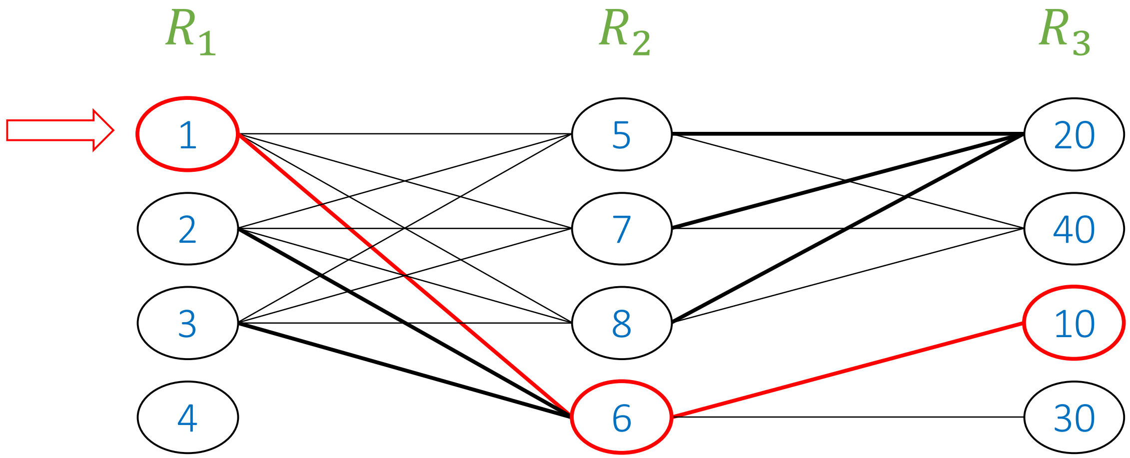
Top-1 result: Example



Each node passes on the minimum total weight it can reach

Minimum result weight = 17

Top-1 result: Example

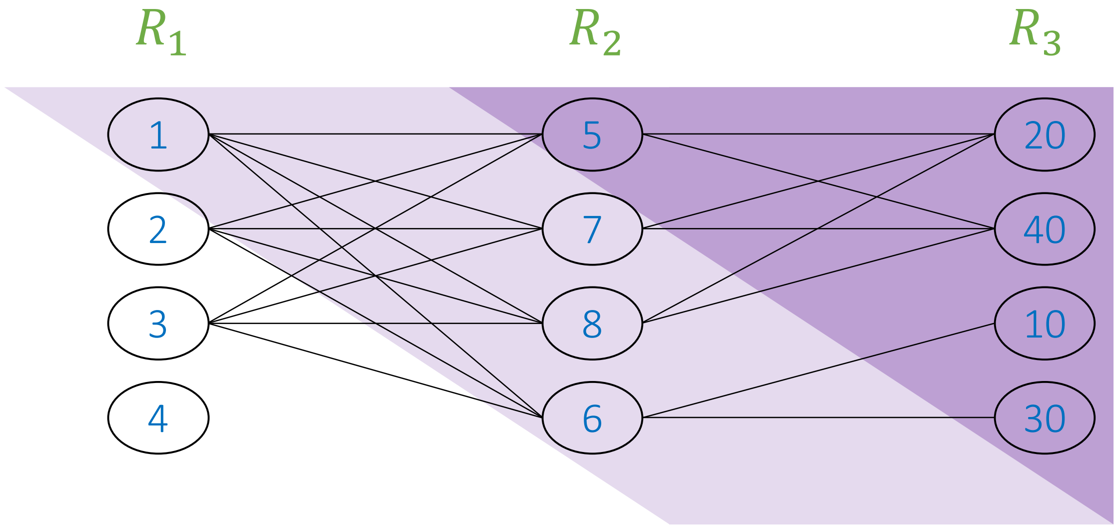


Top-down for
Top-1 result

Follow the winning edges

Top-1 result & DP

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



Subproblem
Minimum achievable weight starting from $r_i \in R_i$

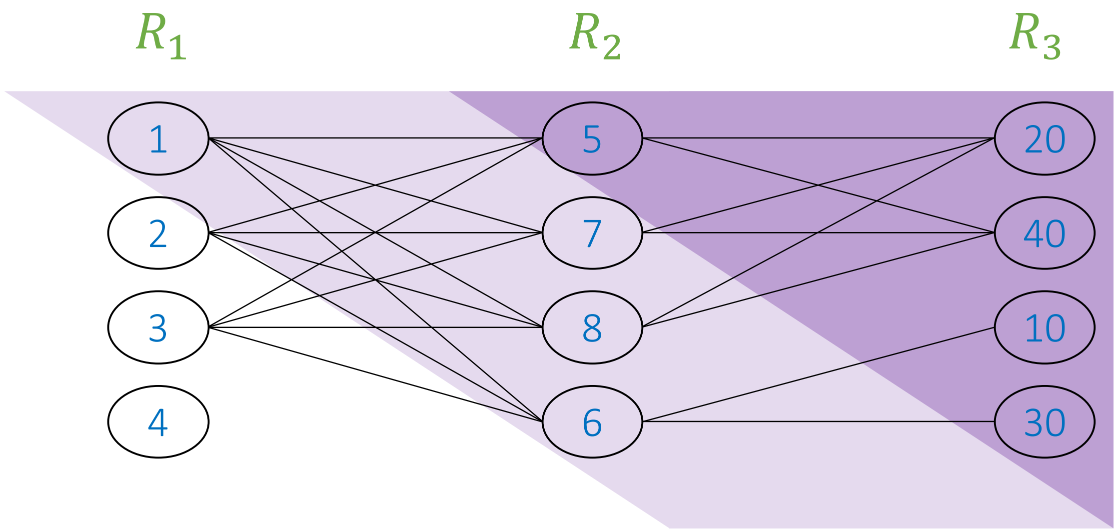
Subproblem from tuple “5”

Subproblem from tuple “1”

Overlapping Subproblems

Top-1 result & DP

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



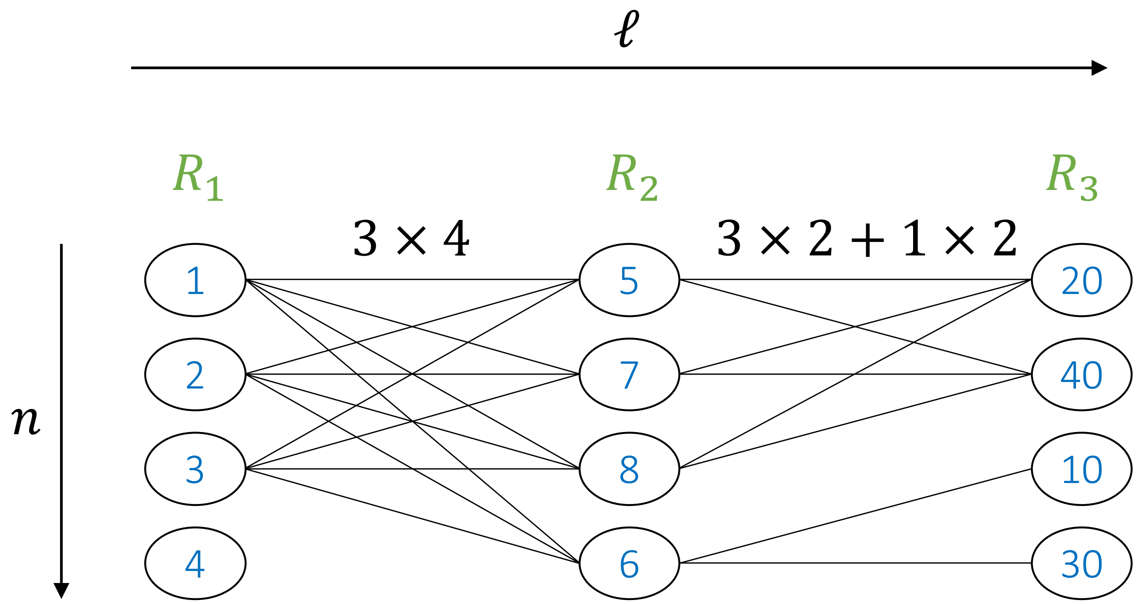
Relations = Stages
(Independent problems)

Nodes = States
(Subproblems)

<u>Principle of Optimality</u>
An optimal solution must contain optimal solutions (to subproblems)

Edges = Decisions
(Dependencies)

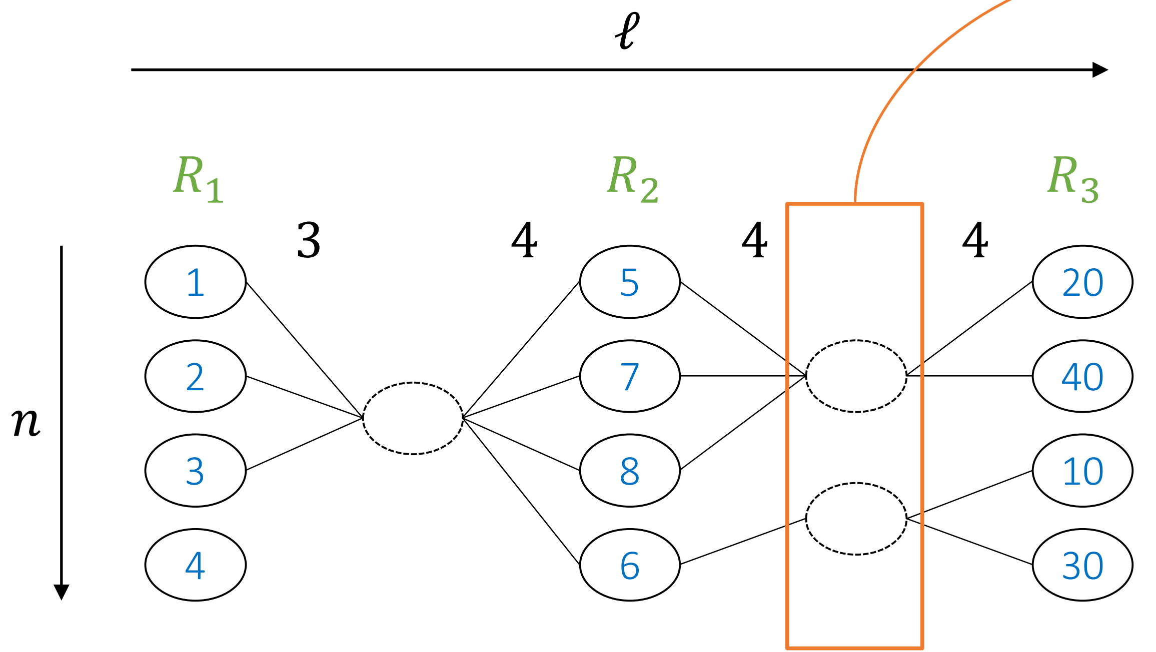
DP Equi-join State Space



Total time = #Edges = $O(n^2\ \ell)$

DP Equi-join State Space

Equivalent to the "messages" of Yannakakis

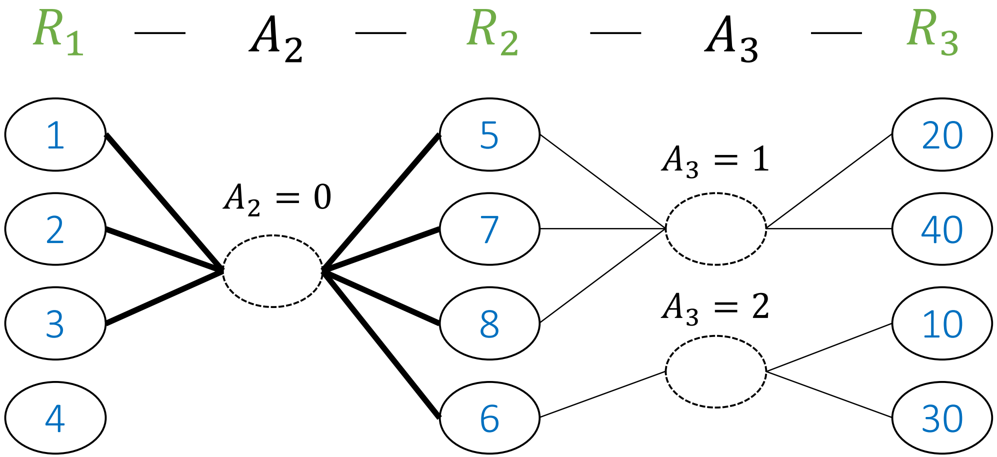


Transform the state space (at most one incoming /outgoing edge per tuple)

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

Linear in the size of the database

Connection to Factorized Databases

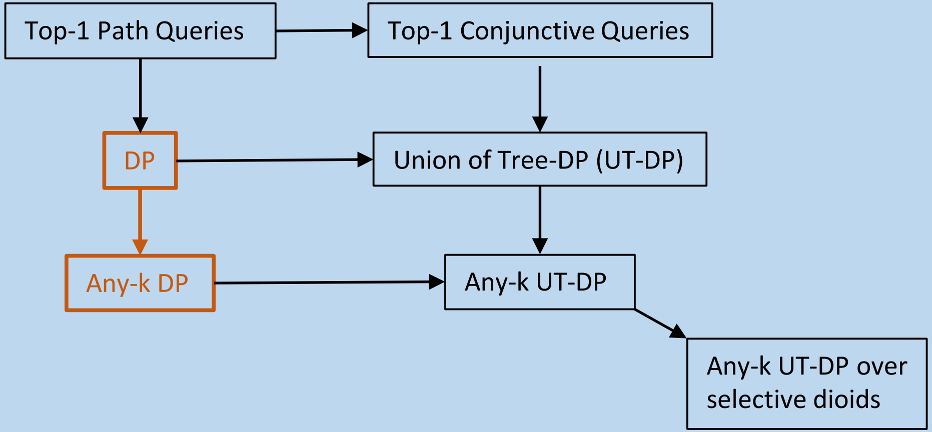


[Olteanu+ 16]:
Conditional independence of the non-joining attributes given the joining attribute value

[Olteanu+ 16] Olteanu, Schleich. Factorized databases. SIGMOD Record'06 https://doi.org/10.1145/3003665.3003667

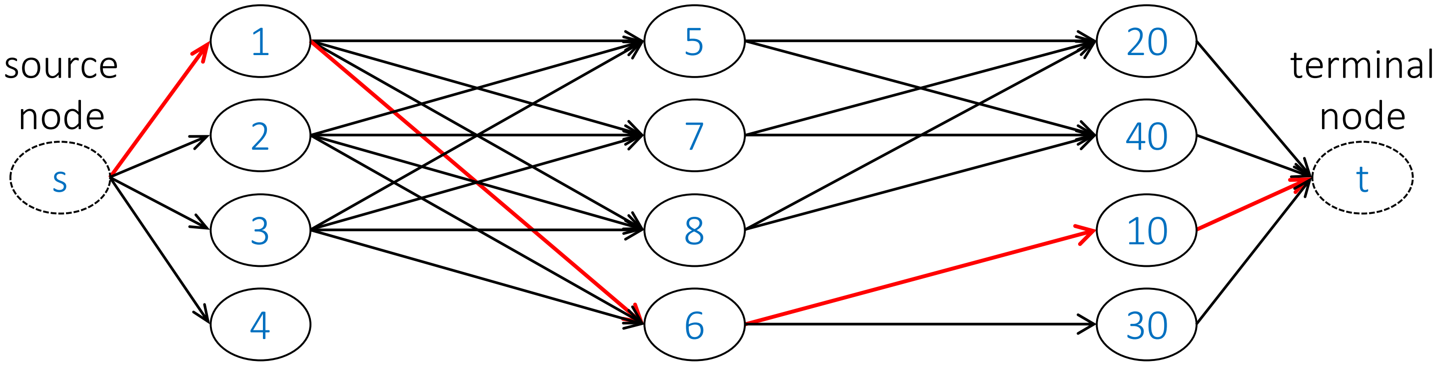
Outline tutorial

- 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



DP as a Shortest Path Problem

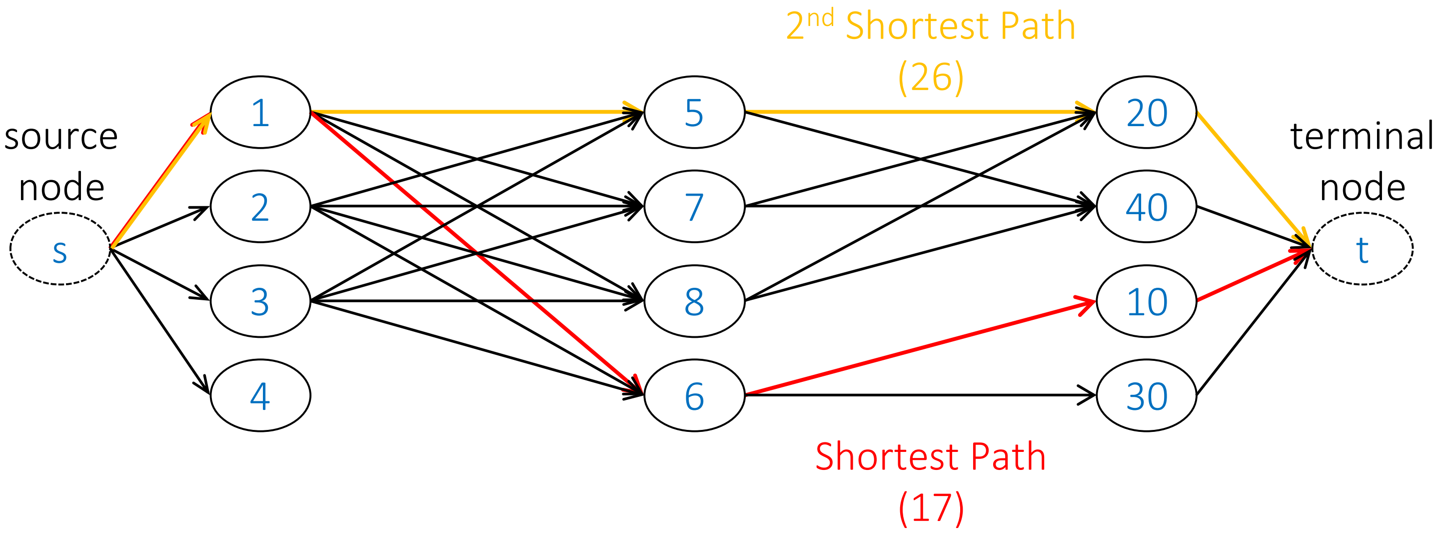
- DP computation equivalent to finding the shortest path in a graph



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 => shortest path
  - Rank-k DP solution => k^{th} shortest path



K-Shortest Paths

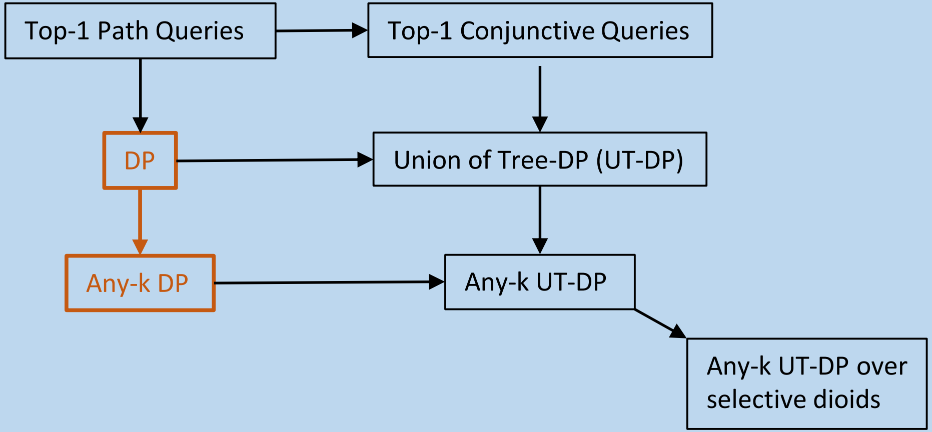
- Two major approaches for computing the k^{th} shortest path in a directed acyclic multi-stage graph

- 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): ~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

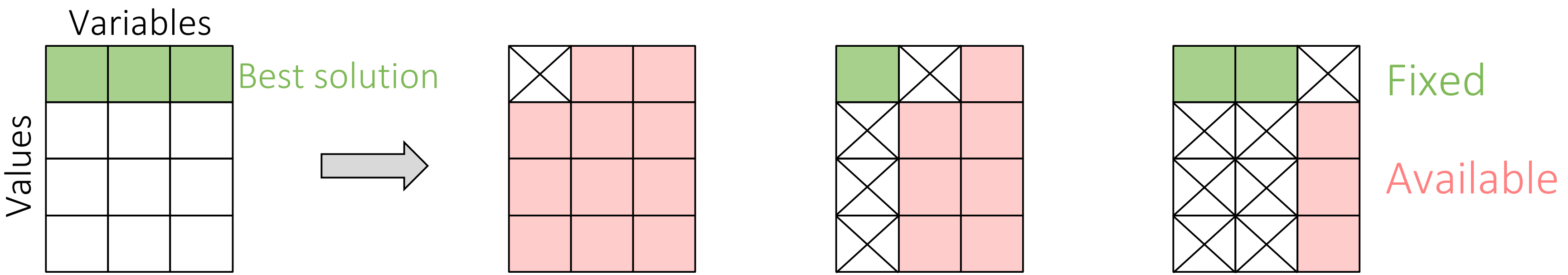


Lawler-Murty Procedure

[Lawler 72]: generic procedure for ranked enumeration

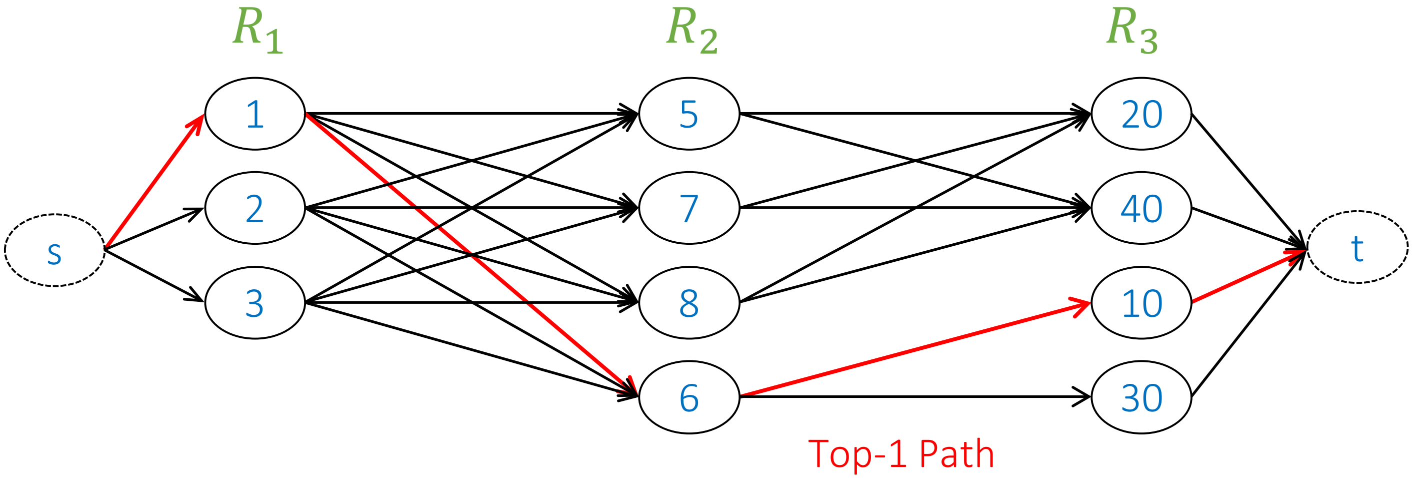
- Repeatedly partitions the solution space
- Applicable to a wide range of problems
- Generalization of an earlier algorithm of [Murty 68]

Original Space **Disjoint Subspaces**



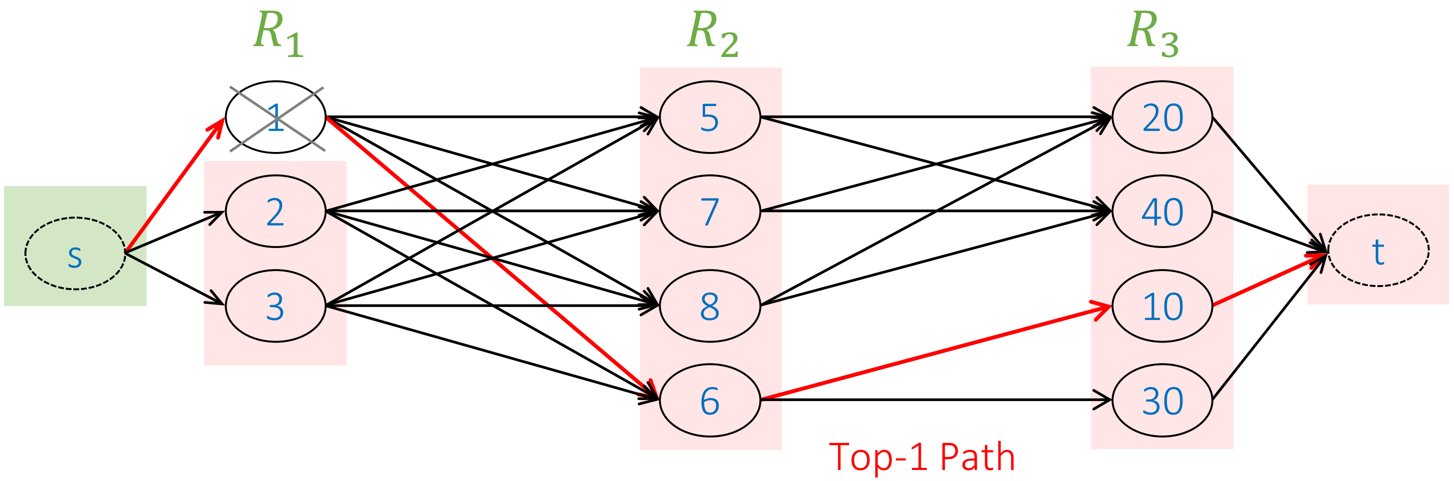
[Lawler 72] Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. Management Science'72 https://doi.org/10.1287/mnsc.18.7.401
[Murty 68] Murty. An Algorithm for Ranking all the Assignments in Order of Increasing Cost. Operations Research'68 https://doi.org/10.1287/opre.16.3.682

2^{nd} Best Path



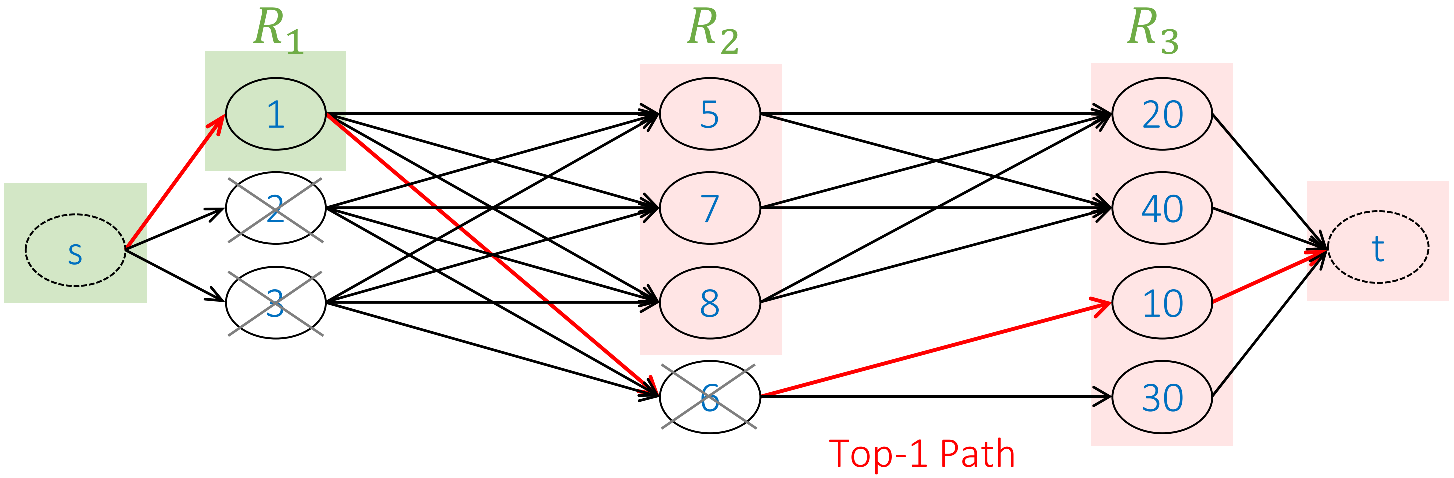
What can the 2^{nd} best path be?

2^{nd} Best Path



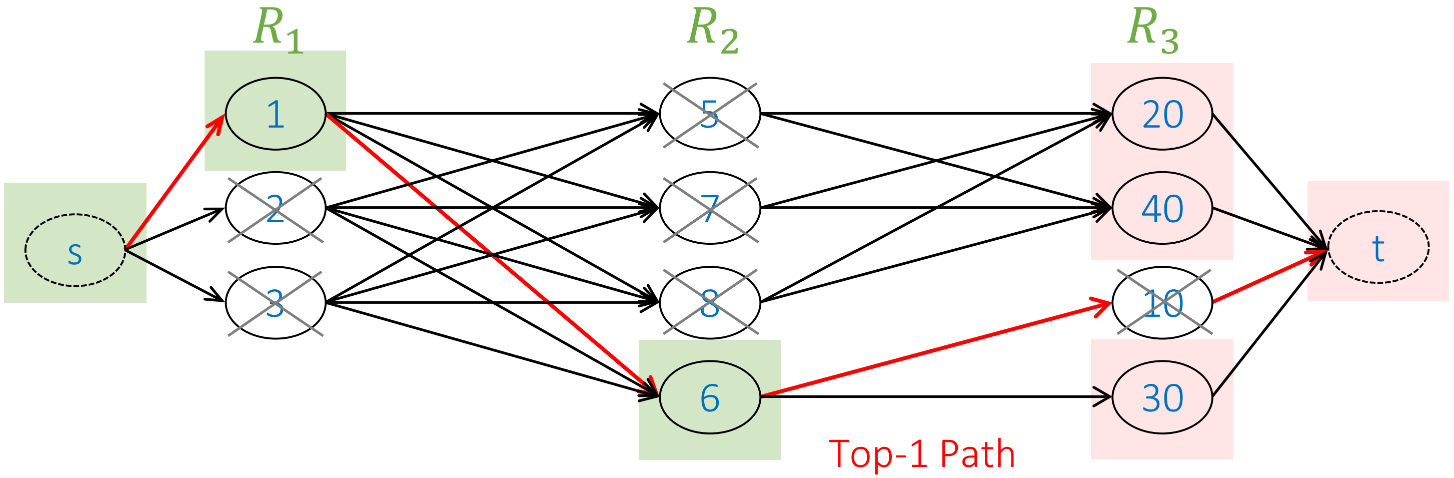
Option 1: Deviate in the first stage

2^{nd} Best Path



Option 2: Keep the first decision
Deviate in the second stage

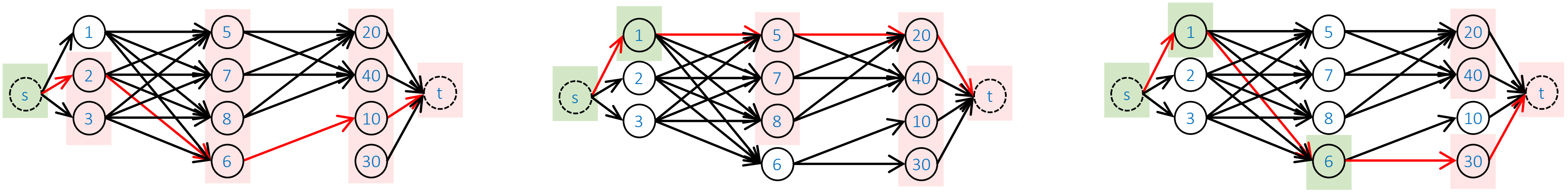
2nd Best Path



Option 3: Keep the first and second decisions
Deviate in the third stage

2nd Best Path

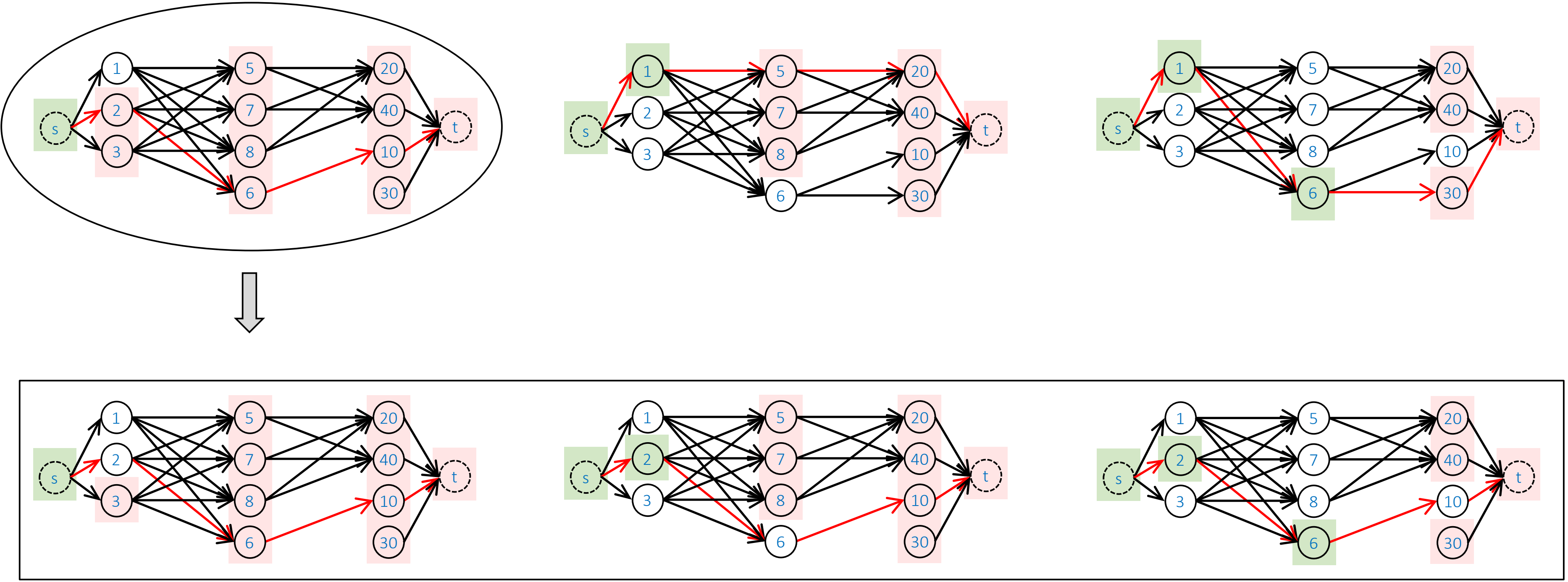
- Partition the solution space into 3 disjoint subspaces (subgraphs)
- Compute the best solution in each subspace
- 2nd best = winner among the 3



(18) (26) (37)

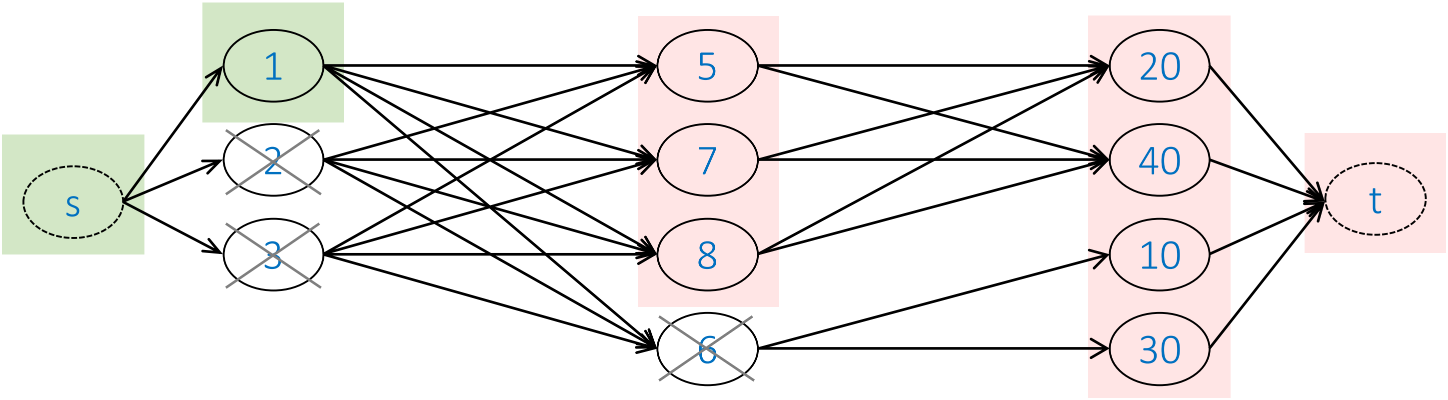
Rank-k path

- In general, maintain a global Priority Queue
  - Pop to find winner
  - Partition winner further



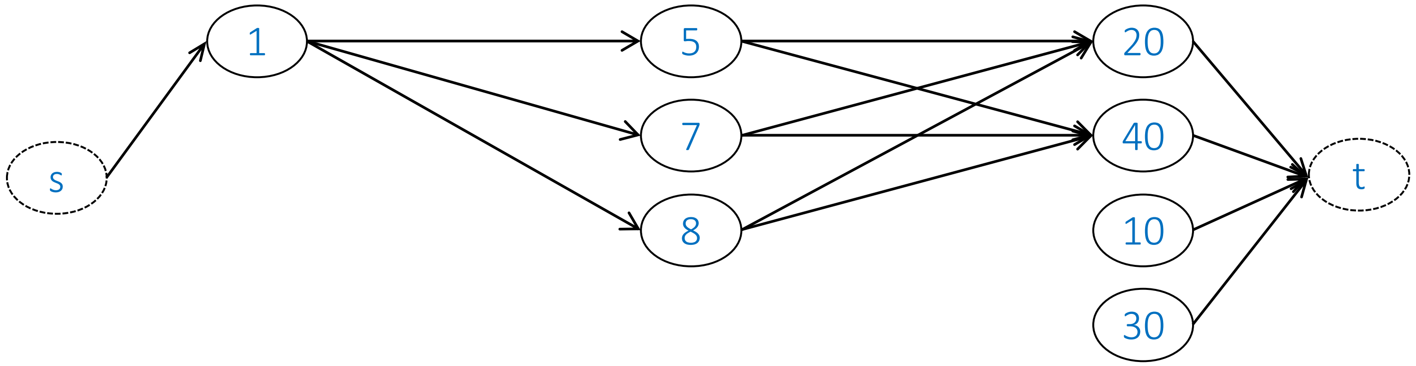
Anyk-Part: Default

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



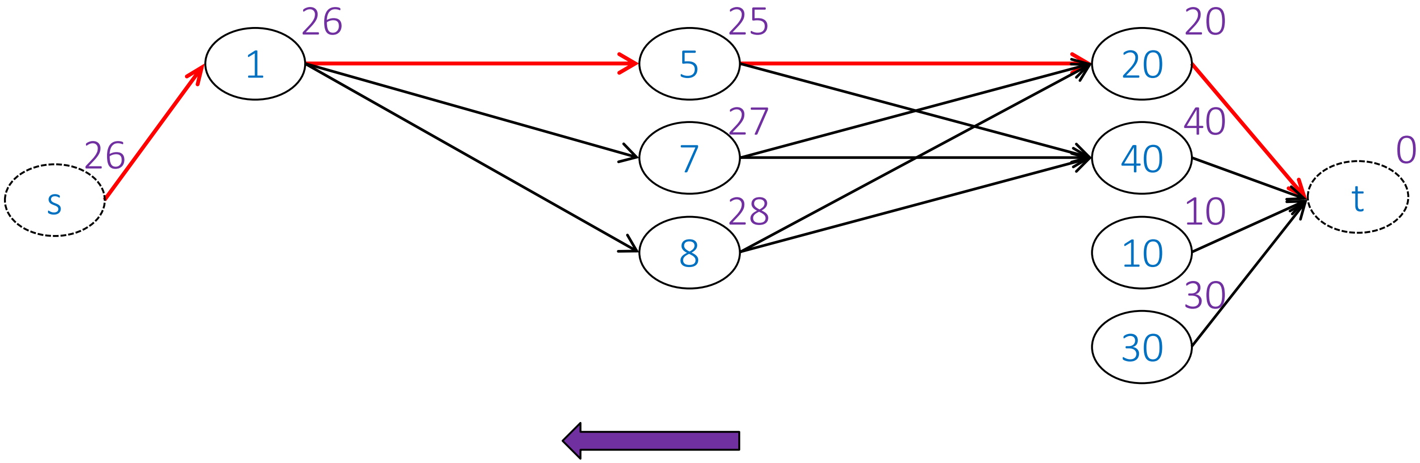
Anyk-Part: Default

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



Anyk-Part: Default

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



$O(n\,\ell)$ per new subspace

Anyk-Part: Default

[Kimelfeld+ 06]:

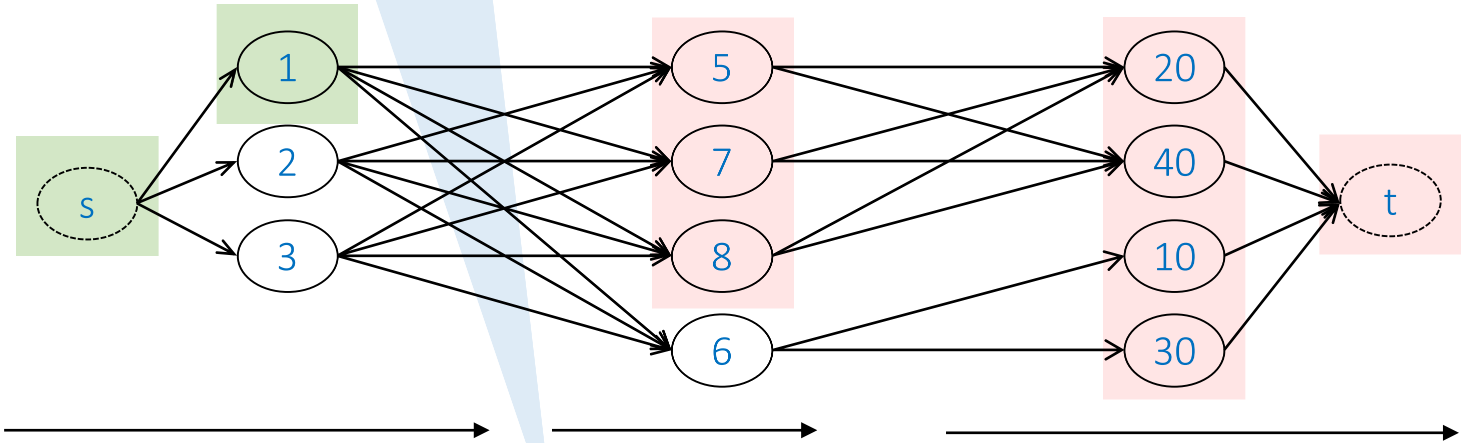
- Ranked enumeration with delay linear in the size of the database
- Does not fully exploit the structure of the problem

[Kimelfeld+ 06] Kimelfeld, Sagiv. Incrementally Computing Ordered Answers of Acyclic Conjunctive Queries. NGITS'06 https://doi.org/https://doi.org/10.1007/11780991_13

Anyk-Part: Exploiting the DP structure

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?

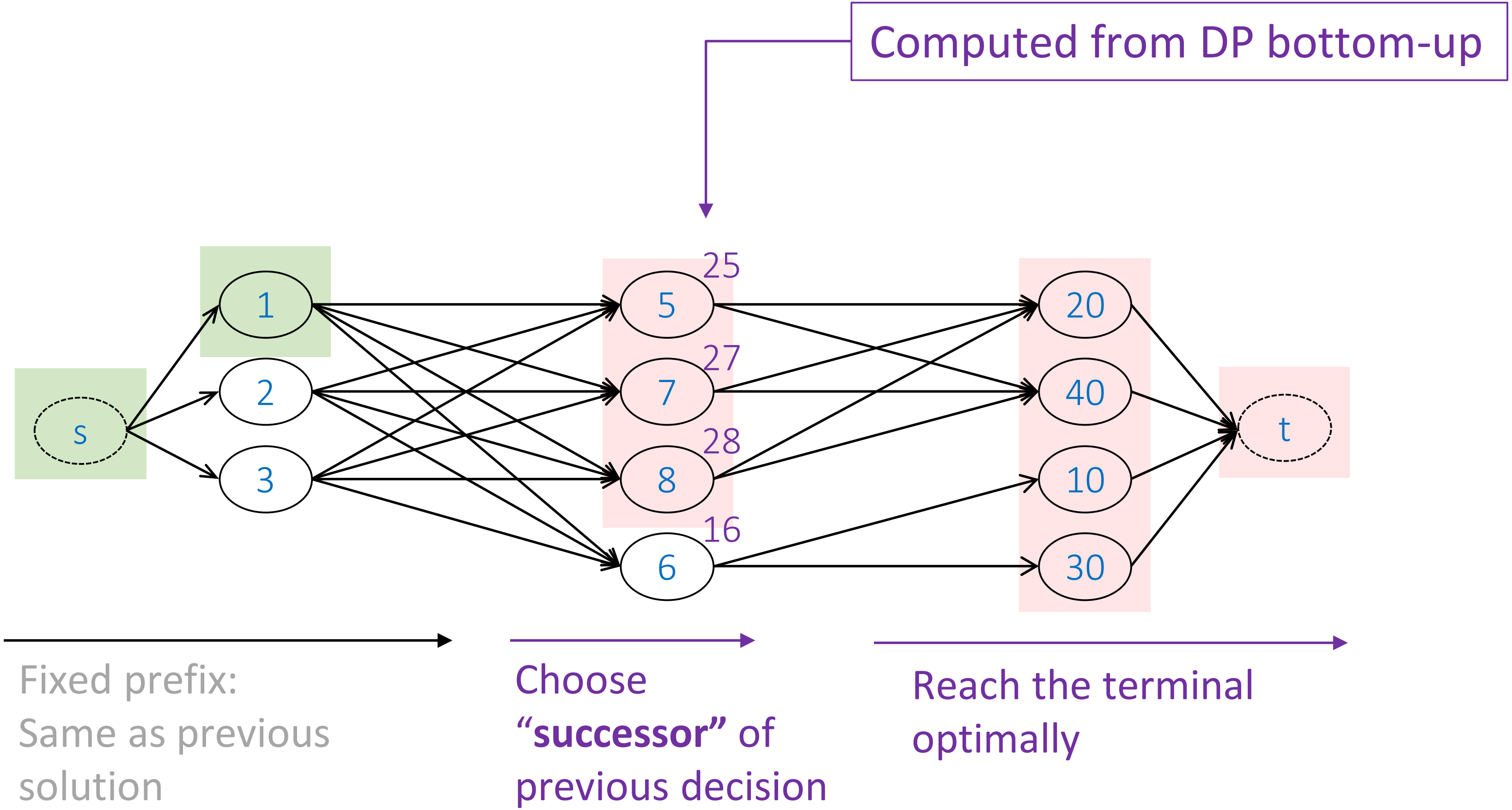


Fixed prefix: Same as previous solution

Choose "**successor**" of previous decision

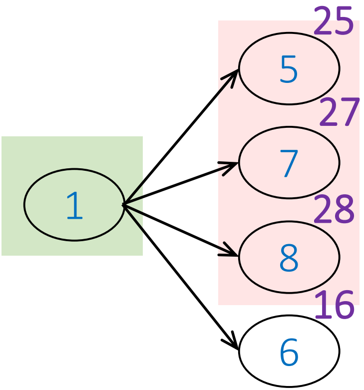
Reach the terminal optimally

Anyk-Part: Exploiting the DP structure



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

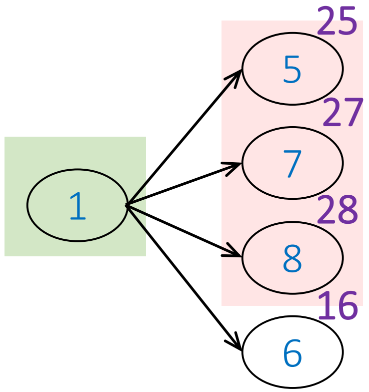


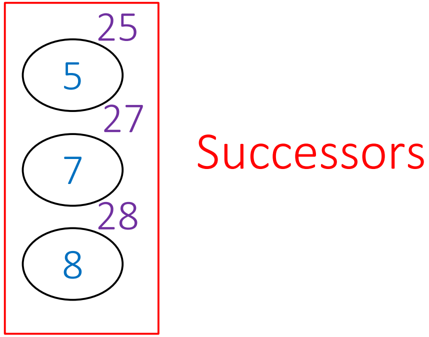
What is the successor of 6?

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

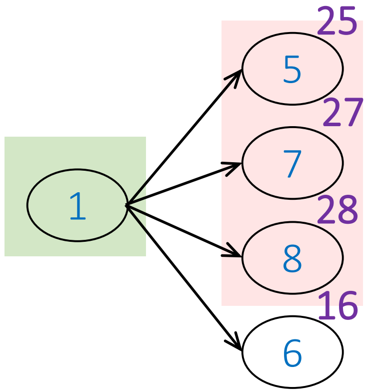


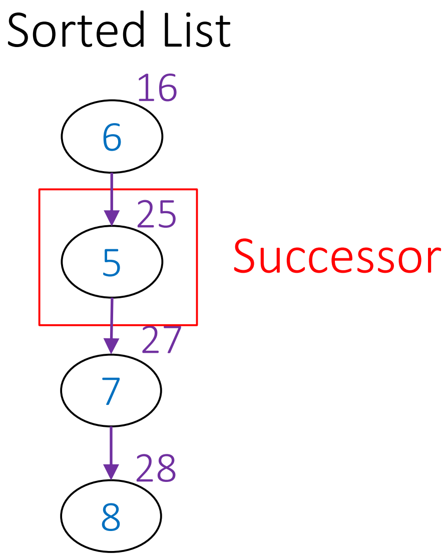


[Yang+ 18]: Yang, Ajwani, Gatterbauer, Nicholson, Riedewald, Sala. Any-k: Anytime Top-k Tree Pattern Retrieval in Labeled Graphs. WWW'18
https://doi.org/https://doi.org/10.1145/3178876.3186115

Anyk-Part Variant 2: "Eager"

- Invest more into pre-processing to get a lower delay
- Sort the decisions and find the true successor



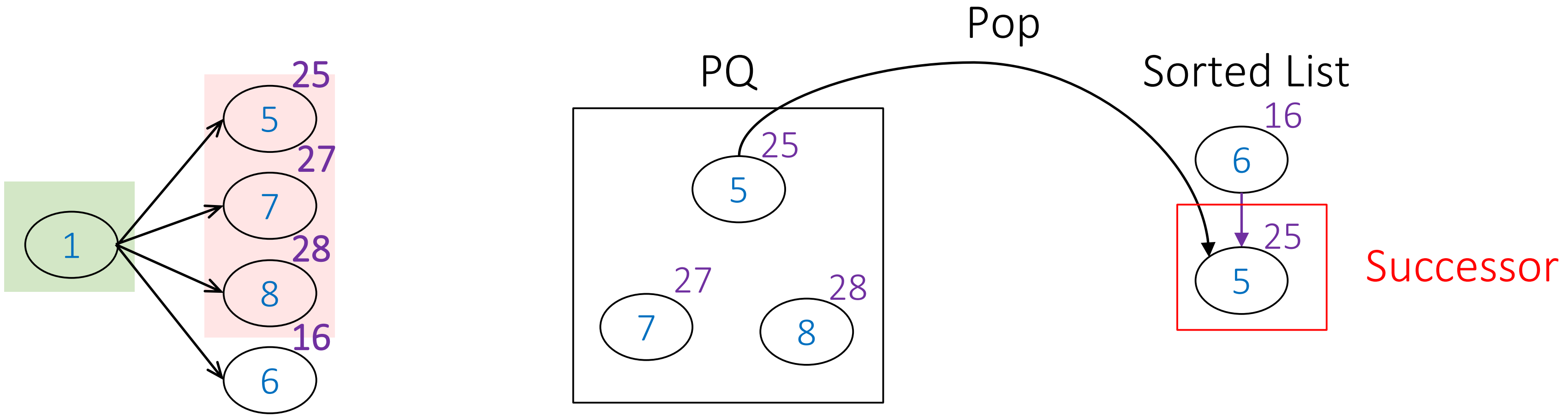


Anyk-Part Variant 3: "Lazy"

- Sorting = wasted effort if enumeration is stopped early

[Chang+ 15]:

- sort incrementally with a priority queue (per node)
- store order for future reusage



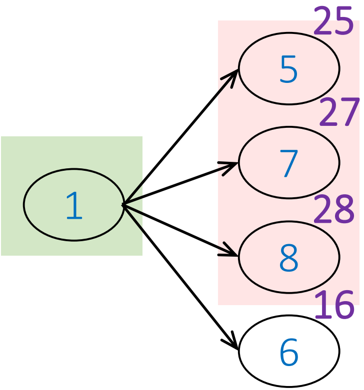
[Chang+ 15] Chang, Lin, Zhang, Yu, Zhang, Qin. Optimal enumeration: Efficient top-k tree matching. PVLDB'15 https://doi.org/10.14778/2735479.2735486

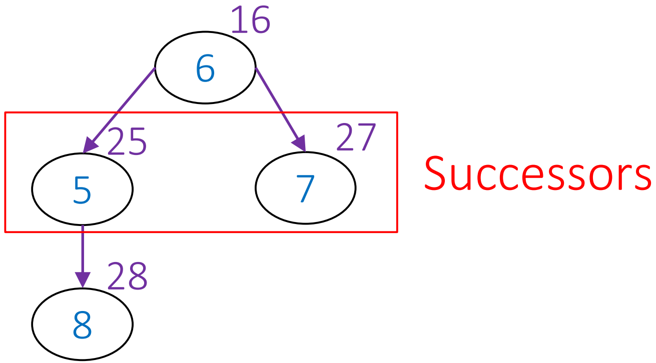
Anyk-Part Variant 4: “Take2”

- 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)





[Tziavelis+ 20] 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

Anyk-Part Complexity

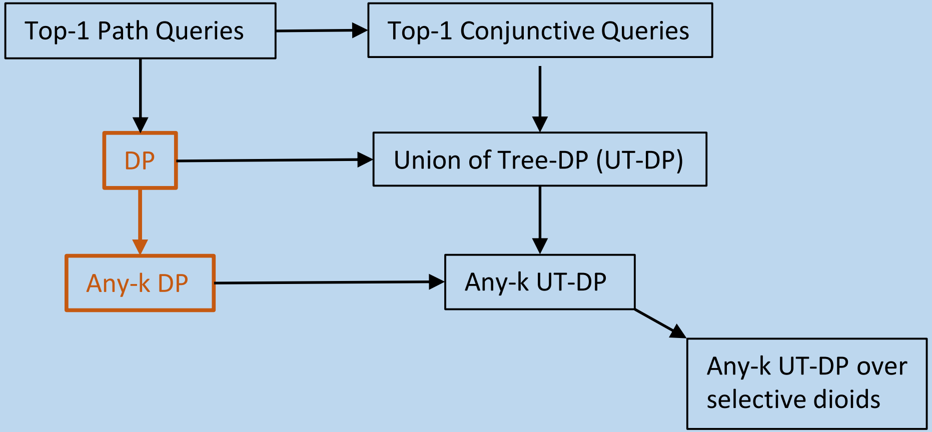
- $O(\ell\, n)$ same as DP bottom-up
- $O(k\, log k)$ same as sorting k objects
- $O(k\, \ell)$ needed to enumerate each result

Algorithm	TT(k)
Take2	$\mathcal{O}(\ell n + k(\log k + \ell))$
Lazy	$\mathcal{O}(\ell n + k(\log k + \ell + \log n))$
Eager	$\mathcal{O}(\ell n \log n + k(\log k + \ell))$
All	$\mathcal{O}(\ell n + k(\log k + \ell n))$

(*) assuming constant-time lookup with hashing

Outline tutorial

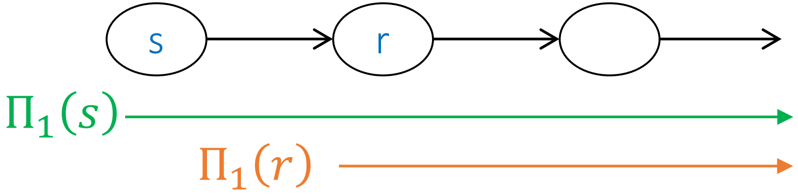
- Part 1: Top-k (Wolfgang): ~20min
- Part 2: Optimal Join Algorithms (Mirek): ~30min
- Part 3: Ranked enumeration over Joins (Nikolaos): ~40min
  - Ranked Enumeration
  - Top-1 Result for Path Queries
  - From Top-1 to Any-k
    - Anyk-Part
    - Anyk-Rec
  - Beyond Path Queries
  - Ranking Function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



Anyk-Rec: Motivation

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

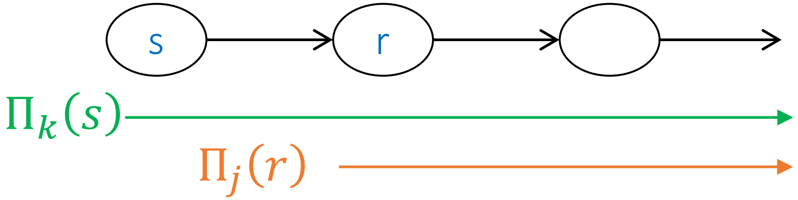
Principle of Optimality (DP)



If $\Pi_1(s)$ begins with node r then

$$\Pi_1(s) = s \circ \Pi_1(r)$$

Generalized Principle of Optimality



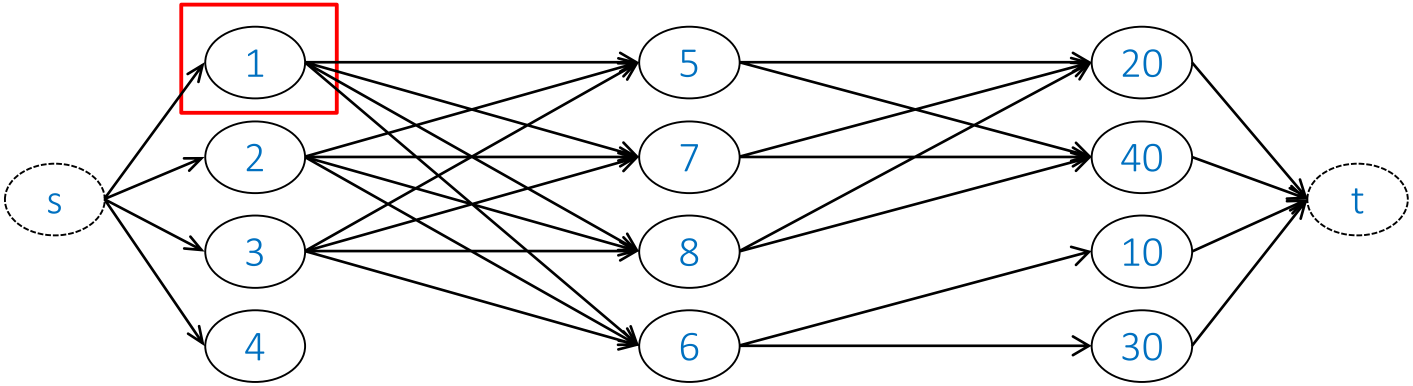
If $\Pi_k(s)$ begins with node r then

$\Pi_k(s) = s \circ \Pi_j(r)$ for some $j \leq k$

Martins, Pascoal, Santos. A new improvement for a K shortest paths algorithm. Investigação Operacional'01 http://apdio.pt/documents/10180/15407/IOvol21n1.pdf

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

$\Pi_1(1)$

$\downarrow$

$\Pi_2(1)$

$\downarrow$

$\Pi_3(1)$

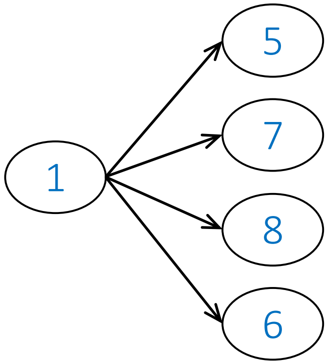
$\downarrow$

Anyk-Rec: Example

PQ

5	$1 \circ \Pi_1(5)$ [25]
7	$1 \circ \Pi_1(7)$ [27]
8	$1 \circ \Pi_1(8)$ [28]
6	$1 \circ \Pi_1(6)$ [16]

One entry per outgoing edge



Sorted List

Initially
Empty

Stores ordering of suffixes

Anyk-Rec: Example

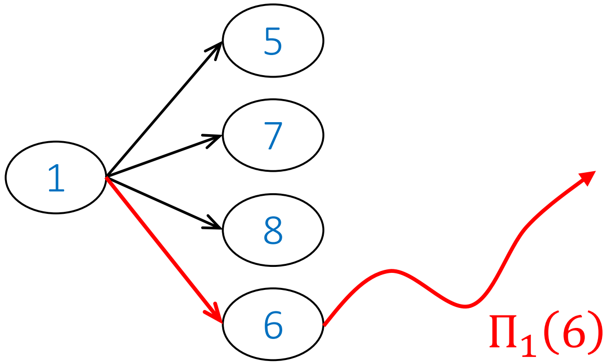
PQ

5	$1 \circ \Pi_1(5)$ [25]
7	$1 \circ \Pi_1(7)$ [27]
8	$1 \circ \Pi_1(8)$ [28]
6	

Pop

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

Sorted List



Anyk-Rec: Example

PQ

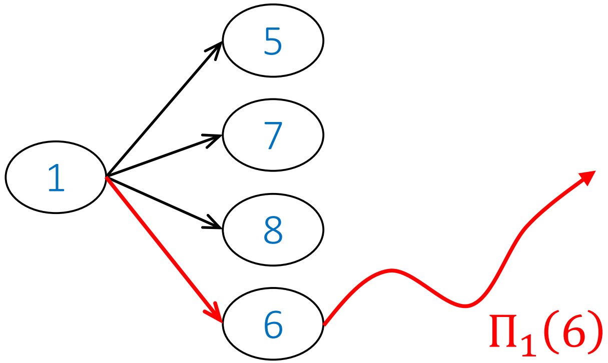
5	$1 \circ \Pi_1(5)$ [25]
7	$1 \circ \Pi_1(7)$ [27]
8	$1 \circ \Pi_1(8)$ [28]
6	

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

Store

Sorted List

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



Anyk-Rec: Example

PQ

5	$1 \circ \Pi_1(5)$ [25]
7	$1 \circ \Pi_1(7)$ [27]
8	$1 \circ \Pi_1(8)$ [28]
6	$1 \circ \Pi_2(6)$ [36]

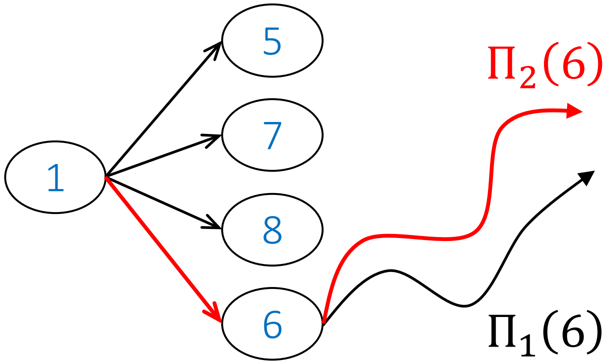
Replace

$1 \circ \Pi_2(6)$ [36]

Computed **recursively**

Sorted List

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



Anyk-Rec: Example

PQ

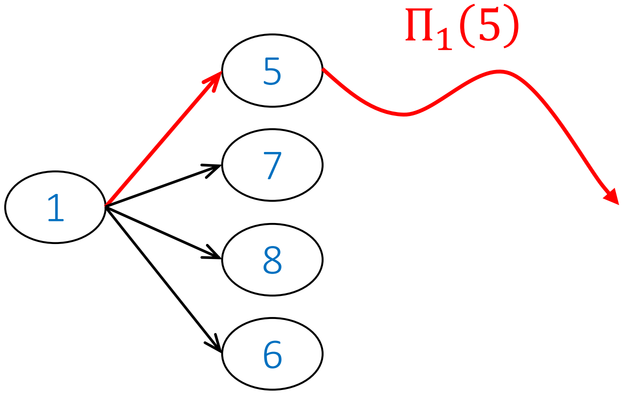
5	
7	$1 \circ \Pi_1(7)$ [27]
8	$1 \circ \Pi_1(8)$ [28]
6	$1 \circ \Pi_2(6)$ [36]

Pop

$1 \circ \Pi_1(5)$ [25]

Sorted List

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



Anyk-Rec: Example

PQ

5	
7	$1 \circ \Pi_1(7)$ [27]
8	$1 \circ \Pi_1(8)$ [28]
6	$1 \circ \Pi_2(6)$ [36]

$1 \circ \Pi_1(5)$ [25]

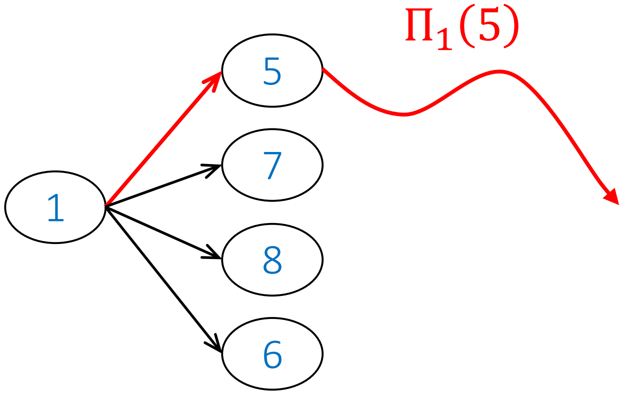
Store

Sorted List

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

$\downarrow$

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



Anyk-Rec: Example

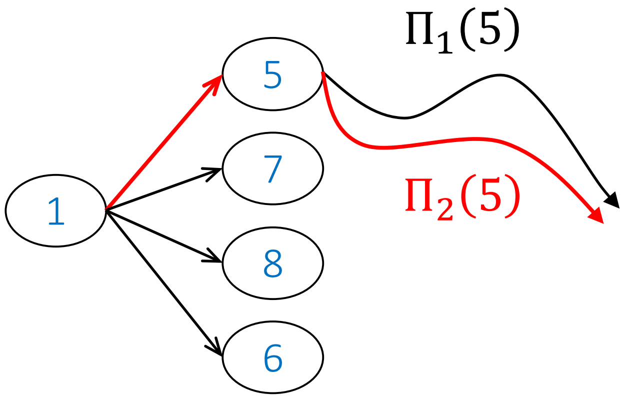
PQ

5	$1 \circ \Pi_2(5)$	[45]
7	$1 \circ \Pi_1(7)$	[27]
8	$1 \circ \Pi_1(8)$	[28]
6	$1 \circ \Pi_2(6)$	[36]

Replace

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

Computed **recursively**



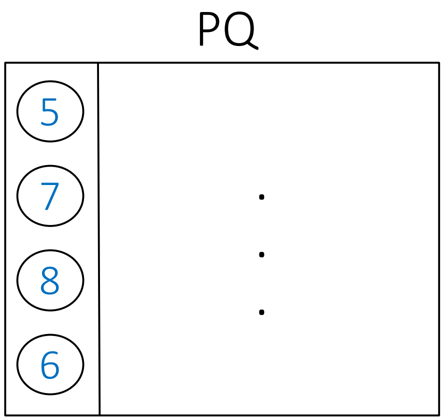
Sorted List

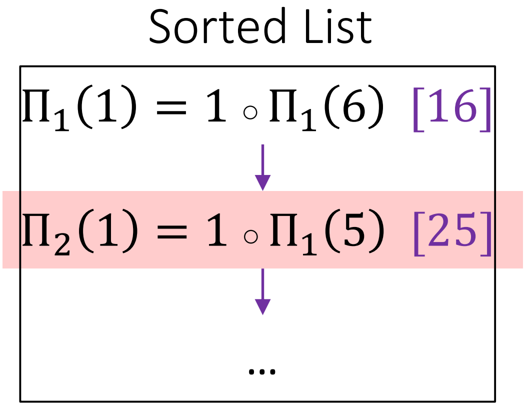
$\Pi_1(1) = 1 \circ \Pi_1(6)$ [16]

$\downarrow$

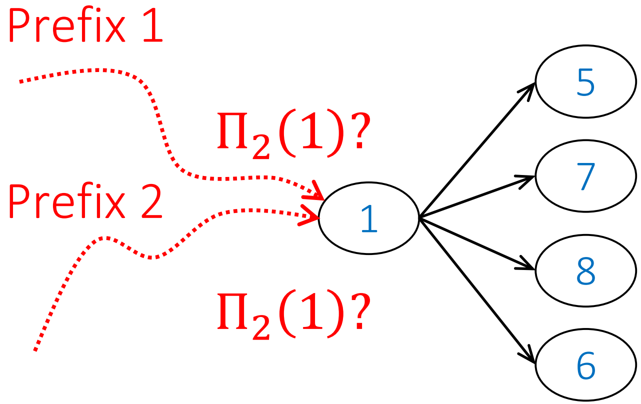
$\Pi_2(1) = 1 \circ \Pi_1(5)$ [25]

Anyk-Rec: Suffix Reusage





Later…



Reuse $\Pi_2(1)$ 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^ℓ results:
  - Anyk-Rec TTL: $O\left(n^\ell(\log n + \ell)\right)$
  - Sorting the output: $O\left(n^\ell \log n \cdot \ell\right)$

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

[Bellman+ 60] Bellman and Kalaba. "On k th best policies". JSIAM'60 https://doi.org/10.1137/0108044

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

[Jiménez+ 99] Jiménez, Marzal. Computing the k shortest paths: A new algorithm and an experimental comparison. WAE'99 https://doi.org/10.1007/3-540-48318-7_4

[Deep+ 19] Deep and Koutris. Ranked enumeration of conjunctive query results. ArXiv'19 http://arxiv.org/abs/1902.02698

[Tziavelis+ 20] 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

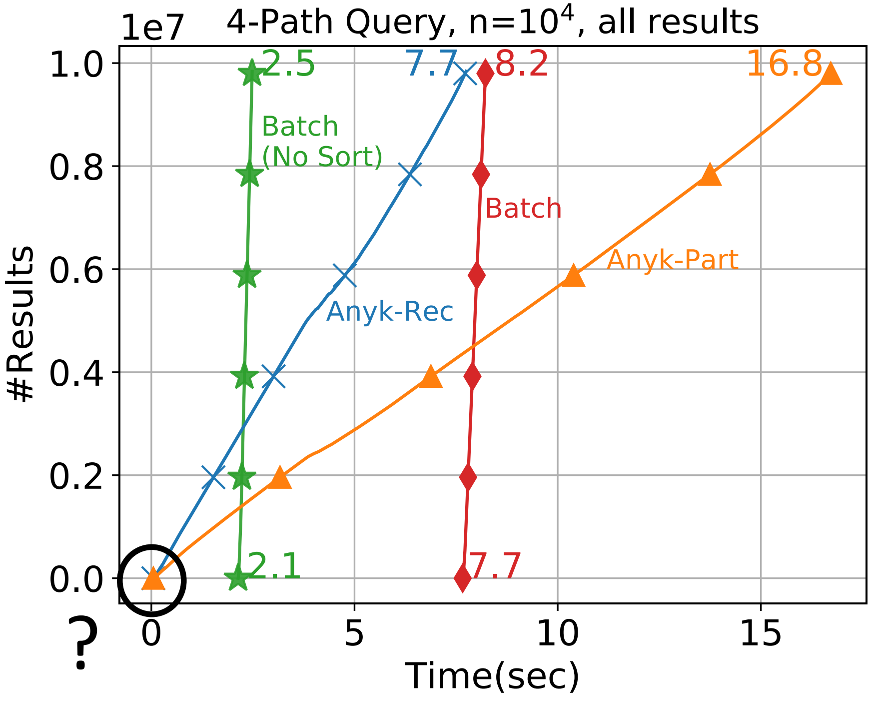
Overview

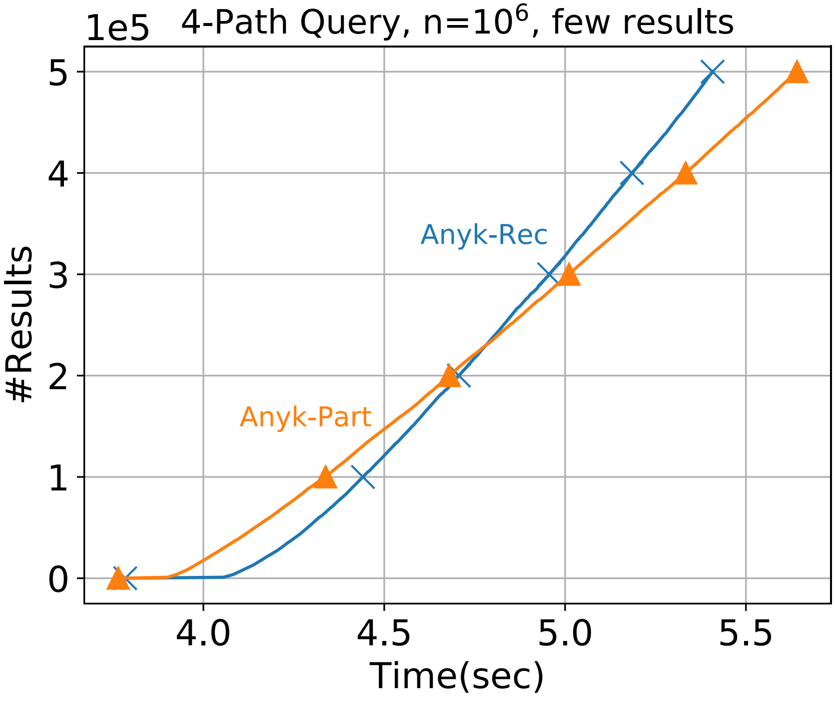
- Take2 has lower complexity over all instances
- But there are cases where the recursive approach wins for TTL

Algorithm	TT(k)	TTL for $\lvert\text{out}\rvert = \Omega(\ell n)$	TTL for $\lvert\text{out}\rvert = \Theta(n^{\ell})$
Recursive	$\mathcal{O}(\ell n + k\ell \log n)$	$\mathcal{O}(\lvert\text{out}\rvert \ell \log n)$	$\mathcal{O}(n^{\ell}(\log n + \ell))$
Take2	$\mathcal{O}(\ell n + k(\log k + \ell))$	$\mathcal{O}(\lvert\text{out}\rvert(\log \lvert\text{out}\rvert + \ell))$	$\mathcal{O}(n^{\ell} \cdot \ell \log n)$
Lazy	$\mathcal{O}(\ell n + k(\log k + \ell + \log n))$	$\mathcal{O}(\lvert\text{out}\rvert(\log \lvert\text{out}\rvert + \ell))$	$\mathcal{O}(n^{\ell} \cdot \ell \log n)$
Eager	$\mathcal{O}(\ell n \log n + k(\log k + \ell))$	$\mathcal{O}(\lvert\text{out}\rvert(\log \lvert\text{out}\rvert + \ell))$	$\mathcal{O}(n^{\ell} \cdot \ell \log n)$
All	$\mathcal{O}(\ell n + k(\log k + \ell n))$	$\mathcal{O}(\lvert\text{out}\rvert(\log \lvert\text{out}\rvert + \ell))$	$\mathcal{O}(n^{\ell} \cdot \ell \log n)$
Batch	$\mathcal{O}(\ell n + \lvert\text{out}\rvert(\log \lvert\text{out}\rvert + \ell))$	$\mathcal{O}(\lvert\text{out}\rvert(\log \lvert\text{out}\rvert + \ell))$	$\mathcal{O}(n^{\ell} \cdot \ell \log n)$

(*) assuming constant-time lookup with hashing

Some Experimental Results



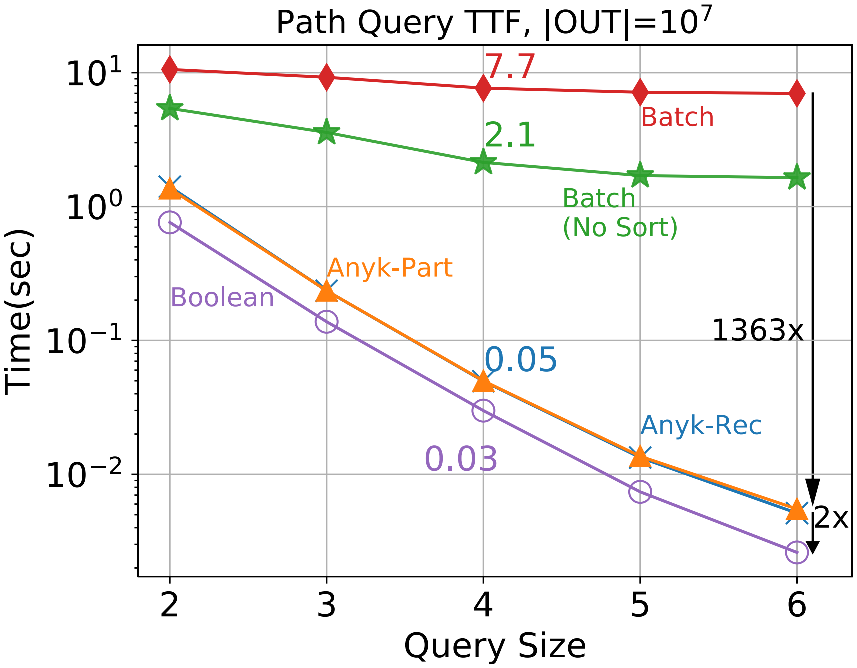


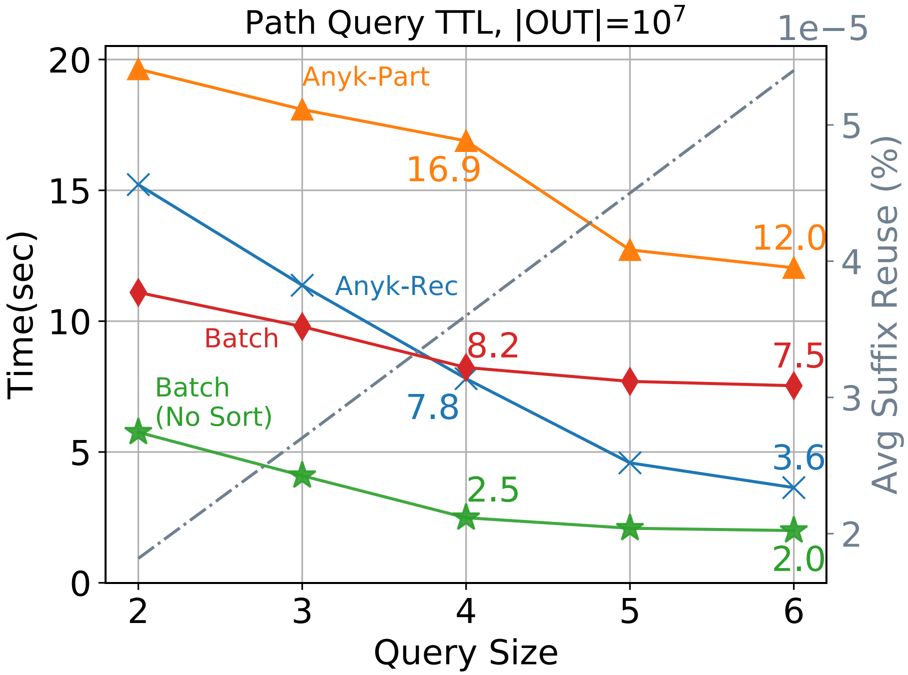
- Anyk starts much faster than Batch
- Anyk-Rec also finishes faster than Batch

- Anyk-Part is usually faster in the beginning

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



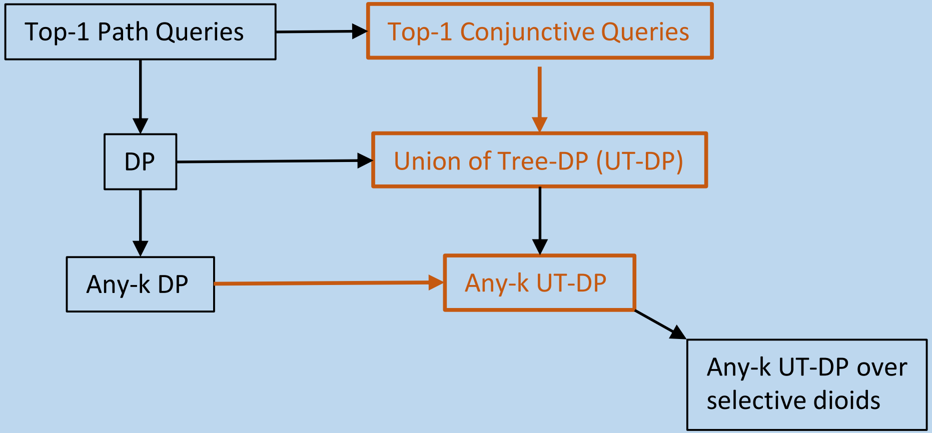


- Boolean (is there any result?) is the best we can do
- Anyk is only 2 times slower

- Anyk-Rec is getting faster when there are more opportunities for suffix reusage

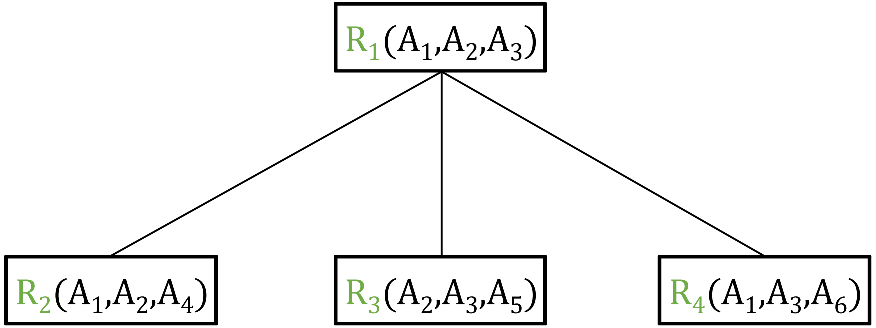
Outline tutorial

- 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



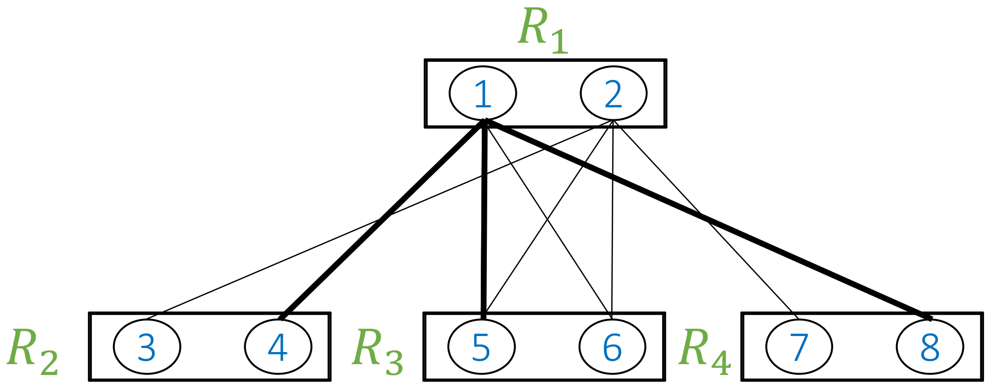
Acyclic Queries

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

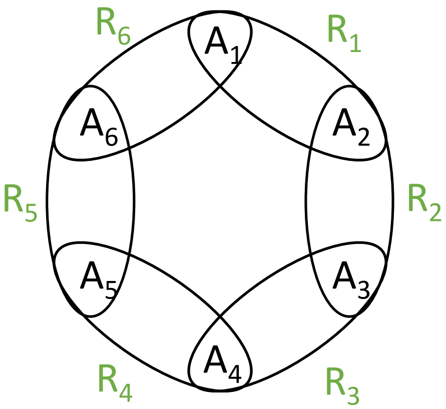
- Anyk-Rec:
  - Apply the path algorithm in each branch
  - Difficulty: how do we combine the solutions from each branch?
  - Improved TTL only if the tree has significant depth

[Deep+ 19] Deep and Koutris. Ranked enumeration of conjunctive query results. ArXiv'19 http://arxiv.org/abs/1902.02698

[Tziavelis+ 20] 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

Cyclic Queries

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



Acyclic 1: $O(n^{5/3})$

Acyclic 2: $O(n^{5/3})$

Acyclic 3: $O(n^{5/3})$

Acyclic 4: $O(n^{5/3})$

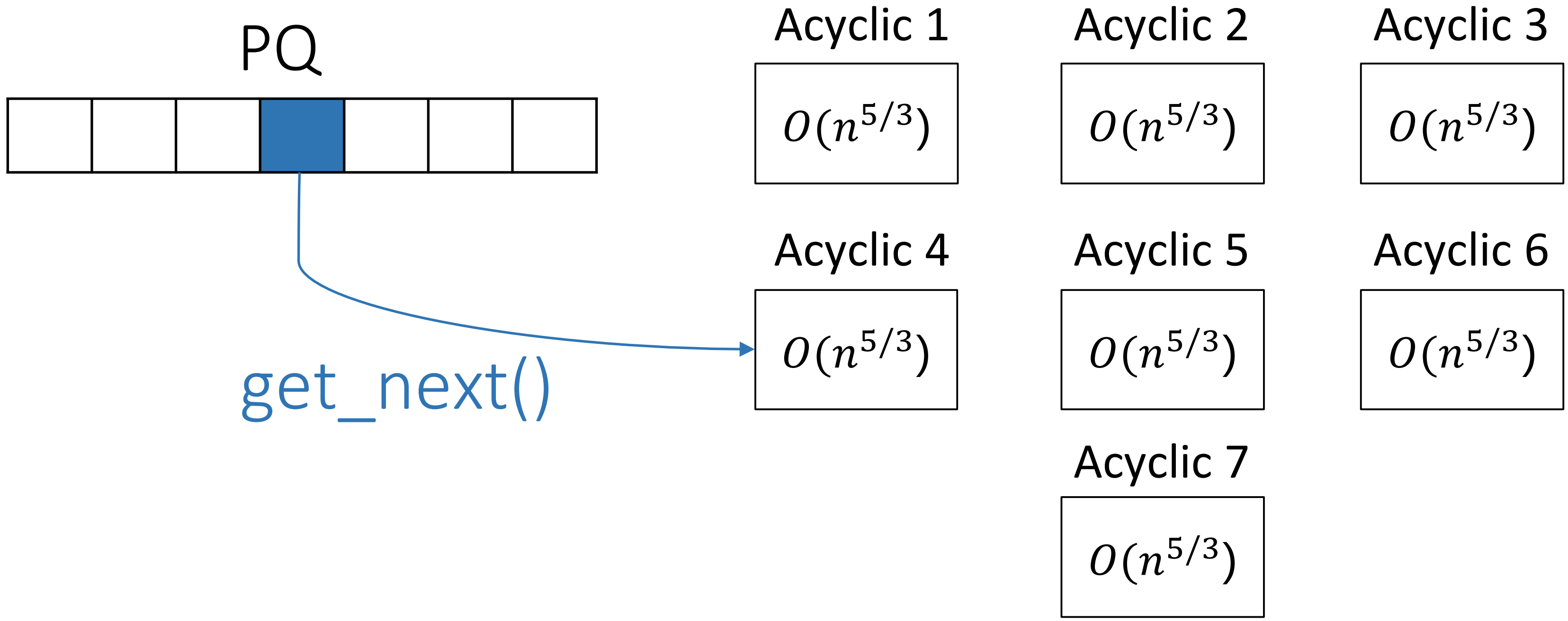
Acyclic 5: $O(n^{5/3})$

Acyclic 6: $O(n^{5/3})$

Acyclic 7: $O(n^{5/3})$

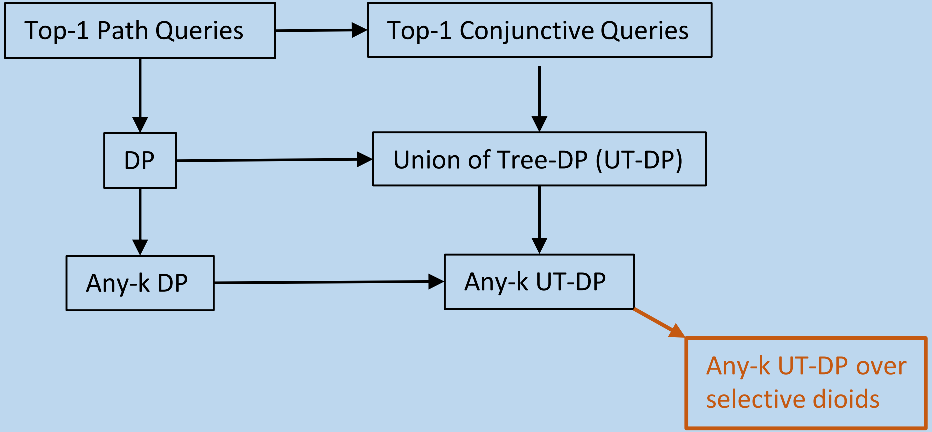
Ranked Enumeration for Cyclic Queries

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



Outline tutorial

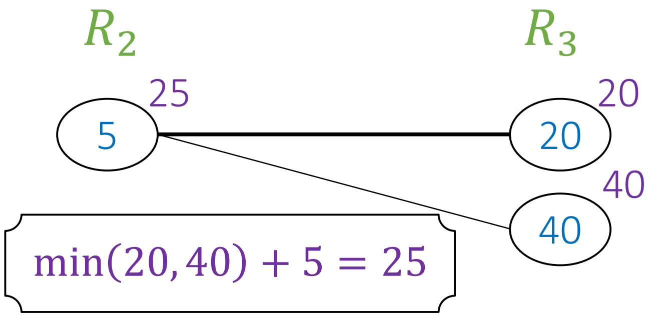
- Part 1: Top-k (Wolfgang): ~20min
- Part 2: Optimal Join Algorithms (Mirek): ~30min
- Part 3: Ranked enumeration over Joins (Nikolaos): ~40min
  - Ranked Enumeration
  - Top-1 Result for Path Queries
  - From Top-1 to Any-k
    - Anyk-Part
    - Anyk-Rec
  - Beyond Path Queries
  - Ranking Function
  - 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 1st operator has to induce an order on the domain

Semirings

- Semiring $(W,\oplus,\otimes,0,1)$
  1. $(W,\oplus,0)$ is commutative monoid
  2. $(W,\otimes,1)$ is monoid
  3. $\otimes$ distributes over $\oplus$: $(x \oplus y)\otimes z = (x \otimes z) \oplus (y \otimes z)$ ← Key property for efficiency (DP)
  4. 0 annihilates $\otimes$: $0 \otimes x = 0$
- Examples
  1. $(R^{\infty},\min,+,\infty,0)$ “Tropical semiring”
  2. $(\{0,1\},\vee,\wedge,0,1)$ Boolean
  3. $(N,+,\cdot,0,1)$ Number of paths ← No ordering (What would the 2nd best solution be?)

Aji, McEliece. The generalized distributive law. IEEE Trans. Inf’00 https://doi.org/10.1109/18.825794

Mohri. Semiring frameworks and algorithms for shortest-distance problems. JALC’02 http://www.jalc.de/issues/2002/issue_7_3/abs-321.pdf

Selective Dioids

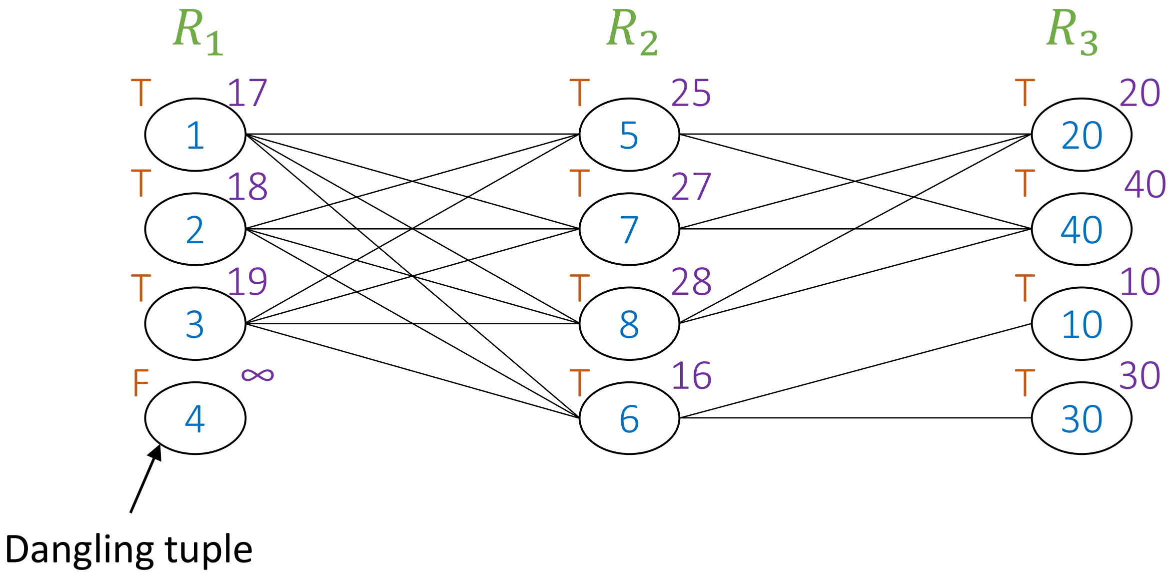
- A selective dioid $(W,\oplus,\otimes,0,1)$ is a semiring with an additional property
  - $\oplus$ is selective: $(x \oplus y = x) \vee (x \oplus y = y)$

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

Gondran, Minoux. Graphs, Dioids and Semirings: New Models and Algorithms. Springer'08. https://doi.org/10.1007/978-0-387-75450-5

DP & Yannakakis

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



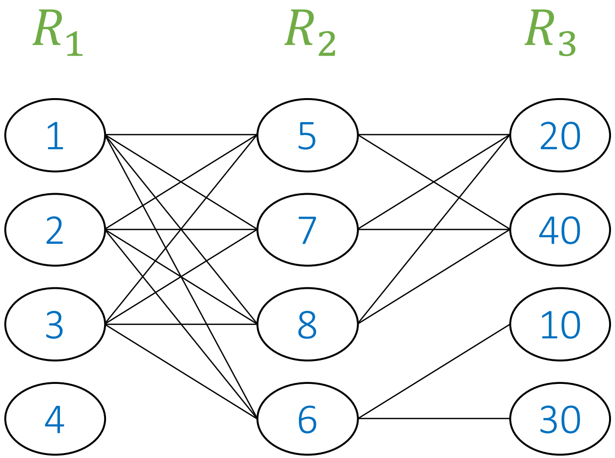
Any-k with Boolean semiring?

Equivalent to standard query evaluation of Yannakakis if we use "smarter" PQs (sorted lists of 0-1)

Yannakakis

Minimum SUM

Lexicographic Orders



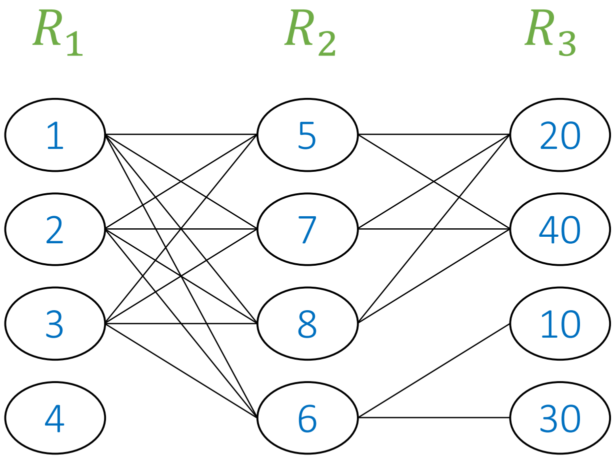
Lexicographic Order

$R_2 - R_1 - R_3$

Results first weighted on R_2 then R_1 then R_3

(5, 1, 20) ⇨ (5, 1, 40) ⇨ (5, 2, 20) ⇨ …

Lexicographic Orders



Lexicographic Order

$R_2 - R_1 - R_3$

W: ℓ-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): ~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

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?