Any-k Algorithms for Exploratory Analysis with Conjunctive Queries

Xiaofeng Yang, Mirek Riedewald, Rundong Li, Wolfgang Gatterbauer

ExploreDB @ SIGMOD 2018
(June 15, 2018)

DATALAB
@Northeastern

Northeastern University

Summary Paper

- Based on our 2018 WWW paper in collaboration with Bell Labs, Ireland:

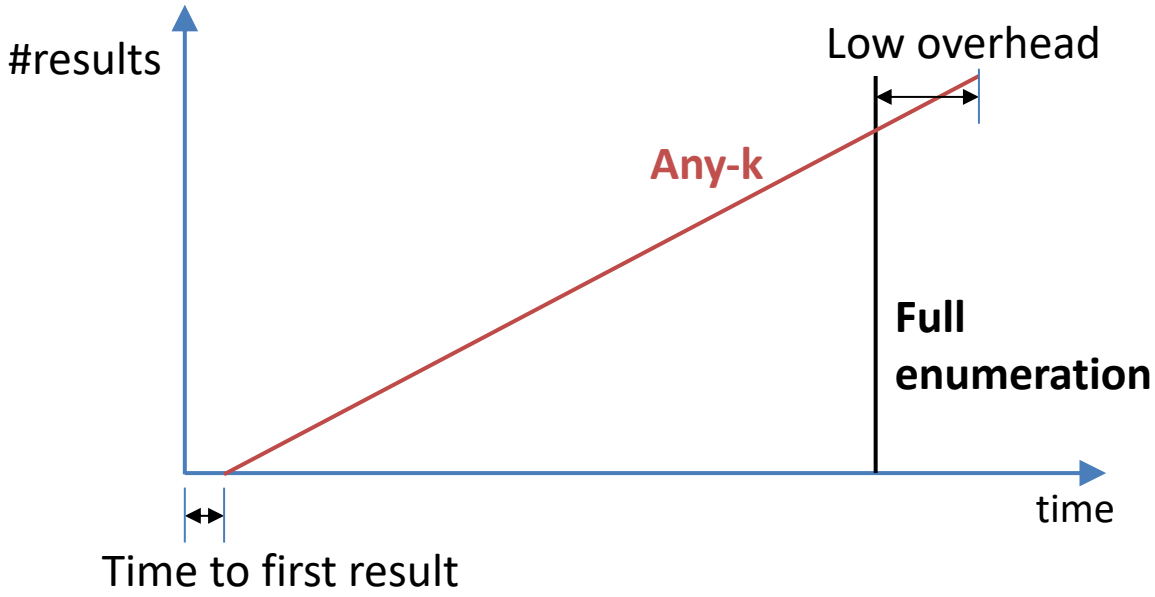
Xiaofeng Yang, Deepak Ajwani, Wolfgang Gatterbauer, Patrick K. Nicholson, Mirek Riedewald, Alessandra Sala. Any-k: Anytime Top-k Tree Pattern Retrieval in Labeled Graphs. WWW 2018: 489-498 (https://arxiv.org/abs/1802.06060)

Top-k Queries for Exploration

- Cheaper than finding all results
- Problem: how to set k
  - "When will I have seen enough?"
- Solution: anytime ranking algorithm
  - When stopped, produce top-i results, for largest possible i
  - Not easy: top-k algorithms typically exploit knowledge of k for pruning.

What's New about Any-k?

- **Non-trivial guarantees** for
  - Time to return top-1 result
  - Time between results
  - Time for full enumeration
  - Space requirement
- Problem types:
  - Subgraph isomorphism
  - Conjunctive queries (SPJ query with conjunctions of selections only)
    - SELECT * FROM R, S, T WHERE R.B = S.B AND R.C = T.C



Ideal Guarantees

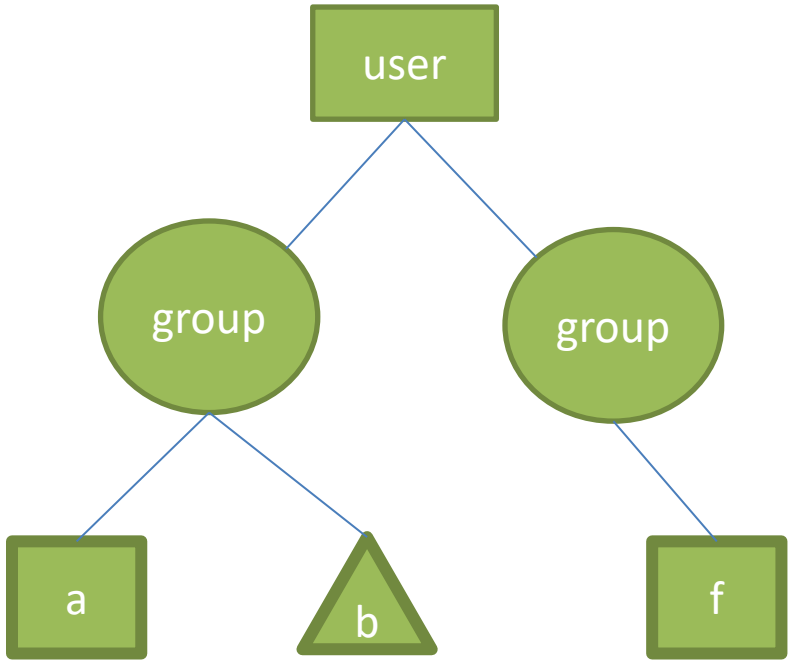
- Time to top-1 vs. O(time for boolean query)
  - Boolean query: *is there any result*?
  - Top-1 is at least as hard as boolean
  - "Get top-1, but maybe more" is at least as hard as top-1
- Time to all vs. O(resultSize · log resultSize)
  - Trivial lower bound (comparison-based sorting); may be loose
- Space: ???

Current Results: Any-k for Pattern Retrieval in Graphs

A Complex Labeled Graph G

e.g. User-Photo interactions on Flickr, Enron email network

Pattern Query Q

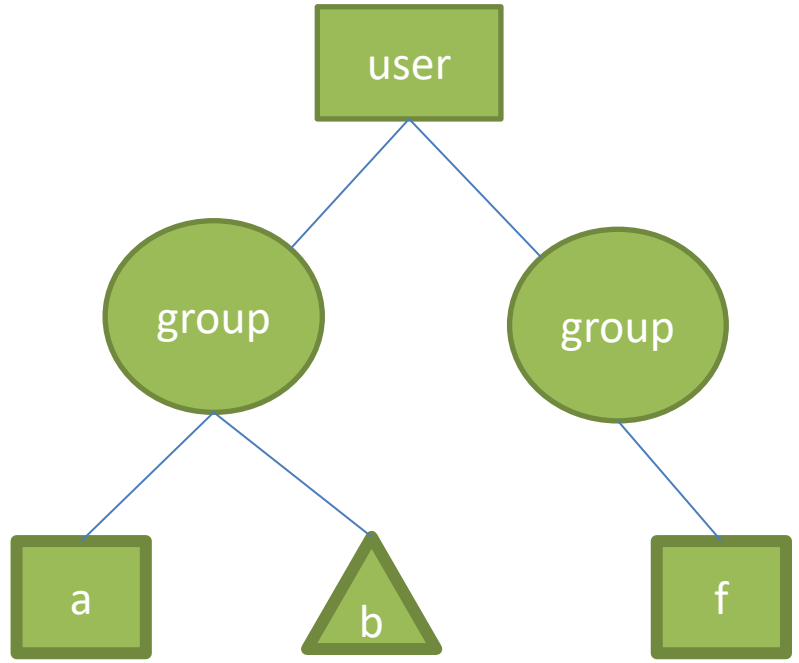


Find the *lightest* isomorphic subgraphs matching query Q in an Labeled Graph G

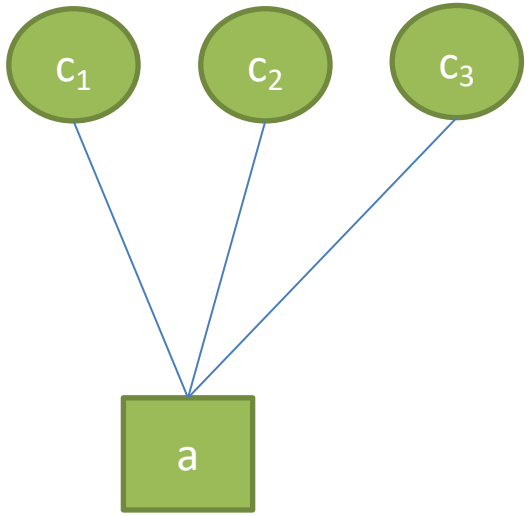
Additional information: a, b, and f are known

Why is this hard?

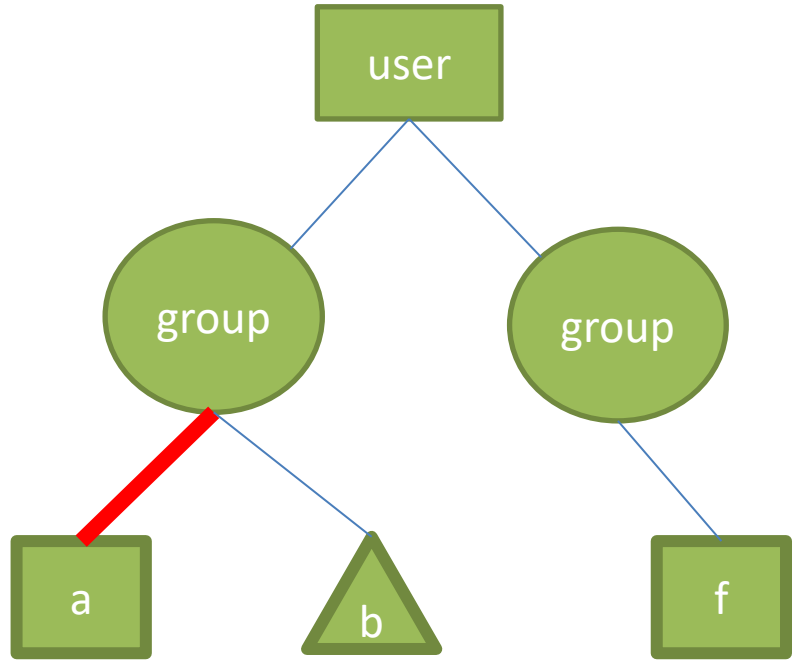
Candidate Graph



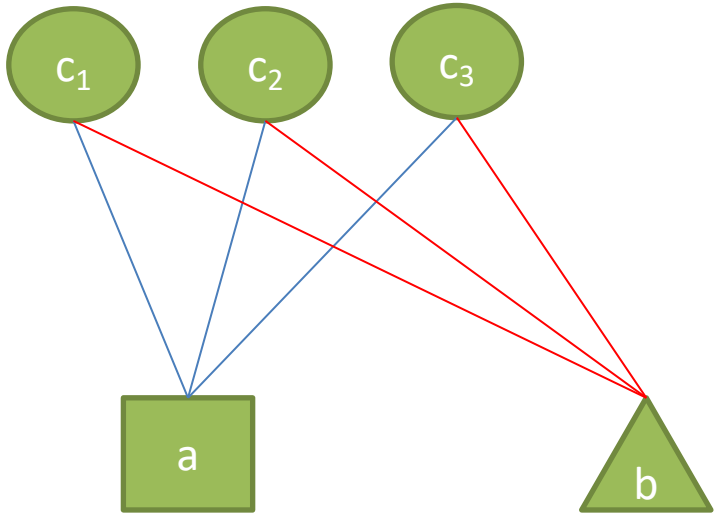
Query



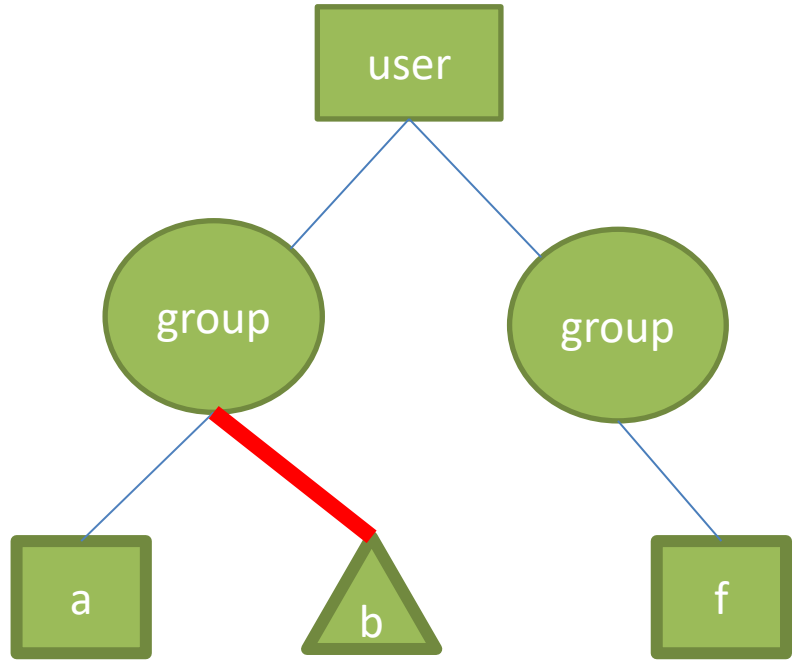
Candidate Graph



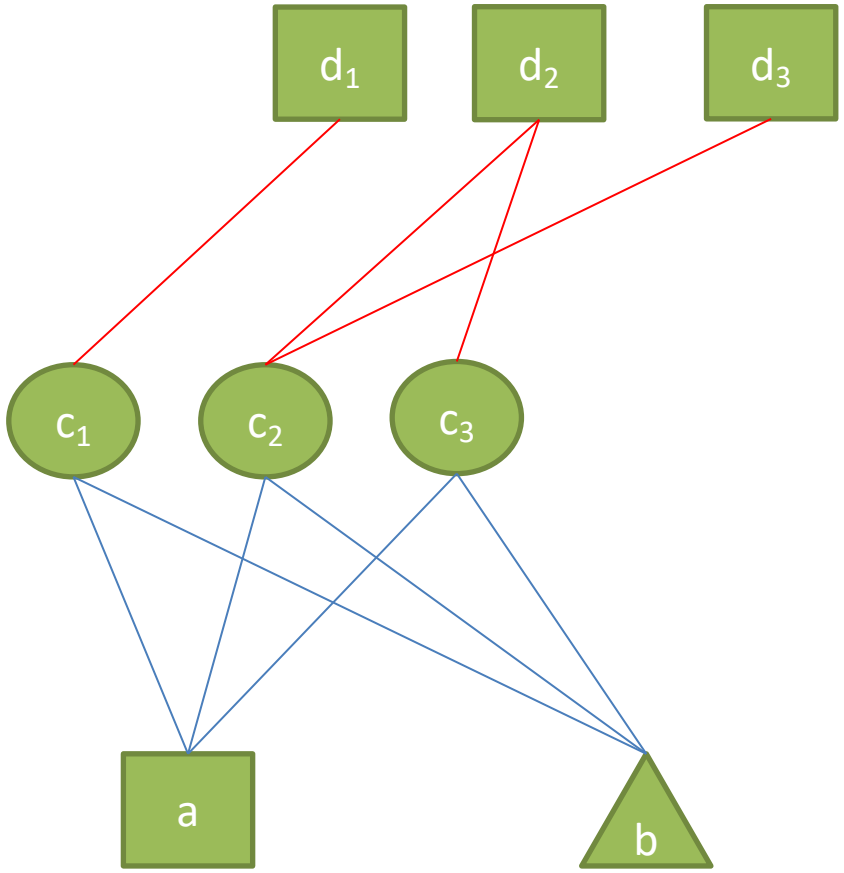
Query



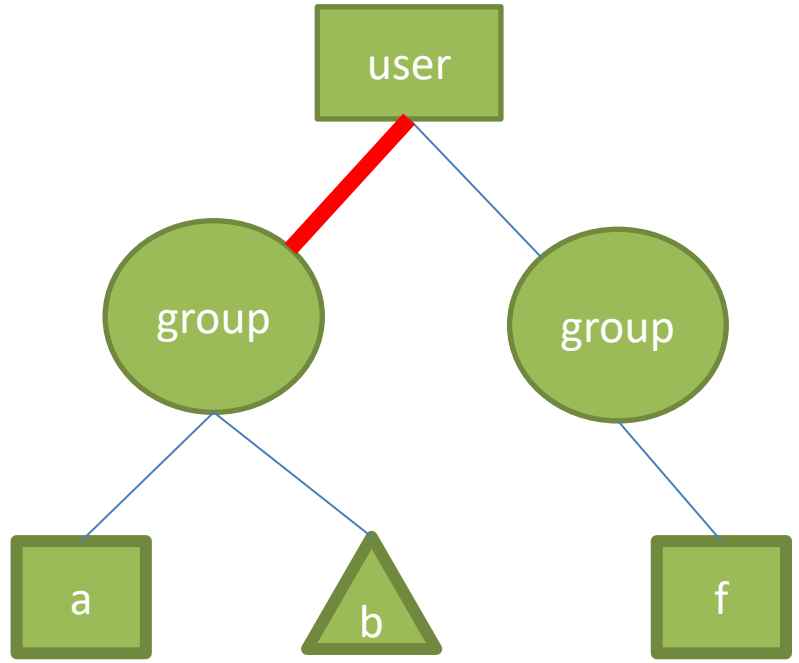
Candidate Graph



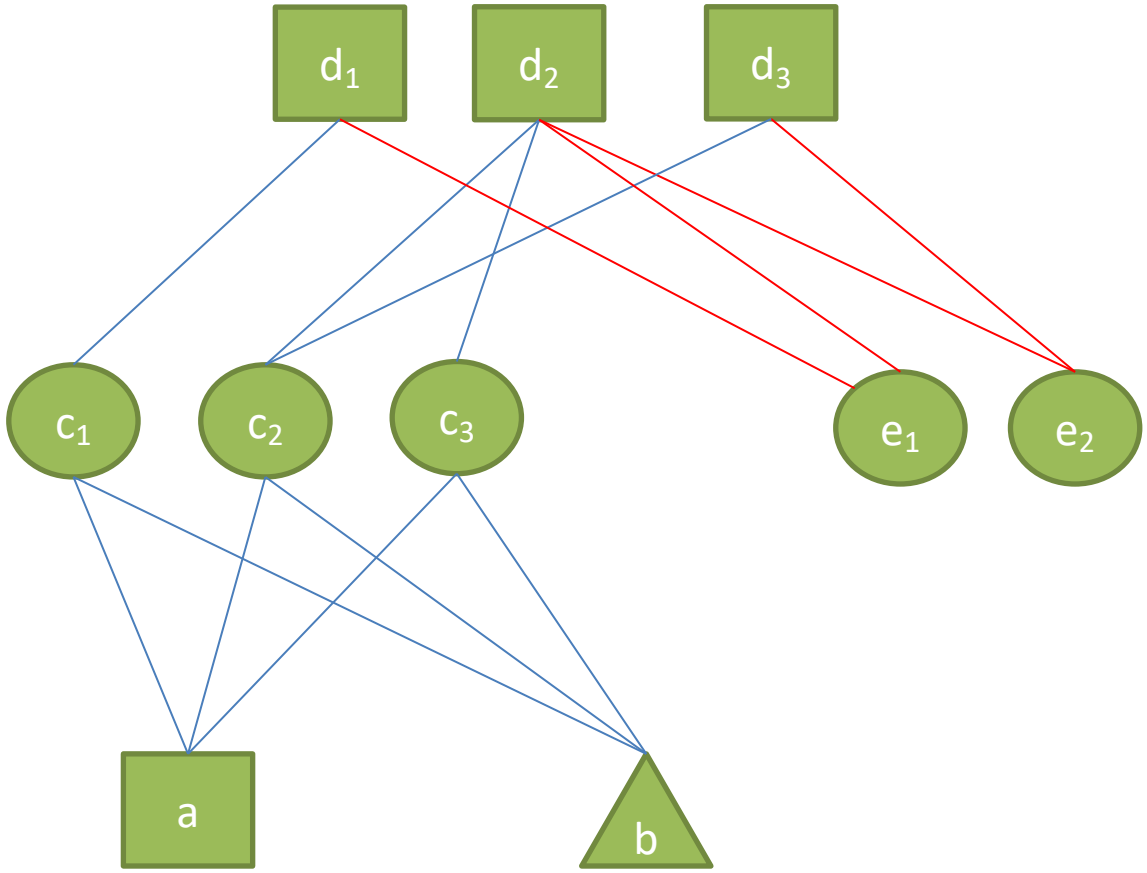
Query



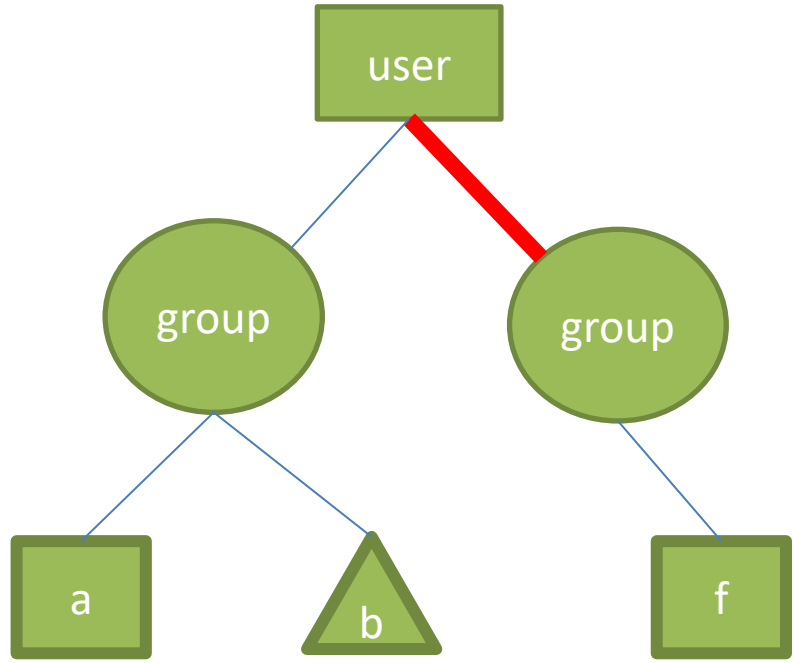
Candidate Graph



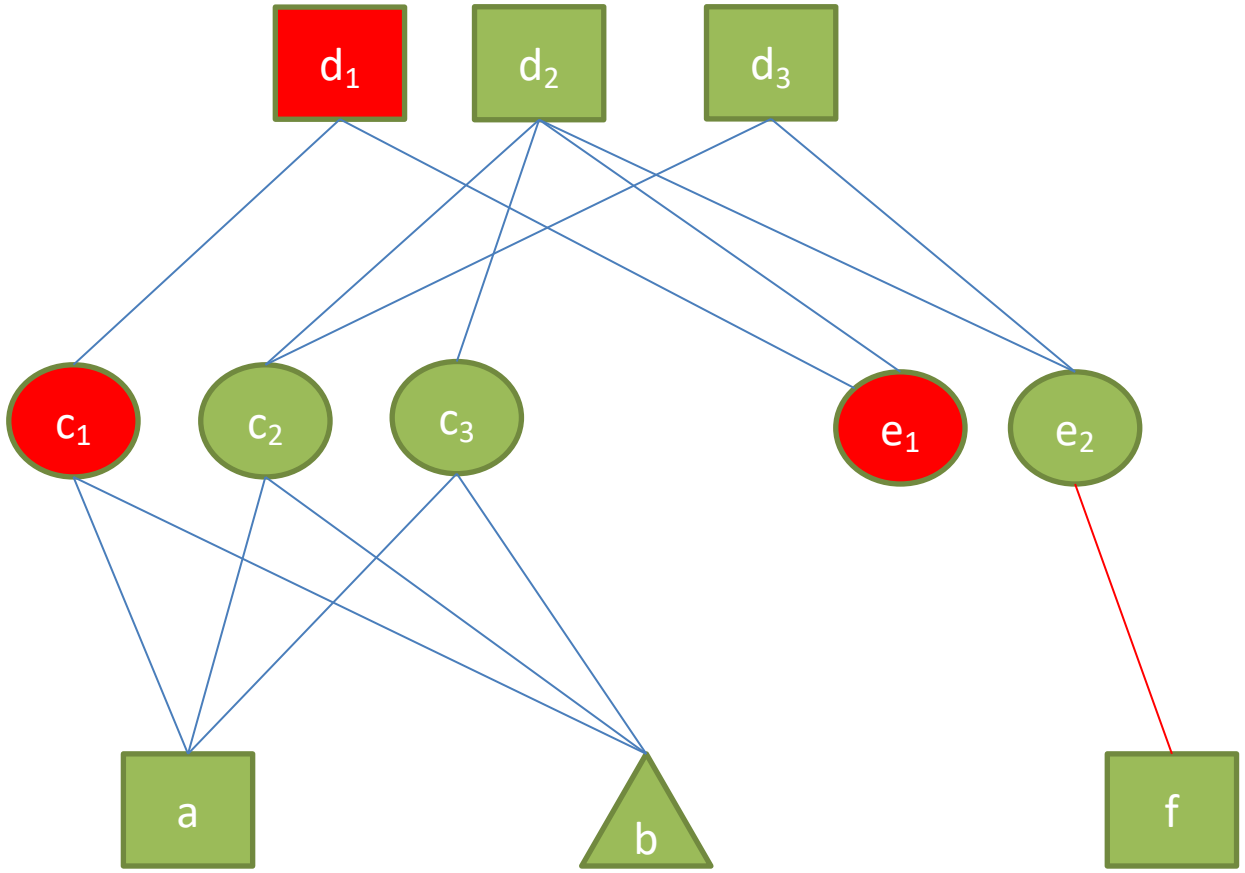
Query



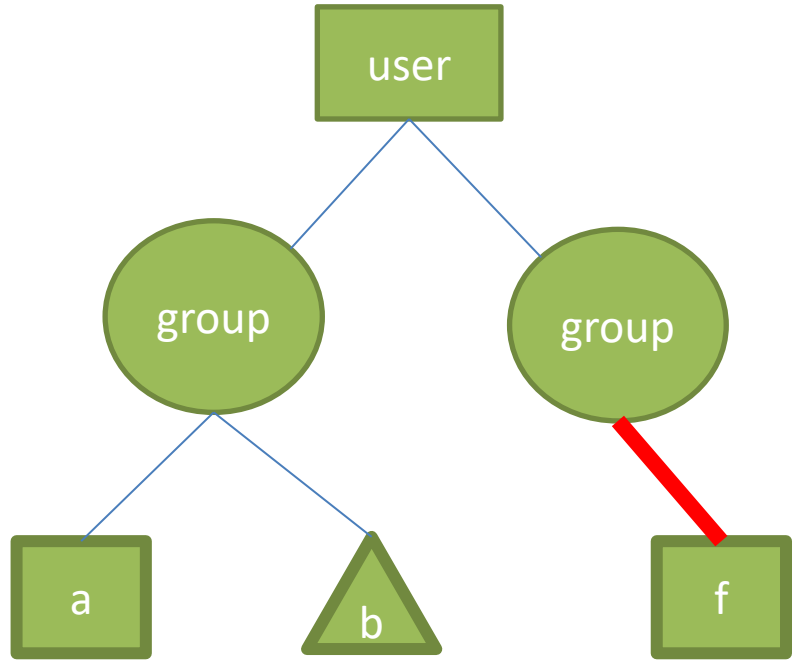
Candidate Graph



Query



Candidate Graph



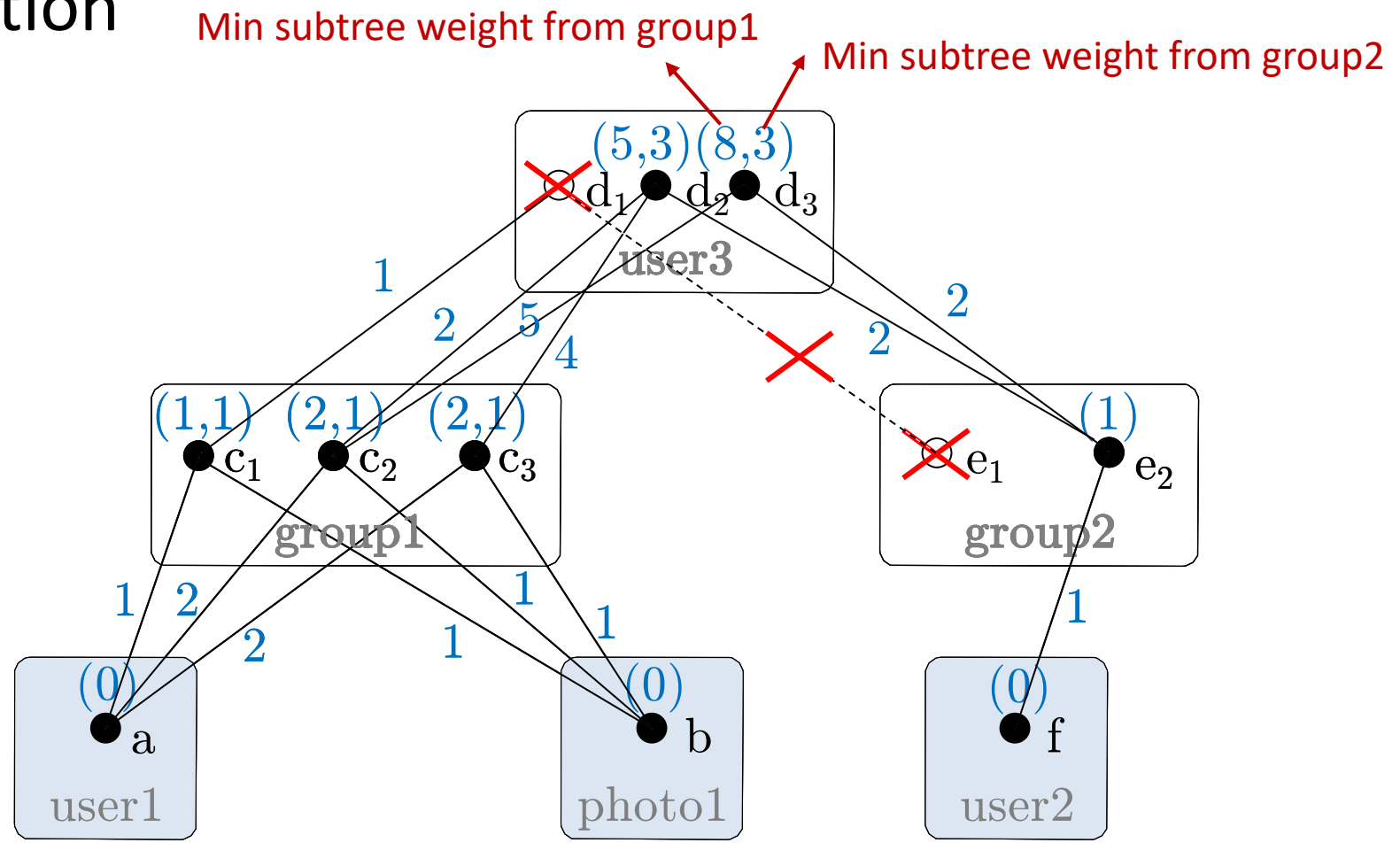
Query

Can we early on avoid (1) spurious nodes and (2) heavy results?

Algorithm Overview

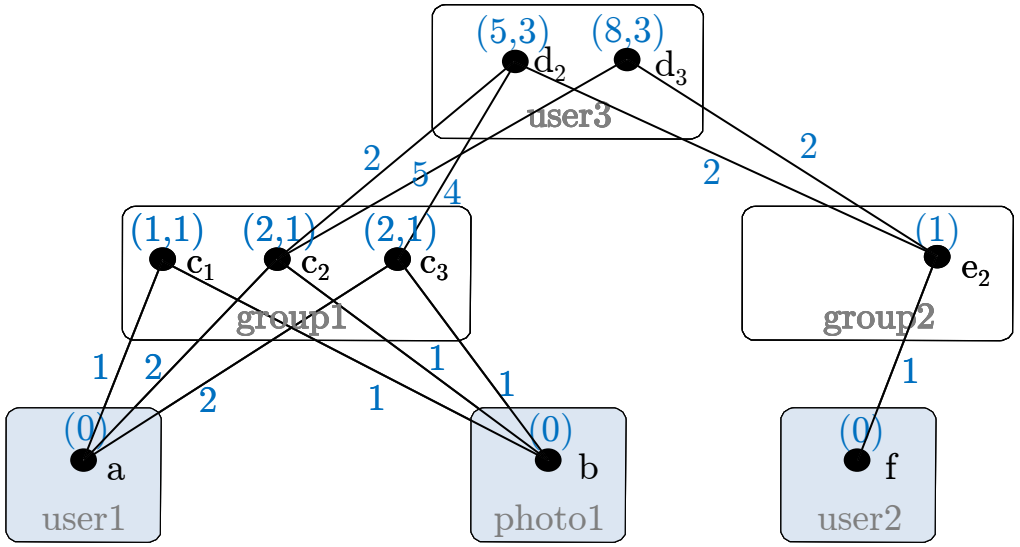
- **Classic BFS approach:** $O(|E|^{|Q|})$ is exponential in query size!
- **Our approach:** two sweeps that prune spurious nodes in $O(|E| \cdot |Q|)$
  - Bottom-up, then top-down
- **Advantage:** Simple 1-hop neighborhood look-ups, no assembly of patterns
  - Avoids combinatorial complexity
  - Note: theoretical guarantees are for homomorphism, instead of isomorphism

Step1: bottom-up semi-join reduction



Next step: top-down traversal, guided by subtree weights

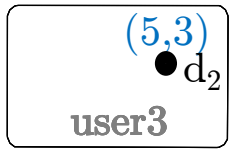
Step2: top-down



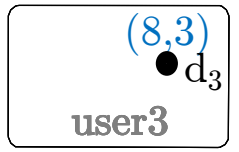
Candidate graph with subtree weights

Output

Priority queue



Key=8

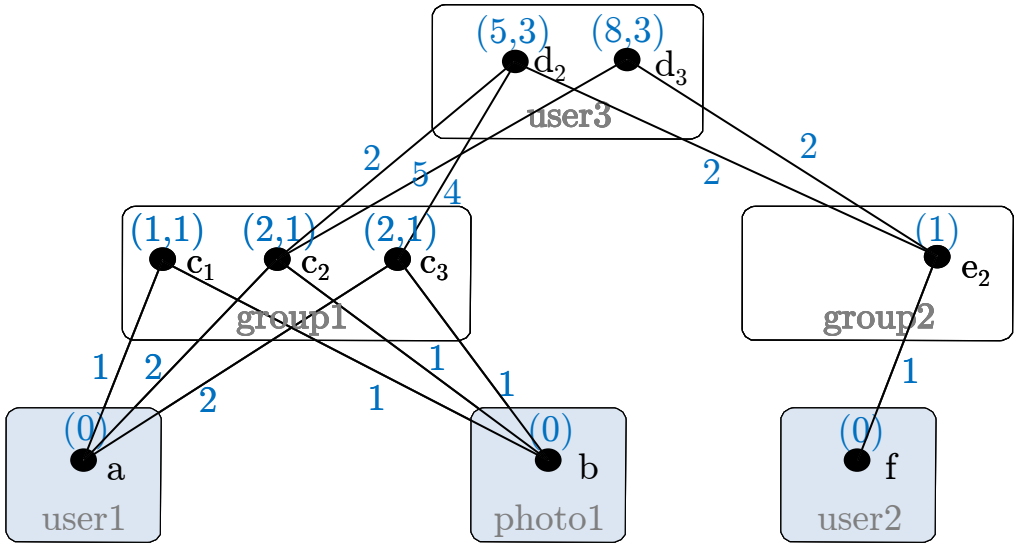


Key=11

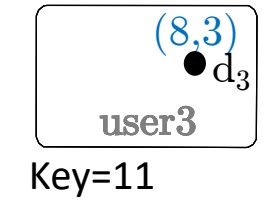
Partial Matches

push	push	pop

Priority queue



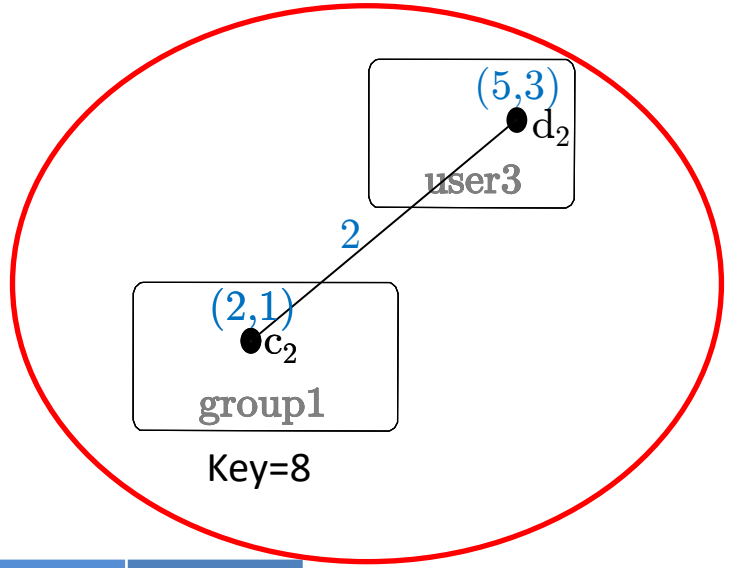
Candidate graph with subtree weights

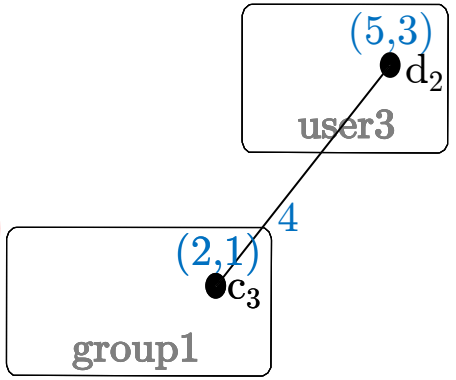


Key=11

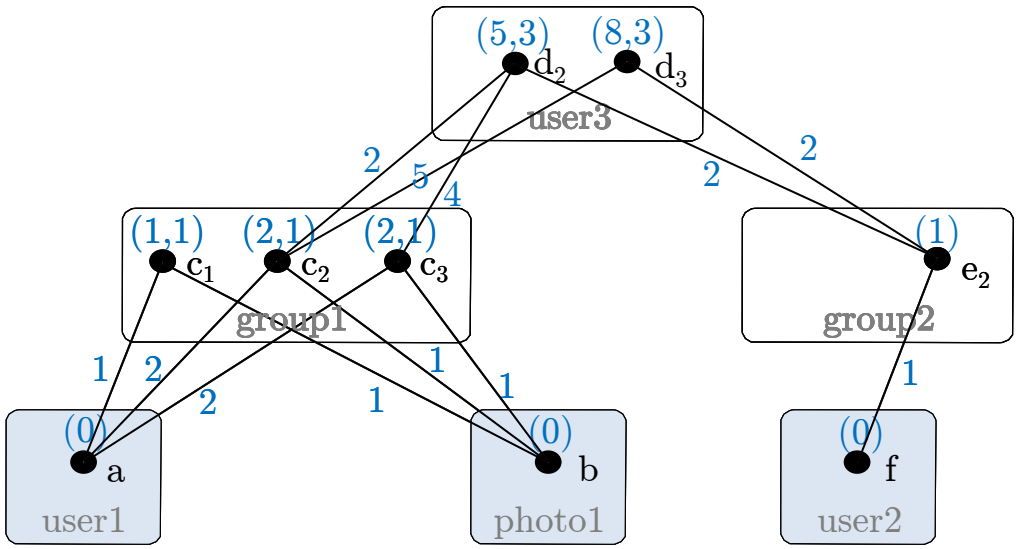
Output

Partial Matches





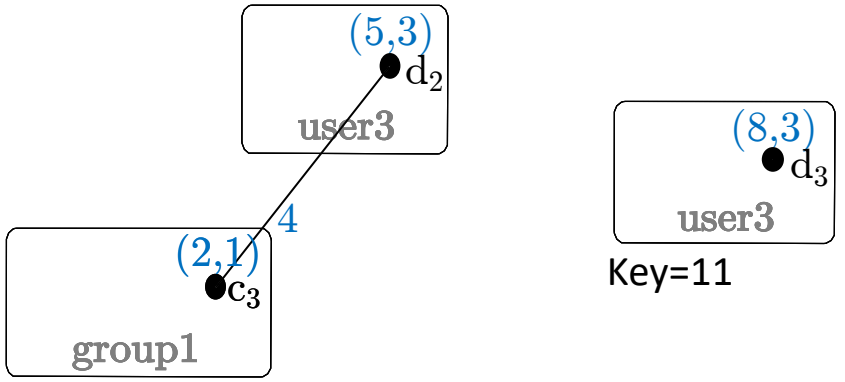
push	push	pop	push



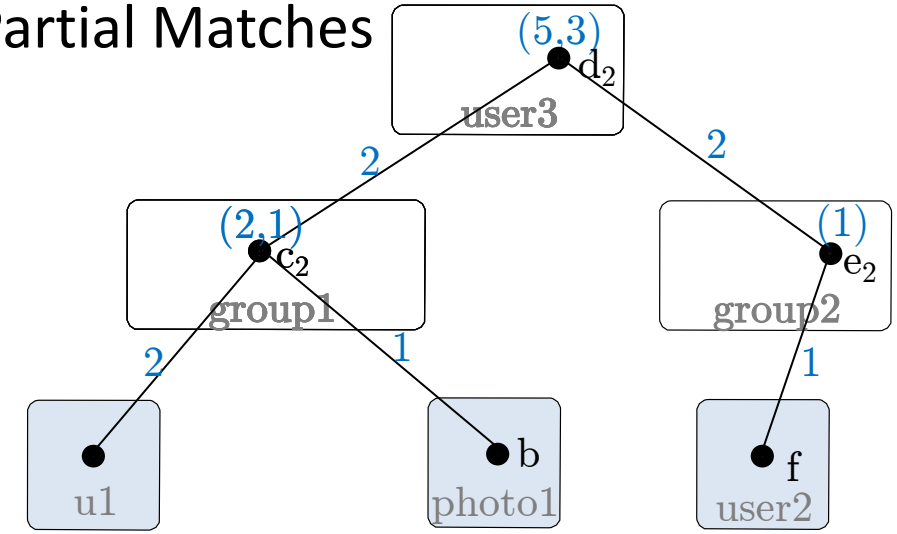
Candidate graph with subtree weights

Output

Priority queue

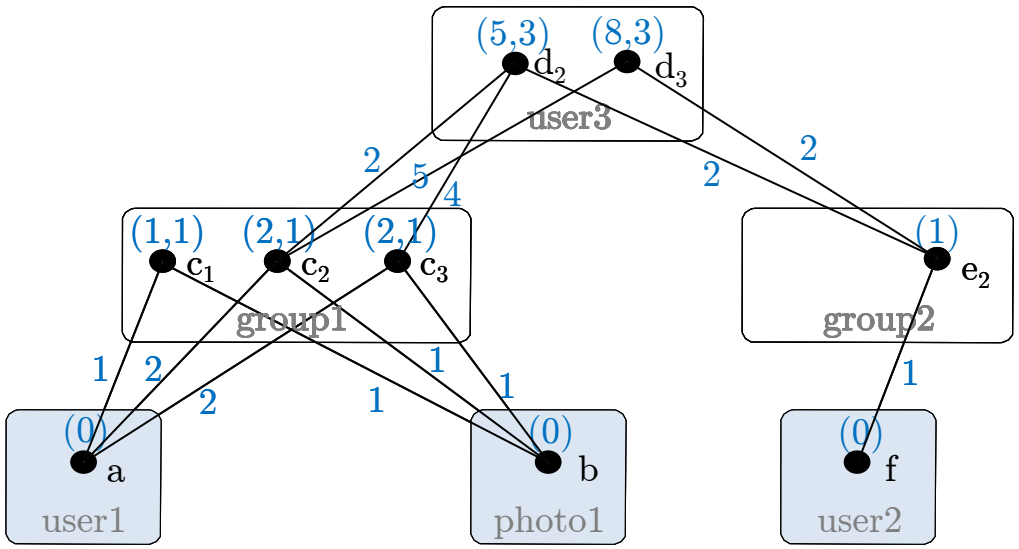


Partial Matches



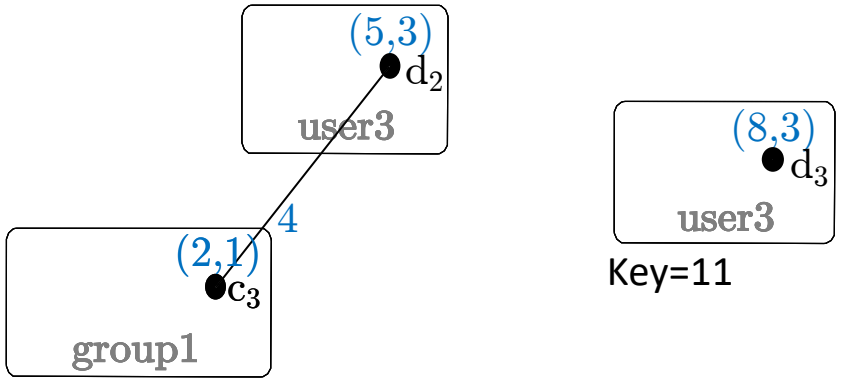
Key=8

push	push	pop	push



Candidate graph with subtree weights

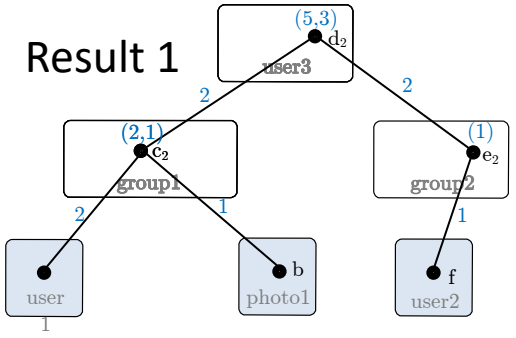
Priority queue



Partial Matches

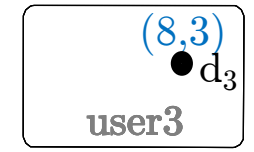
Output

Result 1

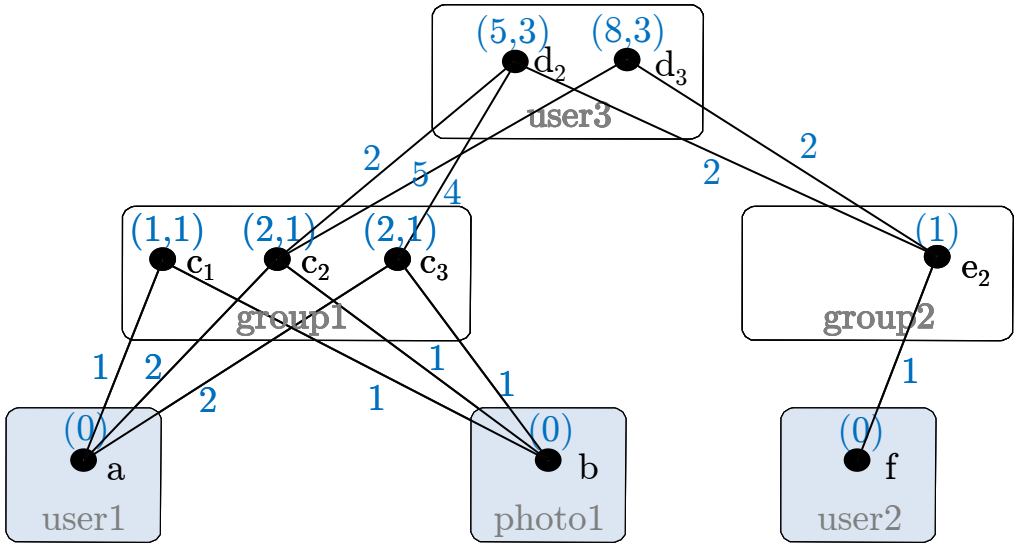


push	push	pop	push

Priority queue



Key=11

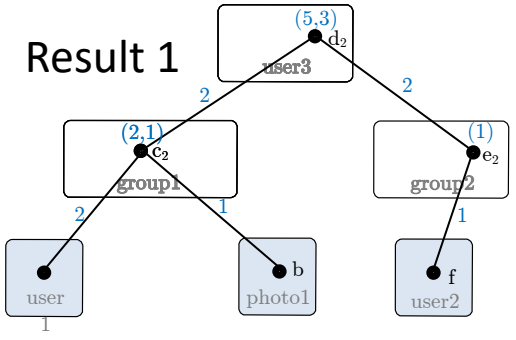


Candidate graph with subtree weights

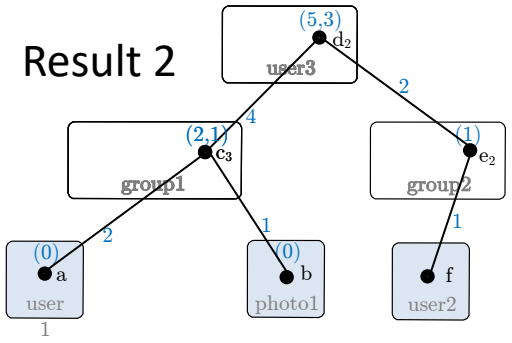
Partial Matches

Output

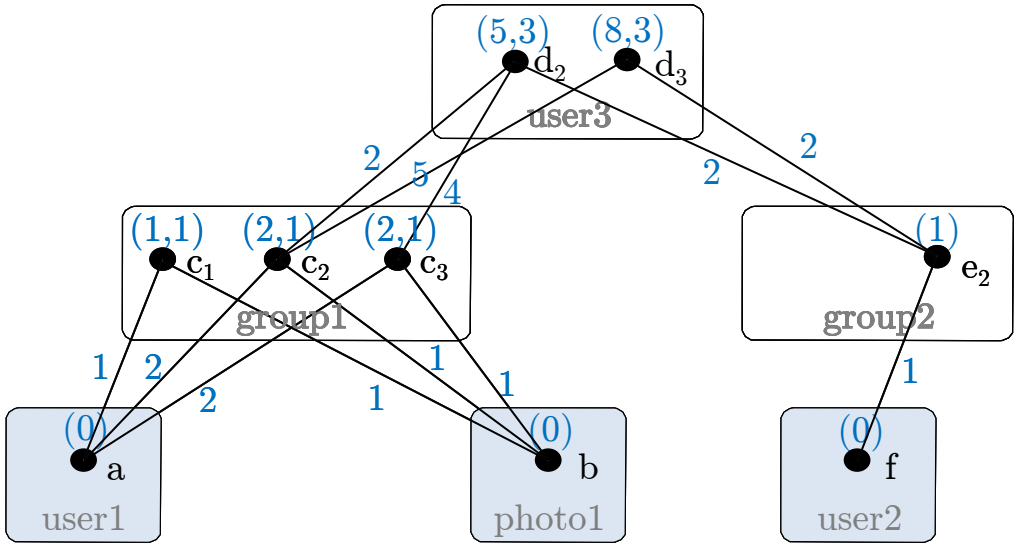
Result 1



Result 2



push	push	pop	push	pop	pop



Candidate graph with subtree weights

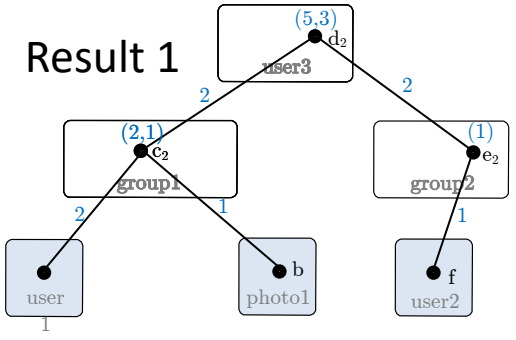
For all results, #push = #pop = #results (r_H)

r_H = #homomorphism matches.

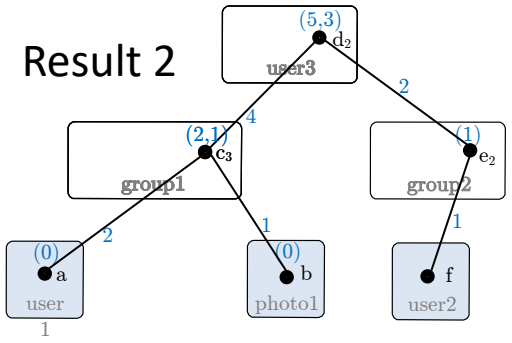
This is also the max space used in the priority queue.

Output

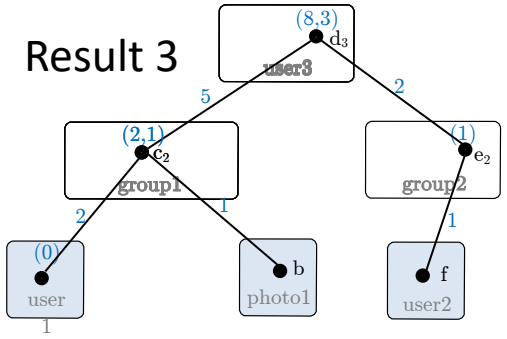
Result 1



Result 2



Result 3

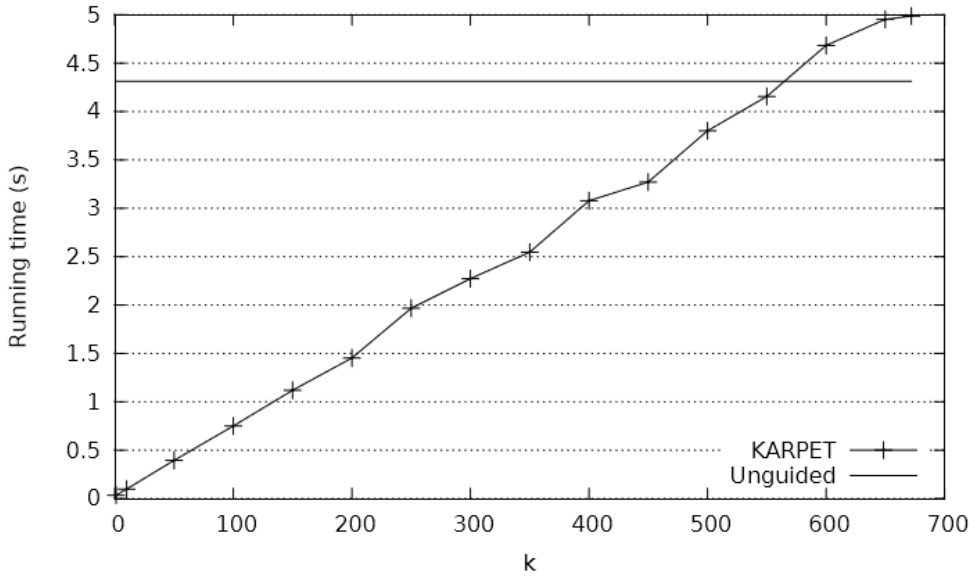


push	push	pop	push	pop	pop

Understanding the Any-k Property

- Theoretical guarantees:
  - Time to get a **homomorphism** next result
    - O(maxDegree + log(r_H))
  - Time to get all **homomorphism** results
    - O(r_H log(r_H))
    - Same as lower bound for bulk computing!

- In practice:
  - How does our **isomorphism** algorithm perform?

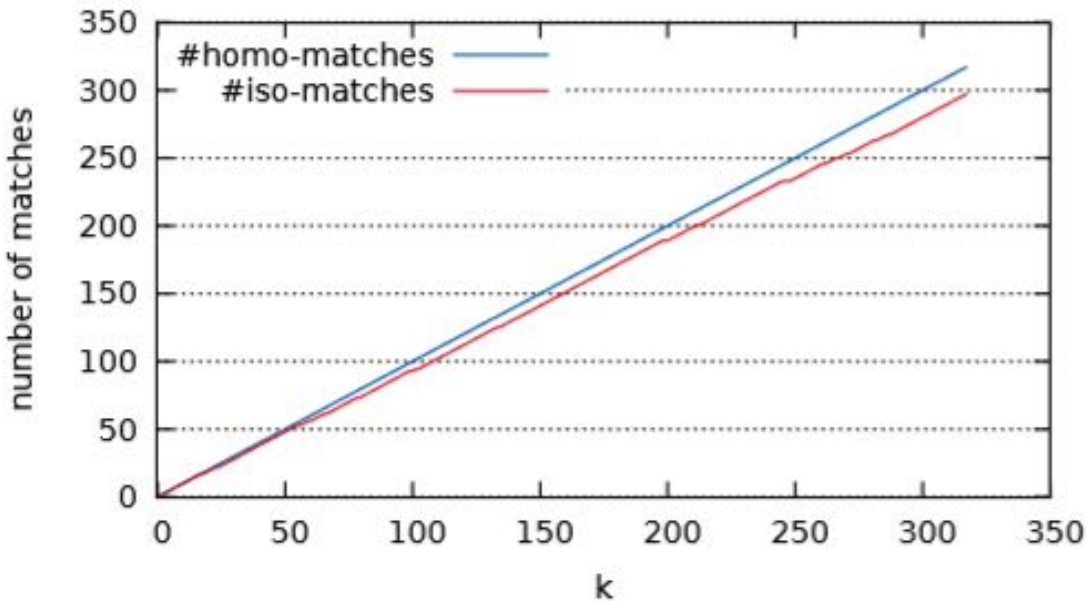


Running time of isomorphism queries

Homomorphism vs. Isomorphism

- Isomorphism: eliminate pattern when same node occurs more than once

- Small gap for heterogeneous graphs
  - Guarantee for homomorphism carries over to isomorphism



Gap between homomorphism and isomorphism on Enron

Summary

- Subgraph isomorphism problem for acyclic queries on labeled graphs—solved via subgraph homomorphism
- Strong worst case guarantees (homomorphism):
  - Time for bottom-up sweep to get candidate graph: **$O(|E| \cdot |Q|)$**
  - Time for top-down sweep to return the first/next result: **$O(\text{maxDegree} + \log(r_H))$**
  - Time for top-down sweep to return all results: **$O(r_H \log(r_H))$**
  - Space for top-down sweep: **$O(r_H)$**
- Speedup of one or more orders of magnitude on large real-world graphs

Current Work

- Subgraph homomorphism = conjunctive query over binary relations
- Extend to N-ary relations
  - Graph vs. hyper-graph
- Queries with cycles
  - Consider different tree decompositions
- Optimality results for general conjunctive queries

Thanks!

https://github.com/northeastern-datalab/Any-k-KARPET

This work was supported in part by the National Institutes of Health (NIH) under award number R01 NS091421 and by the National Science Foundation (NSF) under award number CAREER III-1762268. The content is solely the responsibility of the authors and does not necessarily represent the official views of NIH or NSF.