Optimizing Subgraph Queries With a Mix of Tradition & Modernity

Semih Salihoglu (joint w/ Amine Mhedhbi)

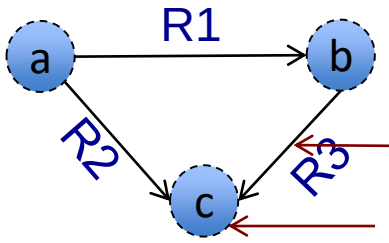
July 5th, 2019

UNIVERSITY OF WATERLOO | DSG Data Systems Group

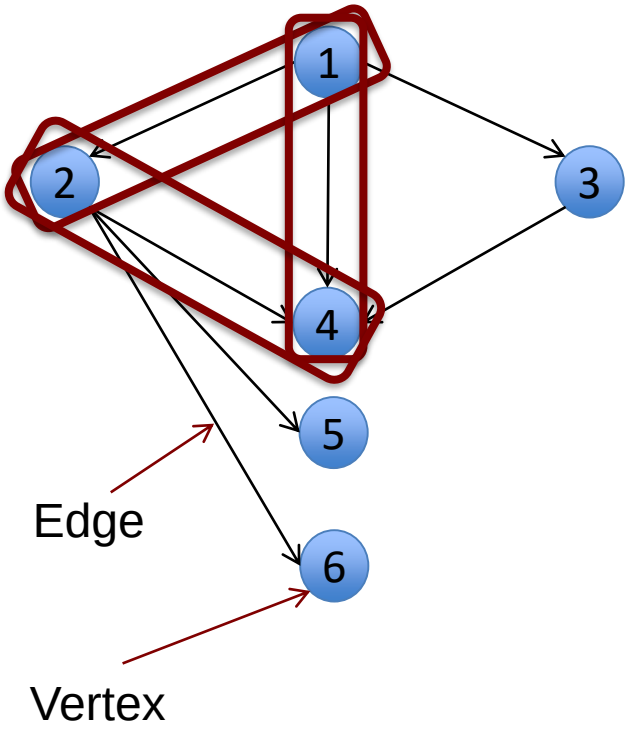
Subgraph Queries = Multiway (Self) Join Queries

```
MATCH (a)->(b)->(c), (a)->(c)
```

Query



Input Graph



R1	
a	b
1	2
1	3
1	4
2	4
2	5
2	6
3	4

⋈

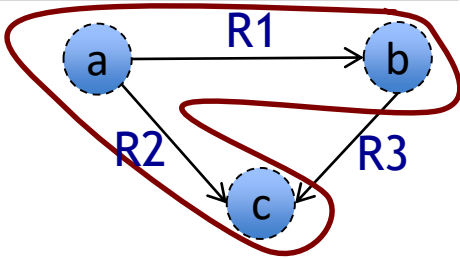
R2	
a	c
1	2
1	3
1	4
2	4
2	5
2	6
3	4

⋈

R3	
b	c
1	2
1	3
1	4
2	4
2	5
2	6
3	4

Traditionally: Binary Join Plans

Table(s)/Q-Edge(s)-at-a-time Joins



R1	
a	**b**
1	2
1	3
1	4
…	

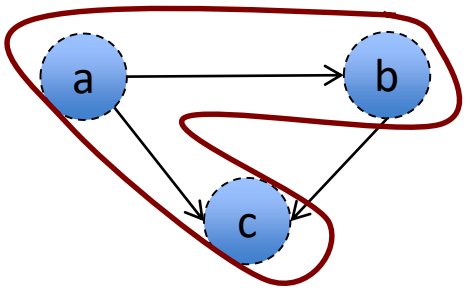
R2	
a	**c**
1	2
1	3
1	4
…	

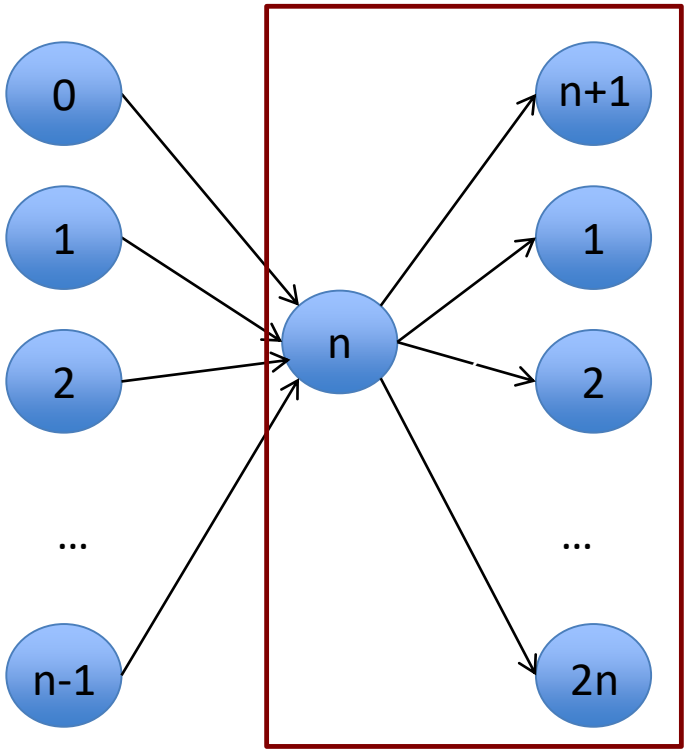
R3	
b	**c**
1	2
1	3
1	4
…	

INT_1		
a	**b**	**c**
1	2	2
1	2	3
1	2	4
1	3	2
1	3	4
…		

Output		
a	**b**	**c**
1	2	4
1	3	4

Recent Developments (AGM '08, NPRR '12): BJ Plans are Not (even) Worst-case Optimal





Any BJ Plan will take $O(m^2)$ time.

But max output is: $m^{3/2}$ (AGM Bound of Triangle Query)

Generic Join: A WCO Algorithm (NPRR, 2013)

a	b
1	2
1	3
1	4
2	4
2	5
2	6
3	4

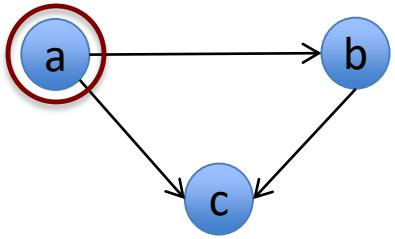
a	c
1	2
1	3
1	4
2	4
2	5
2	6
3	4

b	c
1	2
1	3
1	4
2	4
2	5
2	6
3	4

∩

Column/Q-Vertex-at-a-time

Order q-vertices: say: a,b,c



INT_1
a

Generic Join: A WCO Algorithm (NPRR, 2013)

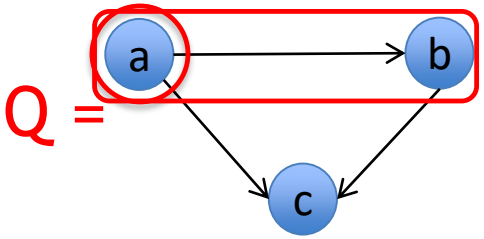
a	b
1	2
1	3
1	4
2	4
2	5
2	6
3	4

a	c
1	2
1	3
1	4
2	4
2	5
2	6
3	4

b	c
1	2
1	3
1	4
2	4
2	5
2	6
3	4

Column/Q-Vertex-at-a-time

Order q-vertices: say: a,b,c



INT_1
a
1
2
3

INT_2	
a	b

Generic Join: A WCO Algorithm (NPRR, 2013)

a	b
1	2
1	3
1	4
2	4
2	5
2	6
3	4

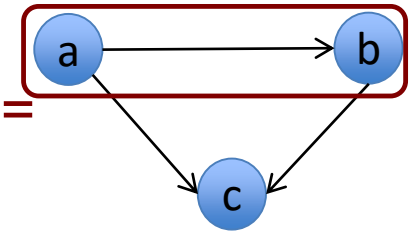
a	c
1	2
1	3
1	4
2	4
2	5
2	6
3	4

b	c
1	2
1	3
1	4
2	4
2	5
2	6
3	4

Column/Q-Vertex-at-a-time

Order q-vertices: say: a,b,c

Q =



INT_1
a
1
2
3

INT_2	
a	b
1	2
1	3

Generic Join: A WCO Algorithm (NPRR, 2013)

a	b
1	2
1	3
1	4
2	4
2	5
2	6
3	4

a	c
1	2
1	3
1	4
2	4
2	5
2	6
3	4

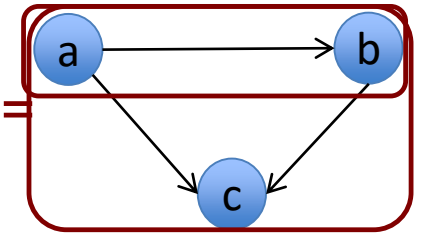
b	c
1	2
1	3
1	4
2	4
2	5
2	6
3	4

∩

Column/Q-Vertex-at-a-time

Order q-vertices: say: a,b,c

Q =



INT_1
a
1
2
3

INT_2	
a	b
1	2
1	3
2	4
2	5
2	6
3	4

Output		
a	b	c

Generic Join: A WCO Algorithm (NPRR, 2013)

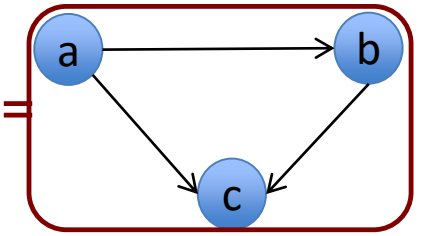
a	b
1	2
1	3
1	4
2	4
2	5
2	6
3	4

a	c
1	2
1	3
1	4
2	4
2	5
2	6
3	4

b	c
1	2
1	3
1	4
2	4
2	5
2	6
3	4

Column/Q-Vertex-at-a-time

Order q-vertices: say: a,b,c

Q = 

INT_1
a
1
2
3

INT_2	
a	b
1	2
1	3
2	4
2	5
2	6
3	4

Output		
a	b	c
1	2	4
1	3	4

Theorem (GJ is WCO): Runtime of GJ ≤ AGM

(for *any query* & *any q-vertex ordering*)

Research Overview on WCO Joins

Serial Setting

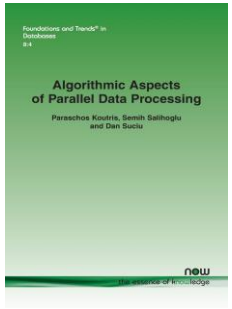
Graphflow

Prototype GDBMS at UWaterloo (SIGMOD ‘17 Demo, VLDB ‘19)

Distributed Setting

f^{∞}

Timely Dataflow (previously Naiad) (VLDB ‘18)

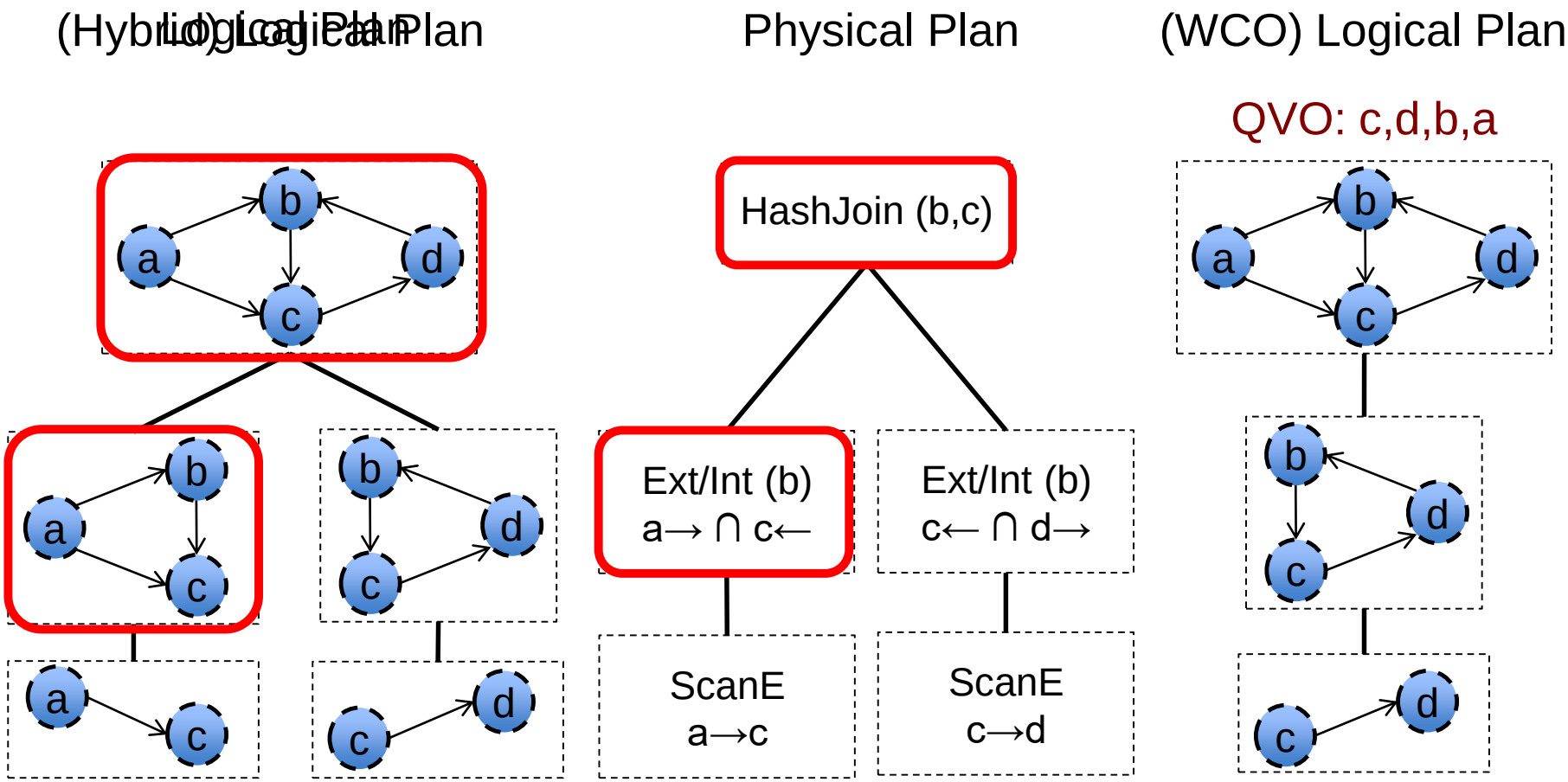


Survey on Load Complexity (F&T, ‘18)

What should the query vertex orderings (QVOs) be?
For which queries to use?
How to mix with traditional joins?
…

Graphflow Logical and Physical Plans for SQs

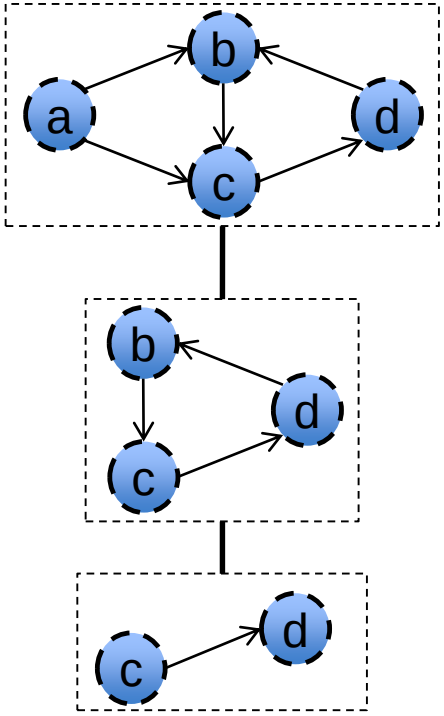
- Storage: 2 adjacency lists per vertex (fw & bw) stored in memory



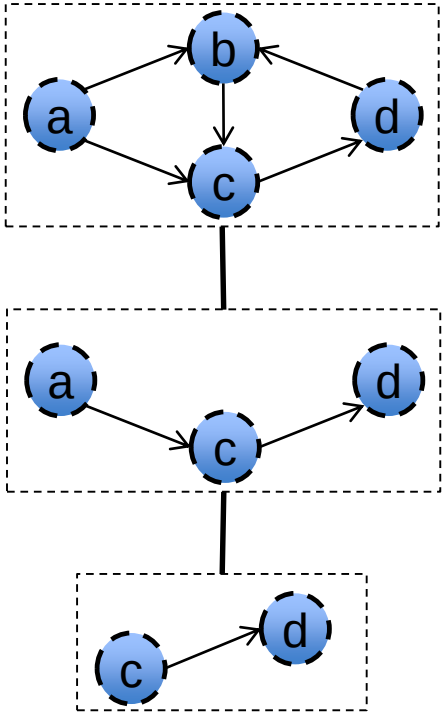
Q1: Ordering QVOs: Good Orders

- Two main effects of QVOs:
- First Effect: # intermediate partial matches.

P1: QVO: c,d,b,a



P2: QVO: c,d,a,b



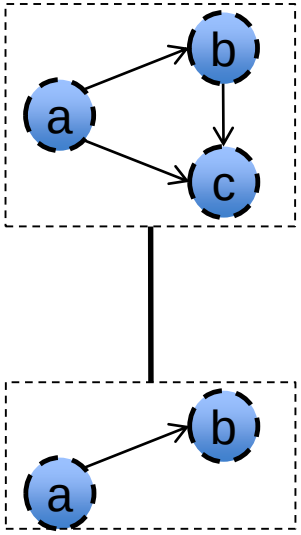
	P1	P2
Intermediate Matches	3.5M	40M
Run time	9s	16s

Soc-Epinions

Q1: Ordering QVOs: Good Orders

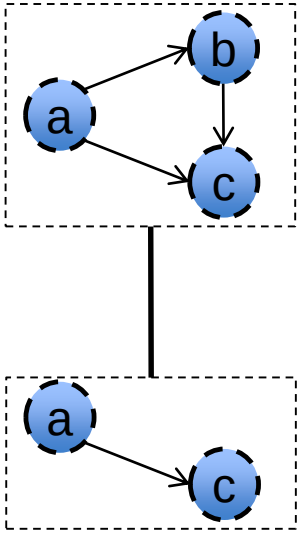
- 2nd Effect: Directions of adjacency lists intersected.

P1:QVO: a,b,c



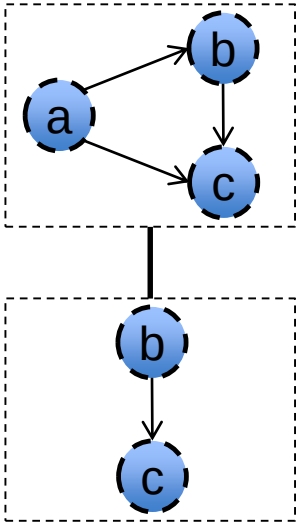
src→ ∩ dst→

P2: QVO: a,c,b



src→ ∩ dst←

P3: QVO: b,c,a



src← ∩ dst←

	P1	P2	P3
Run time	3s	15s	31s

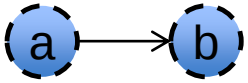
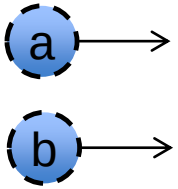
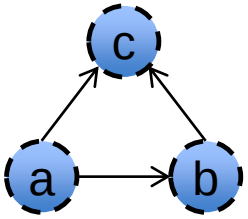
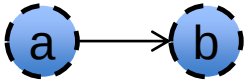
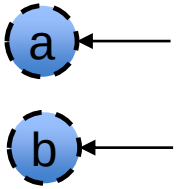
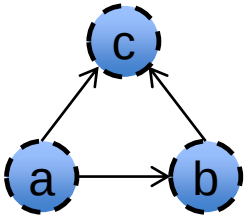
Web-BerkStan

Graphflow Optimizer: Minimize estimated *intersection cost (i-cost)*

I-cost: ∑ (size-of-adj-lists-intersected-throughout-execution)

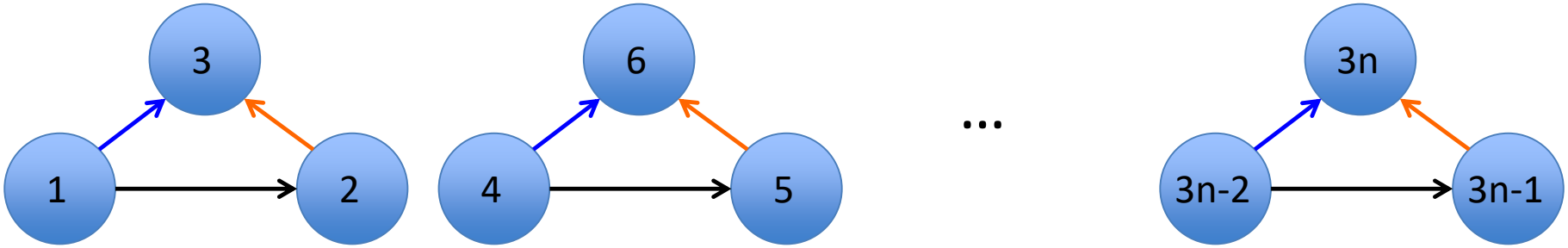
(estimate # partial matches & avg adj-list sizes w/ subgraph catalogue)

I-cost Estimation Technique: Subgraph Catalogue

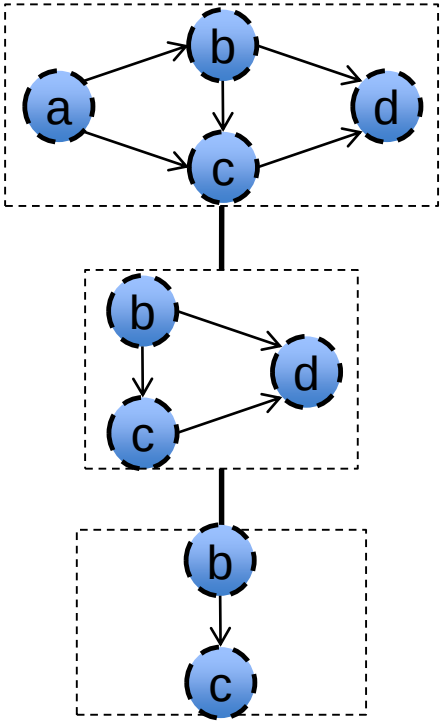
S	Adj. Lists	S'	i-cost	selectivity
...	...	...	...	...
			57	4.6
			158	4.6
...	...	...	...	...

Q1: Ordering QVOs: Adaptive Orders

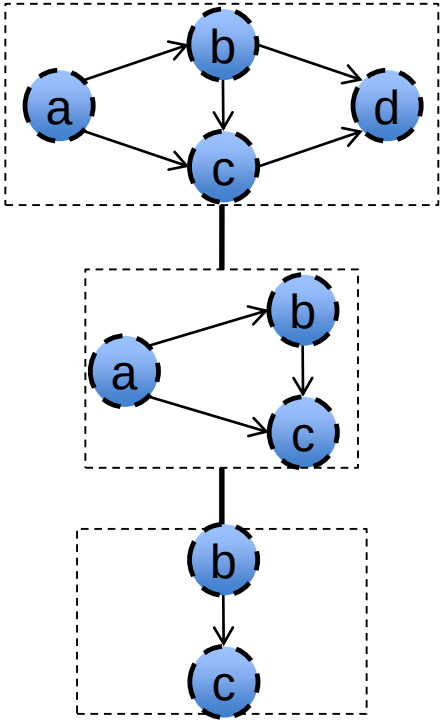
- Picking a fixed QVO may be suboptimal.



P1: QVO: b,c,d,a

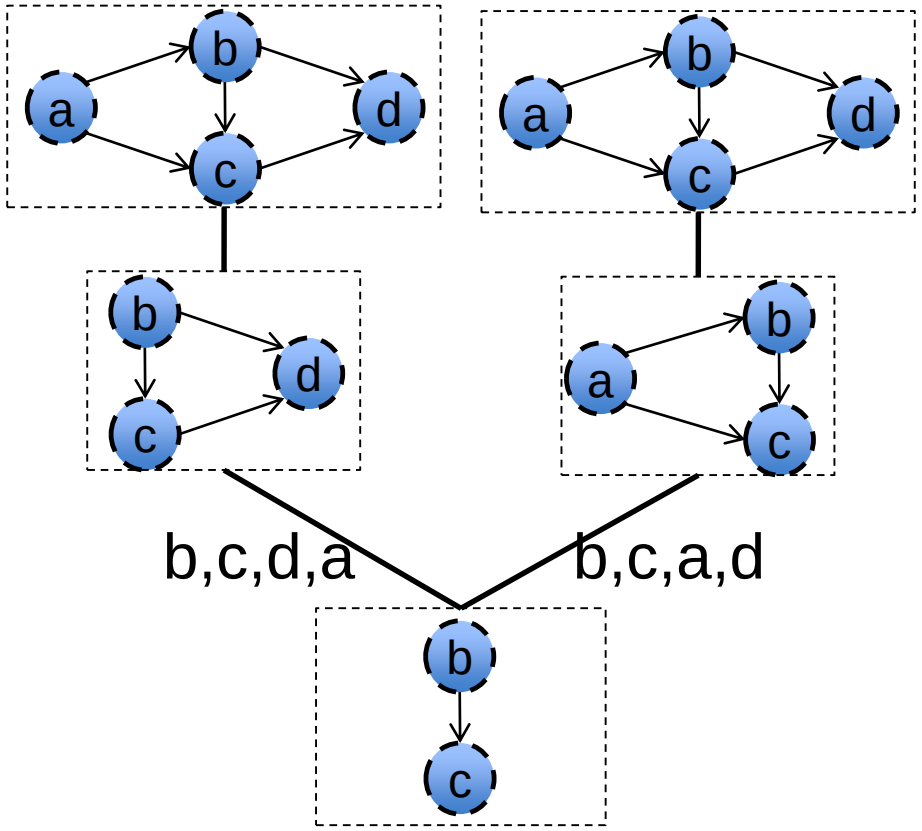


P2: QVO: b,c,a,d

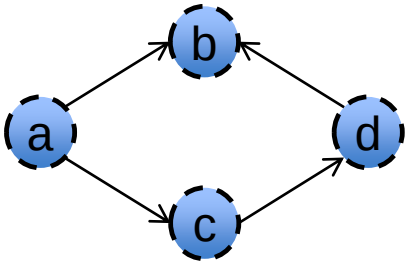


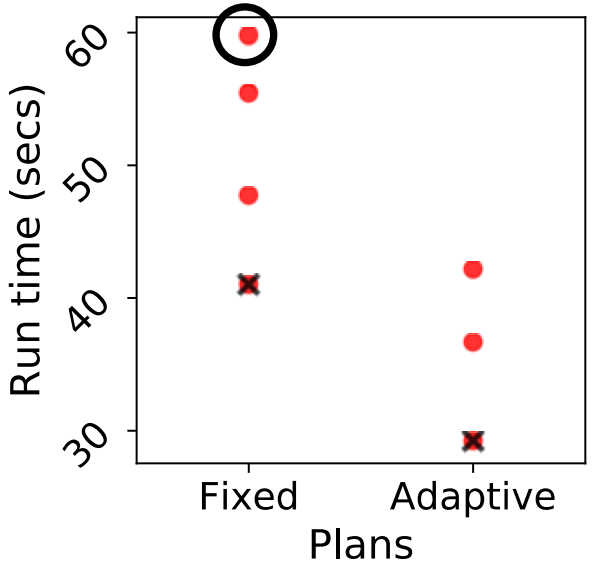
	P1	P2
→	2n	0
→	n	n
→	0	2n
Total	3n	3n

Q1: Ordering QVOs: Example Adaptive WCO Plan

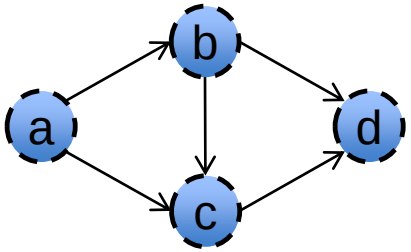


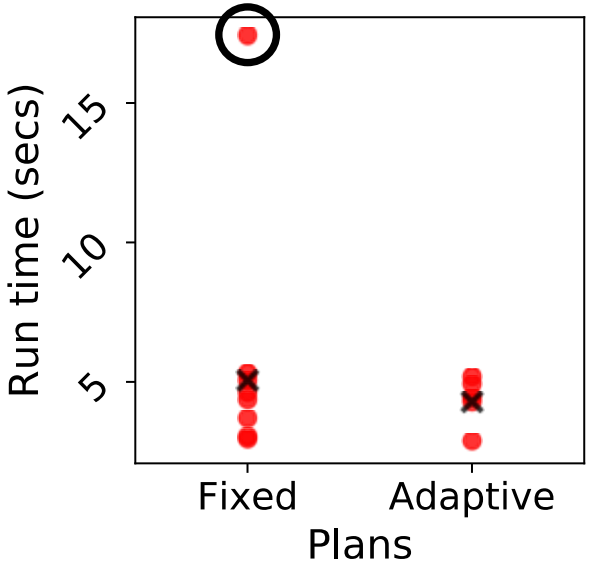
Q1: Ordering QVOs: Effects of Adaptive WCO Plans





Soc-Epinions

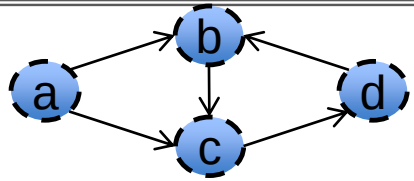




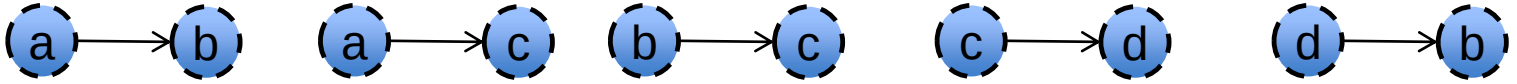
Web-Google

Q2: How to mix BJs with Intersections?

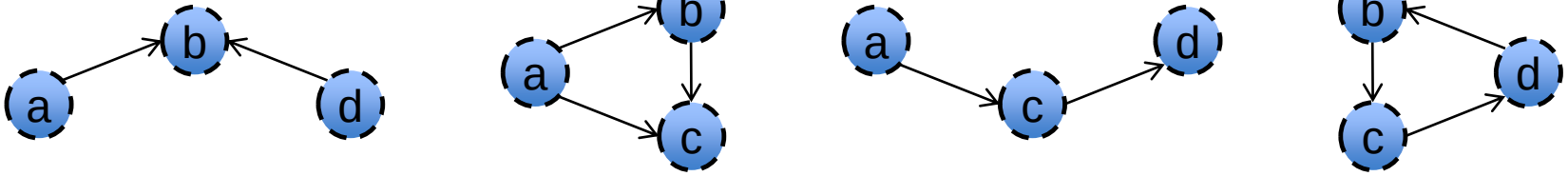
- Cost-based DP optimization algorithm.



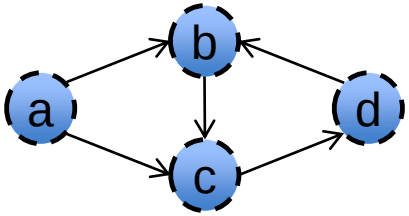
Optimum Plans for all connected sub-queries with 2 q-vertices:



Optimum Plans for all connected sub-queries with 3 q-vertices:

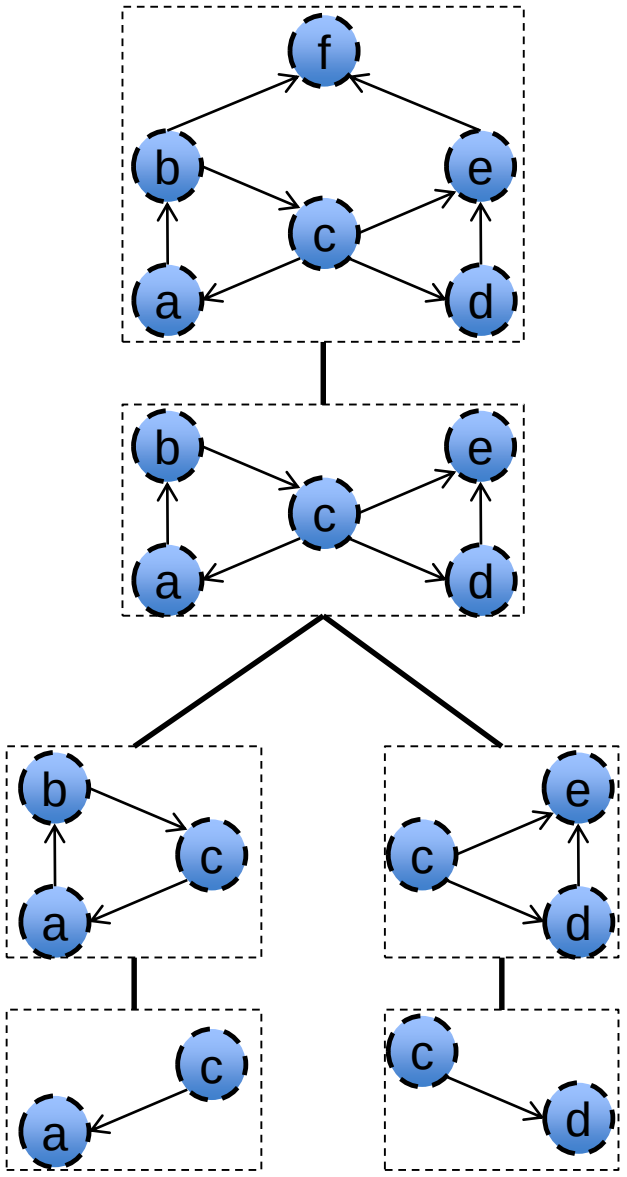


Optimum Plans for all connected sub-queries with 4 q-vertices:

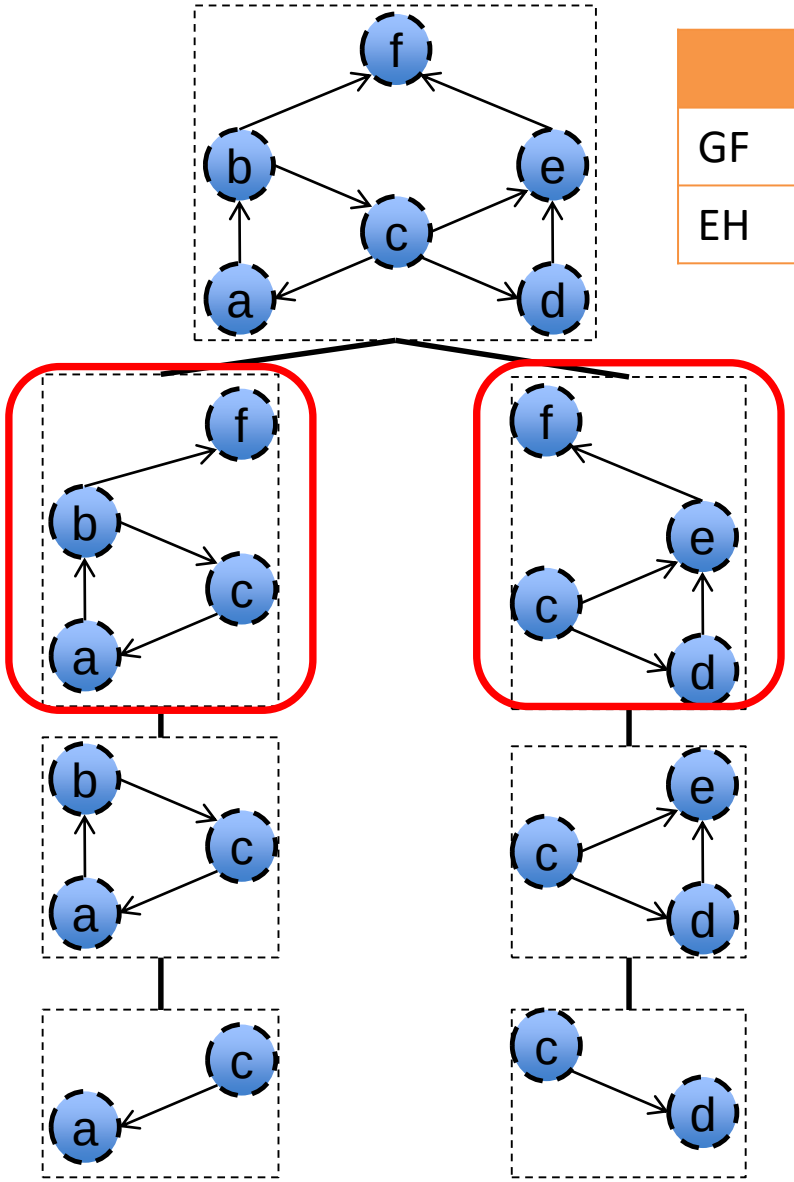


Each step: consider plans with both hash joins & intersections.

Graphflow vs Empty-Headed Hybrid Plans



Graphflow

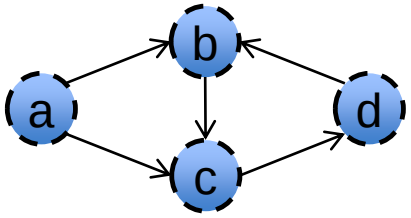


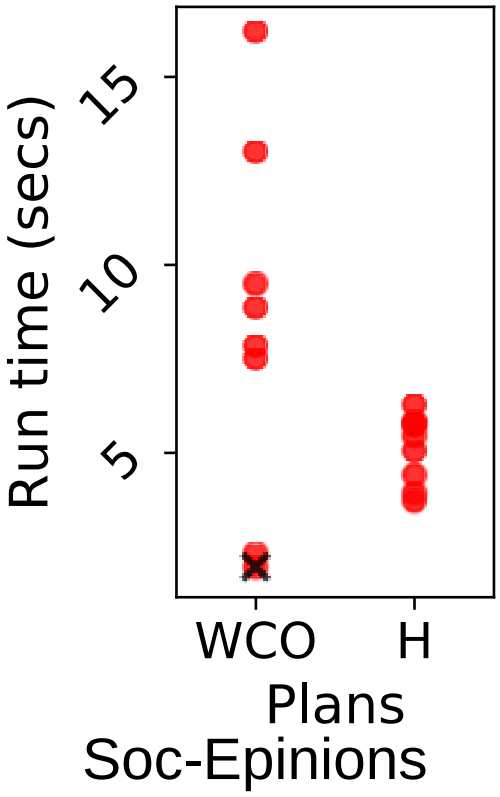
EmptyHeaded

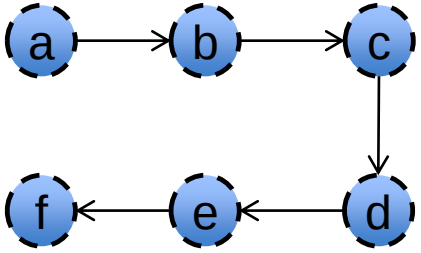
	Amzn	Web-G
GF	6.7s	328s
EH	25.2s	>5 hours

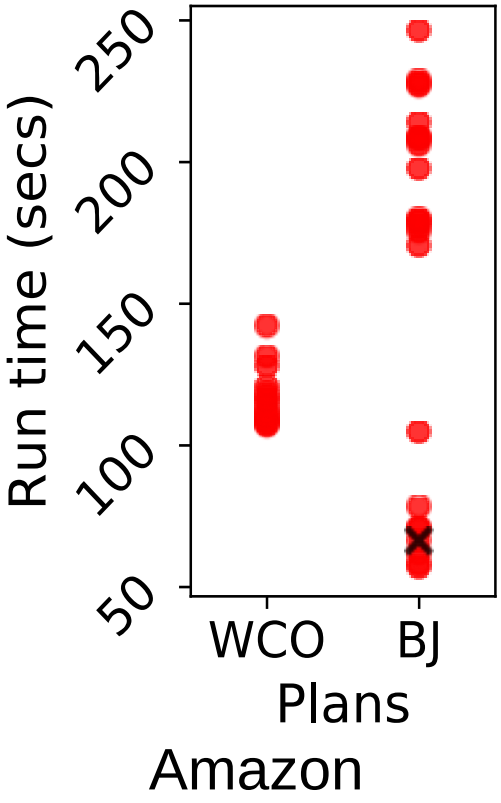
Q3: For Which Queries are WCO, BJ, Hybrid Plans Good?

- Size and cyclicity

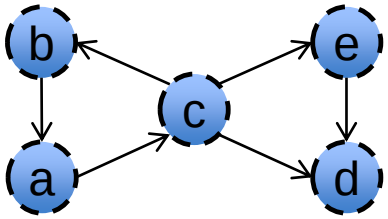


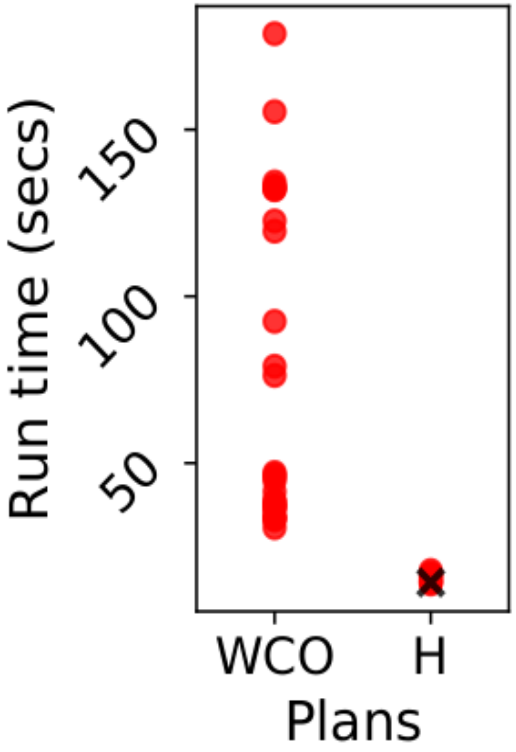




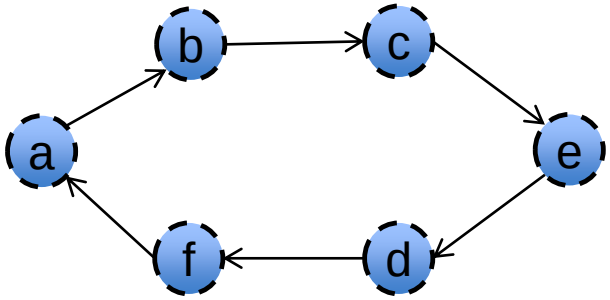


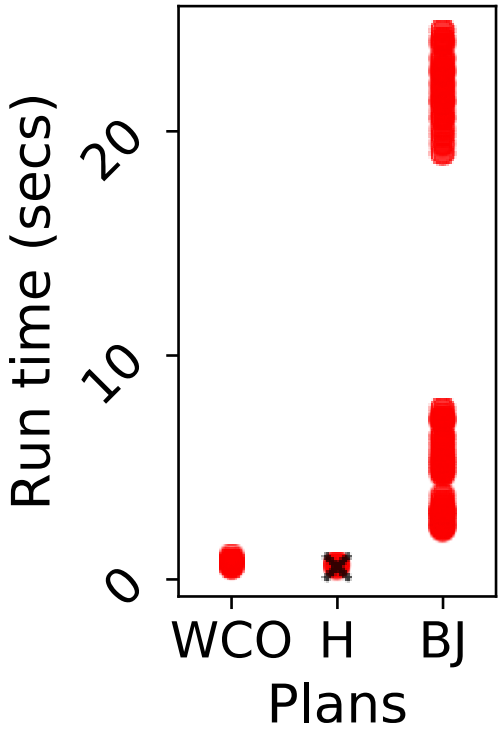
Q3: For Which Queries are WCO, BJ, Hybrid Plans Good?





Web-Google





Gnutella

Summary

- For subgraph queries, Graphflow optimizer:
  - Adopts the *i-cost cost metric*
  - Uses a *subgraph catalogue* for i-cost estimation
  - Seamlessly mixes binary joins and WCO-style intersections with a dynamic programming optimizer
  - Adaptively changes the query vertex orderings during runtime
- Similar techniques for cont. subgraph matching (upcoming work)

Students

Amine Mhedhbi

Siddhartha Sahu

Chathura Kankanamge

References

- Efficiently Updating Materialized Views, BLT, SIGMOD Record, 1986
- An Efficient Distributed Subgraph Mining Algorithm in Extreme Large Graphs, WB, AICI, 2010
- Optimizing Multiway Joins in a Map-Reduce Environment, AU, TKDE, 2011
- Leapfrog Triejoin: a worst-case optimal join algorithm, Veldheuizen, arxiv 2012
- Tri, Tri Again: Finding Triangles and Small Subgraphs in a Distributed Setting, DLP, DISC, 2012
- Worst-case Optimal Join Algorithms, NPRR, PODS 2012
- A Distributed Graph Engine for Web Scale RDF Data, ZYWKW, VLDB, 2013
- Communication Steps for Parallel Query Processing, BKS, PODS, 2013
- Incremental Maintenance for LeapfrogTriejoin, Veldheuizen, arxiv 2013
- Naiad: A Timely Dataflow System, MMIIA, SOSP, 2013
- PATRIC: A Parallel Algorithm for Counting Triangles in Massive Network, CIKM, 2013
- Size Bounds and Query Plans for Relational Joins, AGM, SIAM Journal of Computing, 2013
- Upper and Lower Bounds in the Cost of a Map-Reduce Computations, ASSU, VLDB, 2013
- Skew Strikes Back: New Developments in the Theory of Join Algorithms, NPRR, SIGMOD Record, 2014
- Anchor Points Algorithms for Hamming and Edit Distance, ADRRSU, ICDT 2014
- Arabesque: A System for Distributed Graph Mining, SOSP, 2015
- Continuous Pattern Detection Over Billion-edge Graph Using Distributed Framework, CHCAF, EDBT, 2015
- EmptyHeaded: A Relational Engine for Graph Processing, ATOR, SIGMOD, 2016
- PTE: enumerating trillion triangles on distributed systems, PMK, KDD, 2016
- Scalable Distributed Subgraph Enumeration, LQLZC, VLDB, 2016
- Graphflow: An Active Graph Database, KSMCS, SIGMOD, 2017
- GYM: A Multiround Distributed Join Algorithm, AJRSU, 2017
- Distributed Evaluation of Subgraph Queries Using Worst-case Optimal Low-Memory Dataflows, AMS, VLDB, 2018
- Ubiquity of Large Graphs and Surprising Challenges of Graph Processing, SMSLO, VLDB, 2018
- Algorithmic Aspects of Paralllel Data Processing, KSS, F&T, 2018

Thank you

WATERLOO | DATA SYSTEMS GROUP