http://www.cs.umd.edu/~kostasx

SIGMOD 2017

Extracting and Analyzing Hidden Graphs from Relational Databases

Konstantinos Xirogiannopoulos, Amol Deshpande

University of Maryland, College Park

1. Graph Data Management

Graph Analysis Tasks *Vary Widely*

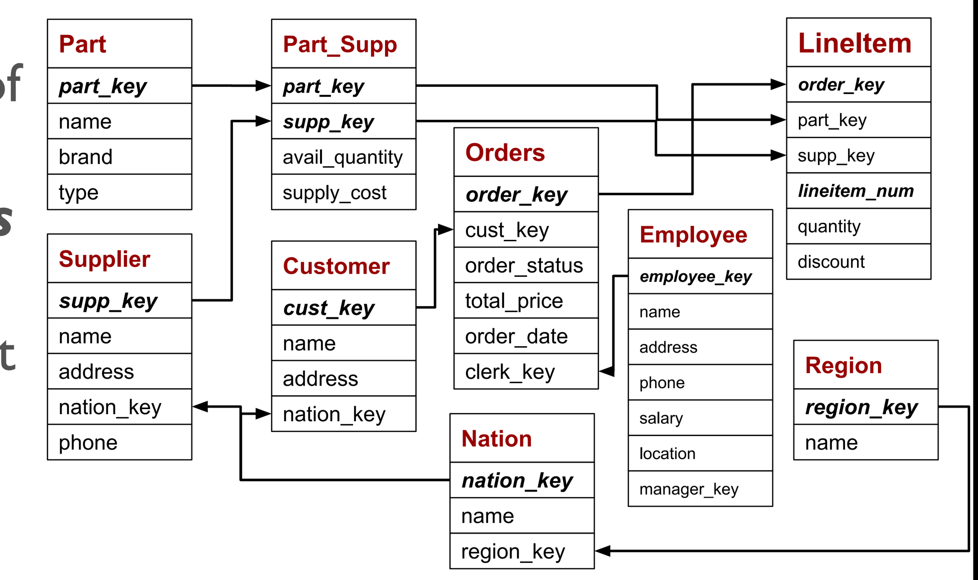
- Different types of Graph **Queries**
- *Continuous* Queries / ***Real-Time*** Analysis
- **Batch** Graph Analytics
- Machine Learning

Many *different* ways to deal with graph data

- Graph Databases (neo4j, orientDB, RDF stores)
- Distributed Batch Analysis Frameworks (Giraph, GraphX, GraphLab)
- In-Memory Systems(Ligra, Green-Marl, X-Stream)
- Many research prototypes / custom indexes

2. But first…Where is your data?

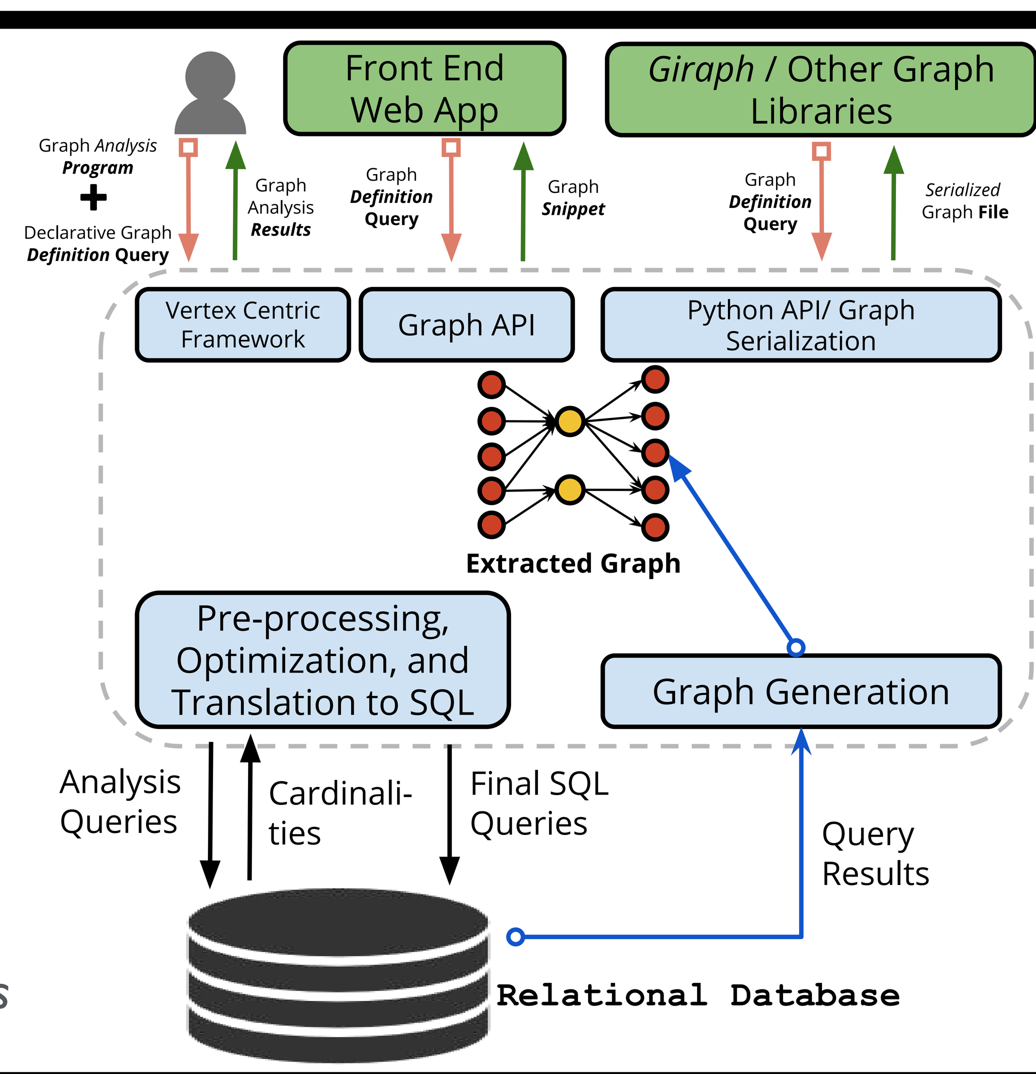
- Users' data typically in ***RDBMSs*** or ***Key-Value*** Stores with some sort of ***schema***
- Graph systems require lists of ***nodes & edges***
- ***Extraction*** step often *overlooked* but can be quite ***involved***
  - »User needs to *write* custom ***SQL*** queries for ***ETL***
  - »Can be ***unintuitive*** & time consuming
  - »Large ***selectivity estimation errors*** due to complex joins
  - »Need to ***repeat*** every time database is updated



Graph	Representation	Edges	Extraction Time (s)
DBLP $10M$ rows	Condensed	17,147,302	105.552
	Full Graph	86,190,578	> 1200.000
IMDB $4.7M$ rows	Condensed	8,437,792	108.647
	Full Graph	33,066,098	687.223
TPCH $765K$ rows	Condensed	52,850	15.52
	Full Graph	99,990,000	> 1200.000
UNIV $32K$ rows	Condensed	60,000	0.033
	Full Graph	3,592,176	82.042

3. GraphGen

- A software ***layer*** over relational/*structured* databases (implemented as a library)
- User specifies graph extraction queries in a ***Datalog-based DSL***
- Can ***serialize*** the graph and load it into other *frameworks/ libraries*
- Exposes ***vertex-centric API*** *or* ***direct*** *graph access through Java API*
  - ***WIP***: *Supporting a Datalog Based DSL for Querying/Analytics*



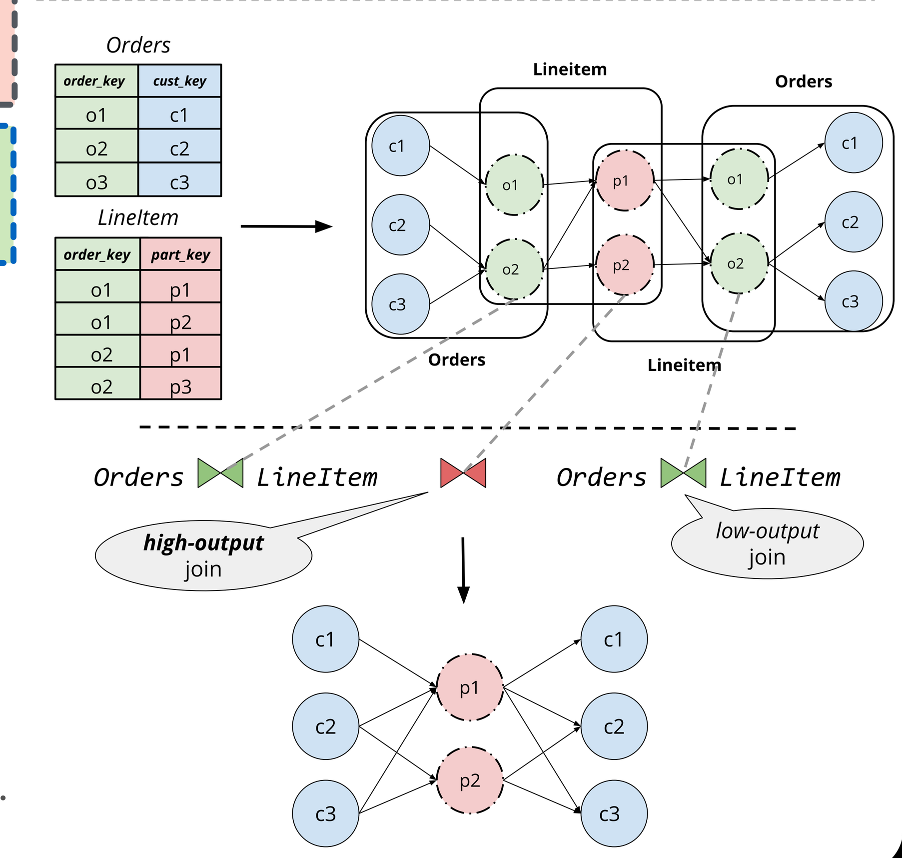
4. Condensed Representation

Key Challenge #1: Graphs often *orders-of-magnitude* ***larger*** than input. May ***not fit*** in-memory!

Solution: Instead extract a ***Condensed Representation***

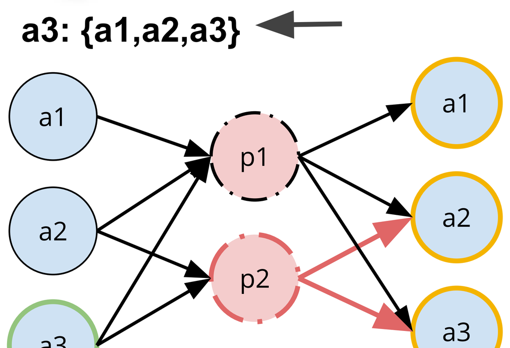
1. Translate ***Nodes*** statements to SQL and ***execute*** them.
2. ***Edges*** statements (acyclic, aggregation-free) are ***split*** by join.
3. For each join between R_i, R_{i+1} retrieve number of ***distinct values d*** for the join condition attribute(s).
4. Every join where $|R_i||R_{i+1}|/d > 2\ (|R_i|+|R_{i+1}|)$ marked ***large-output***
5. Create ***virtual nodes*** for every large-output join. Execute rest of joins ***in-database***

```
Nodes(ID, Name) :- Customer(ID, Name).
Edges(ID1, ID2) :- Orders(o_key1, ID1), LineItem(o_key1, part_key),
                   Orders(o_key2, ID2), LineItem(o_key2, part_key).
```



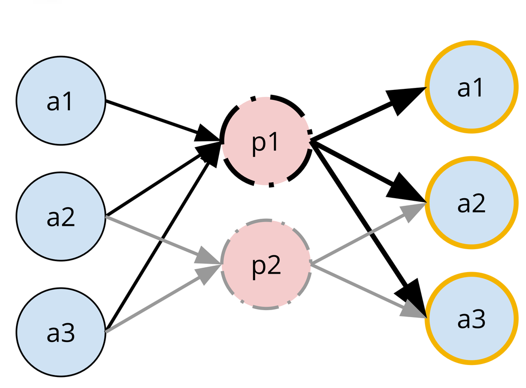
5. Duplicate Elimination

C-DUP



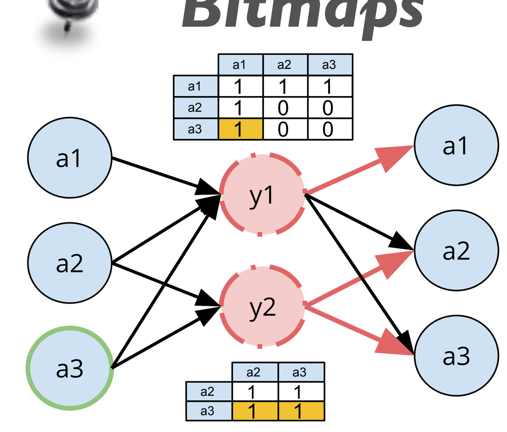
- *On-the fly* de-duplication ***caching*** every `getNeighbors()` call
- Great for graph ***queries*** that touch ***small portions*** of the graph
- Most ***storage-efficient*** solution

DEDUP-1



- ***Structural*** de-duplication of ***C-DUP***.
- *Single-path* per pair of neighbors
- Most ***portable*** solution

Bitmaps



- Add a ***bitmap*** at every *virtual node*
- ***Guides*** iteration for every `getNeighbors()` call to *avoid duplicates*

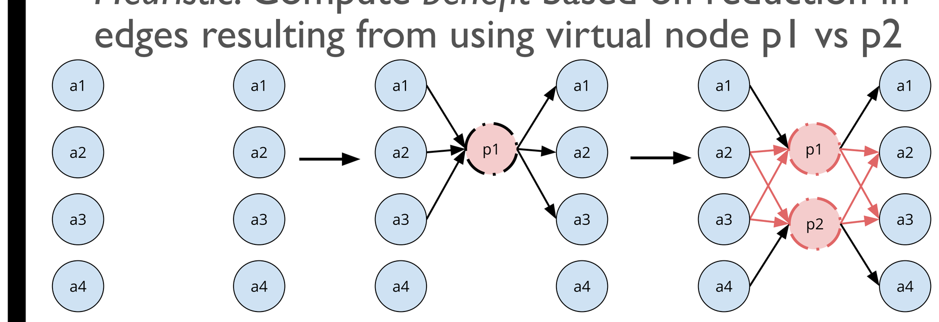
Key Challenge #2: There may be ***multiple paths*** between pairs of nodes in the Condensed Representation

Solution: *Override the* `getNeighbors()` *iterator to enable* ***any algorithm*** *over the Condensed Representation*

6. Structural *De-duplication*

DEDUP-1: Algorithms

- *Naive* Virtual-Nodes-First: ***Choose*** which real node to *remove* ***randomly***
- *Naive* Real-Nodes-First: *Same*, remove ***all duplication*** for each real node ***u*** before moving on the next one
- ***Greedy*** Virtual-Nodes-First: *Heuristic:* Compute "global" *benefit/cost* ratio of disconnecting real node ***u*** from virtual node *p1 vs p2*
- ***Greedy*** Real-Nodes-First: *Heuristic:* Compute *benefit* based on reduction in edges resulting from using virtual node p1 vs p2



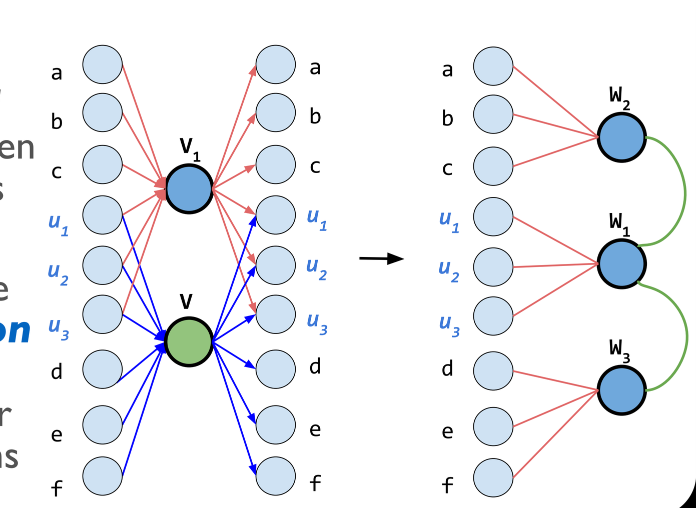
De-duplication: Given a *condensed* graph remove edges until there is ***one path*** between each pair of neighbors

Bi-clique Compression: Partition edges into *minimum* set of ***bipartite cliques*** (NP-Complete) *[Feder, Motwani '94]*

Same ***complexity***, *same* ***output***, *different* ***input***

DEDUP-2: Optimization for Symmetric Graphs

- Uses ***undirected*** edges between virtual nodes
- Can lead to ***10x*** or more ***compression*** *(comp. to DEDUP-1)* for ***dense*** graphs

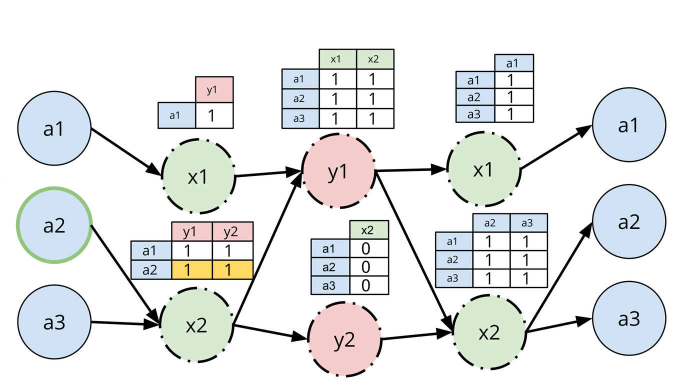


7. De-duplication using *Bitmaps*

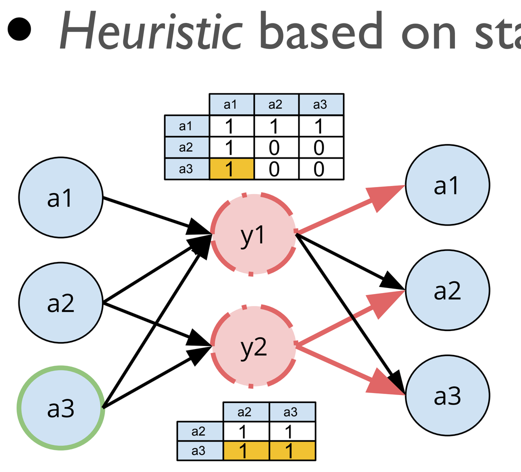
Optimization Problem

- Let O(V_n) the set of *real nodes* connected to virtual node *Vn.*
- Given a real node ***u***, and its virtual nodes {V_1,V_2,…,V_n}, find the ***smallest subset*** of {O(V_1), O(V_2),…,O(V_n)} that *covers* their ***union***
- *Heuristic* based on standard ***greedy set cover***

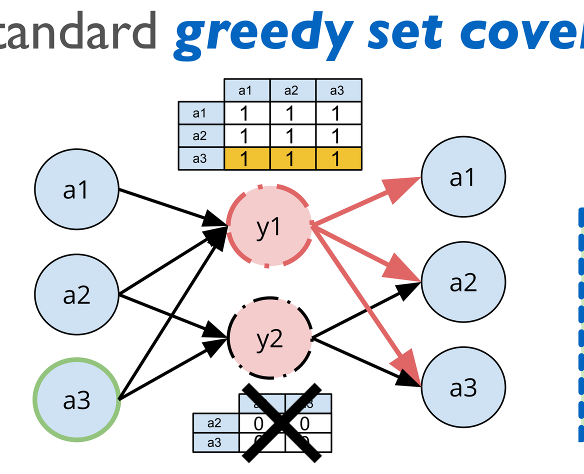
- Works on ***Multi-layered Condensed*** graphs
- *Apply algorithm* ***at every layer***



Main idea: Use ***bitmaps*** at every virtual node to avoid duplicate paths

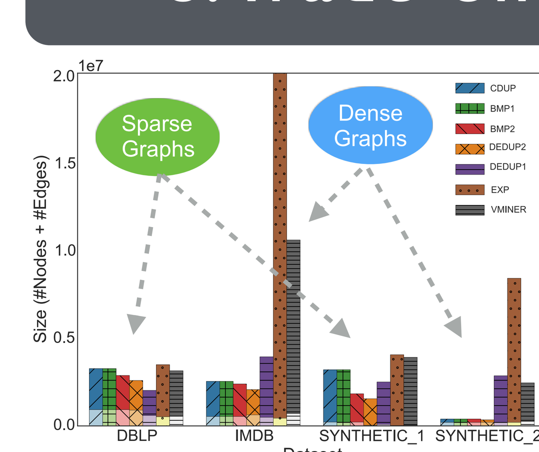


Bad **Bitmap placement**



Good **Bitmap placement**

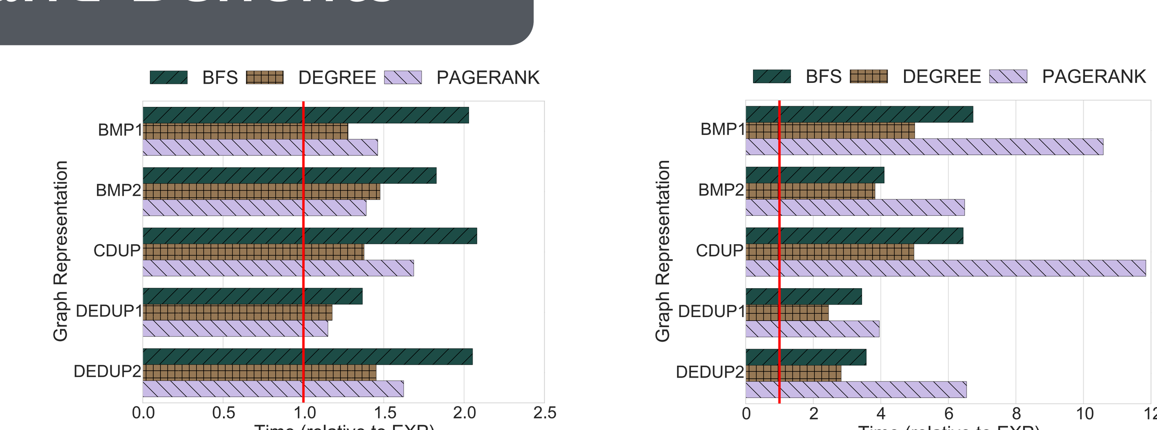
8. Trade-offs and Benefits



Small Datasets

Data Set	Repr	Degree time	Degree mem	ConComp time	ConComp mem	PageRank time	PageRank mem
S1	EXP	61	237	90	202	**245**	526
	DEDUP1	54	227	**81**	171	311	484
	BMP	**50**	**134**	82	**72**	256	**156**
S2	EXP	**294**	2,879	498	2,869	3,287	9,164
	DEDUP1	335	2,582	460	2,573	3,049	8,126
	BMP	311	**186**	**335**	**163**	**812**	**293**
N1	EXP	142	1,109	241	1,088	1,456	3,389
	DEDUP1	141	926	483	901	1,317	2,874
	BMP	**131**	**219**	**149**	**150**	**469**	**377**
N2	EXP	268	2,710	593	2,690	4,493	8,432
	DEDUP1	312	2,216	495	2,194	3,726	6,892
	BMP	**257**	**479**	**280**	**347**	**824**	**691**

Integration with *Apache Graph*



Iteration Performance on Condensed Graphs

	CDUP	BMP-DEDUP	FULL GRAPH
Layered-1	1.421 GB	2.737 GB	>64 GB
Layered-2	1.613 GB	2.258 GB	19.798 GB
Single-1	1.276 GB	1.493 GB	1.2 GB
Single-2	9.9 GB	13.042 GB	>64 GB
TPCH	.023 GB	.049 GB	7.398 GB

	CDUP	BMP-DEDUP	FULL GRAPH
Layered-1	382 s	284 s	DNF
Layered-2	129 s	111 s	85 s
Single-1	0.01 s	0.02 s	0.01 s
Syn-4	1.3 s	0.12 s	DNF
TPCH	86 s	8.5 s	16 s

Large Datasets

Dataset	Nodes	Edges	Join Selectivities
Layered_1	1,299,990	3,999,884	$0.05 \to 0.1 \to 0.05$
Layered_2	1,498,692	3,999,908	$0.2 \to 0.1 \to 0.2$
Single_1	1,245,532	2,000,000	0.25
Single_2	10,010,000	20,000,000	0.01
S1	50,100	96,066	0.002
S2	50,100	463,692	0.0004
N1	84,000	1,585,536	0.00025
N2	150,000	3,972,972	0.0001

GraphGen: Efficient *in-memory* **extraction** and ***analysis*** of ***larger-than-memory*** graphs ***hidden*** within *relational datasets*