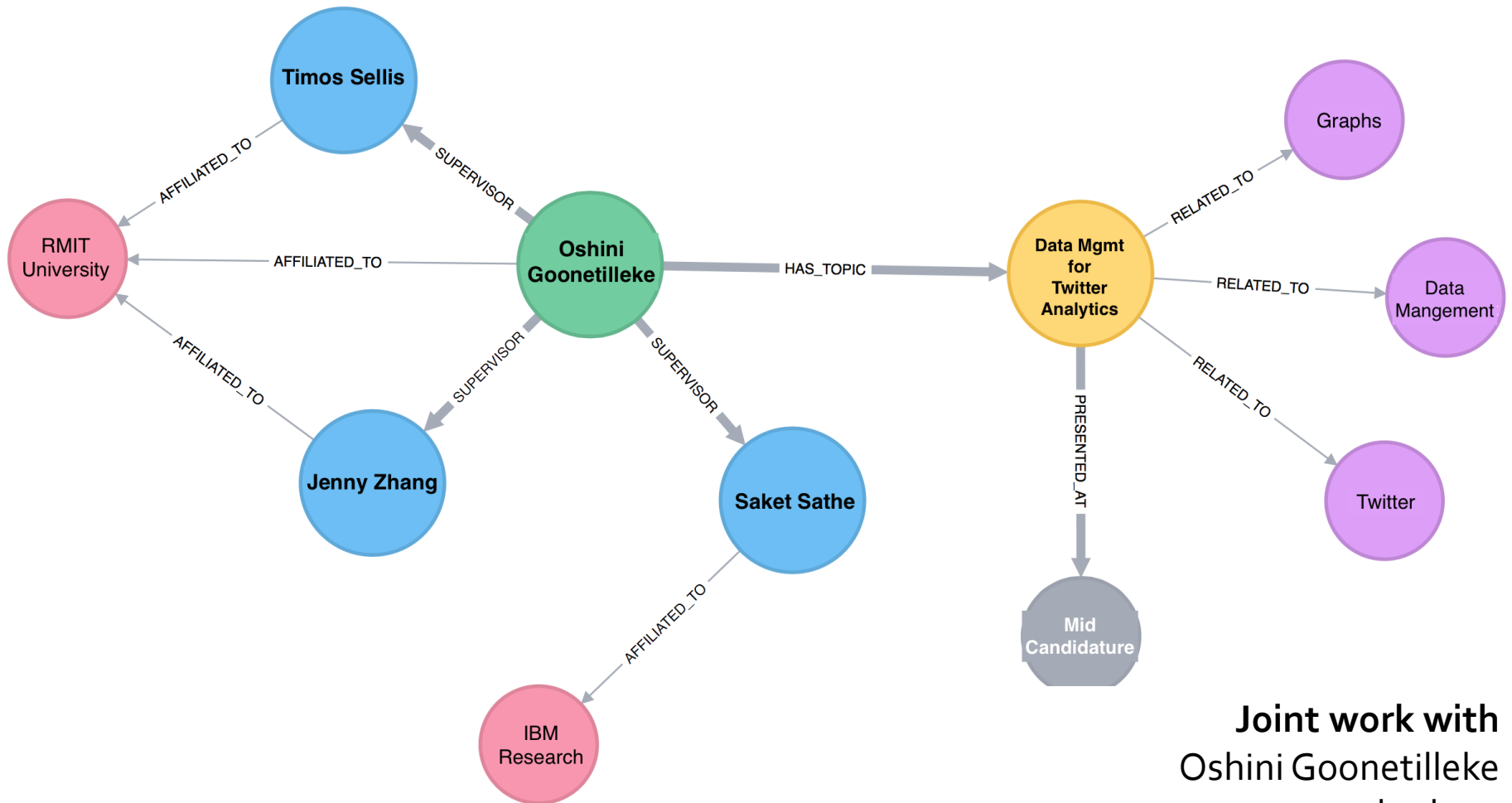


Big Graph Data problems in Social Network Platforms



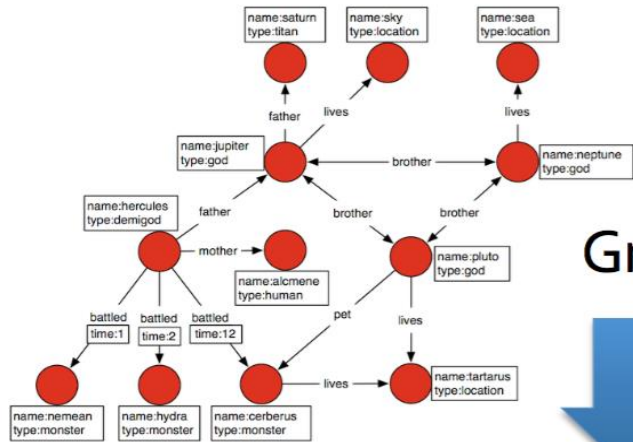
**Joint work with
Oshini Goonetilleke
and others**

Data Analytics on Twitter

- Twitter as a **platform** for research
- Diverse topics and domains in analysis
 - Content-based tasks: opinion mining
 - Analytics on the stream: event detection
 - Offline processing : spread of pandemics...
- Ad-hoc mechanisms to **capture, store** and **query** and **analyze** twitter data



Research Focus



Graph Analysis



Data Management
for Twitter
Analytics



Data Management



Twitter platform

Research Questions

- 1 What are the **requirements** of a data management framework for Twitter?
- 2 How can twitter data be **modeled** in a suitable representation with twitter-specific properties?
- 3 What is the appropriate **storage mechanism** of a large graph on-disk?
- 4 How can **indexes** speed up graph queries?

RQ₁: Requirements & Framework

- What are the **requirements** of a data management framework for Twitter?
 - How do existing systems primarily collect, represent and query twitter data?
 - What are the essential components for an integrated data management framework for Twitter?

Twitter Analytics: A Review

1) Data collection

- Tools that systematically capture data using any one of the Twitter APIs
- Third Party libraries, data resellers, Focused crawlers

2) Data Management Frameworks

- Module for crawling tweets + support for pre-processing, information extraction and/or visualization capabilities
- Generic platforms, Application Specific platforms, Visualizations

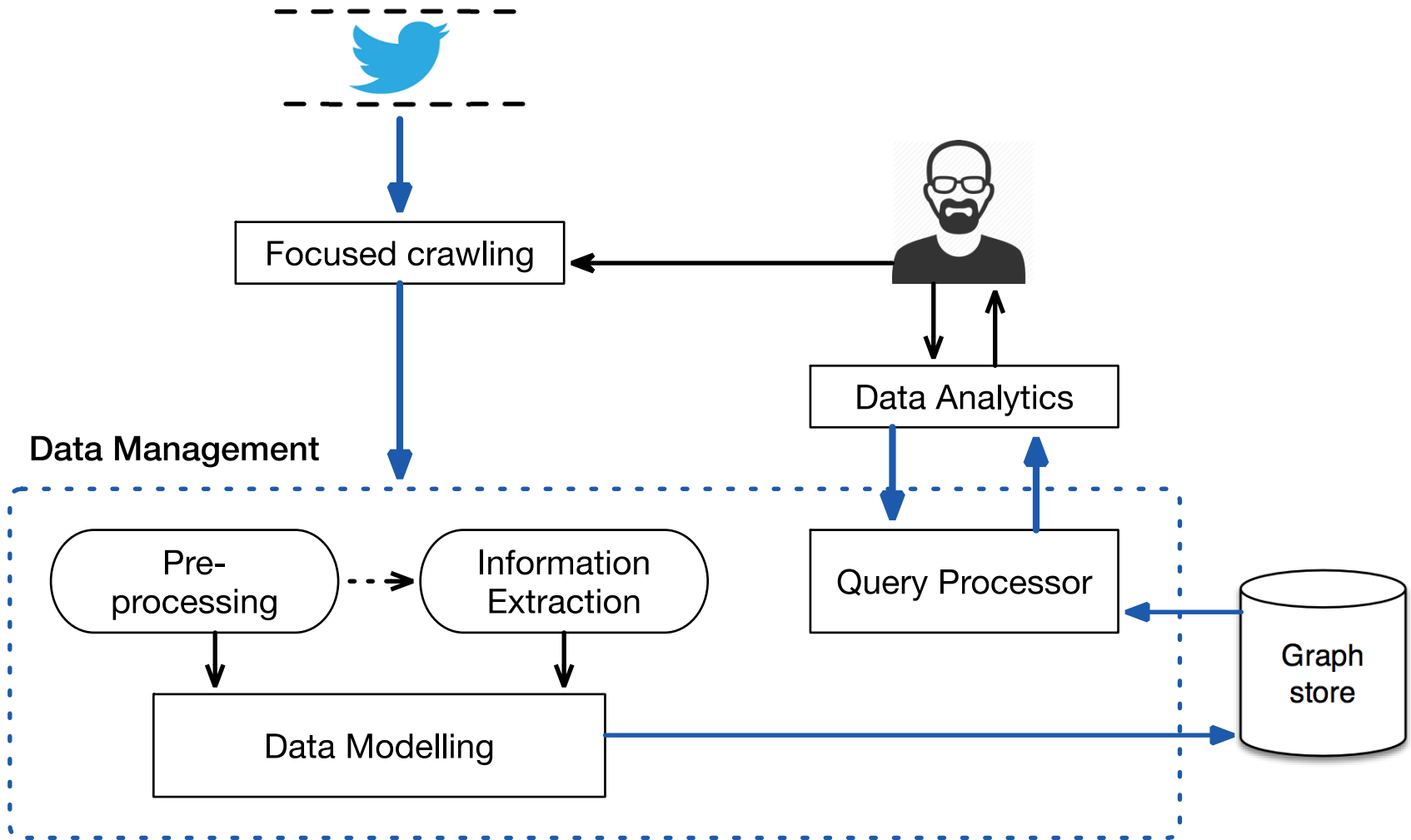
3) Query Languages and Systems

- Provides a set of primitives beneficial in exploring the Twittersphere along different dimensions.
- Generic Languages, Tweet Search, Languages for Social Networks

Research Gap

- **Data Model**
 - Existing data models do not proactively capture twitter **interactions** – user-user / tweet-tweet / user-tweet
 - **Objective:** A **graph-view** of the Twittersphere capturing Twitter interactions
- **Query system**
 - Existing systems do not operate on Twitter **Graphs** – modeled as either **relational** or **RDF**
 - **Objective:** Declarative query system tailored for querying tweets and its interactions.
- **Challenges:** management of very large graphs and optimize queries for large networks.

Integrated framework



RQ2: Data Modeling and Querying

- How can twitter data be modeled in a suitable representation with twitter-specific properties?
 - How can Twittersphere be modeled for graph traversals?
 - What are the types of queries that can be executed using graph and non-graph constructs?
 - How do queries perform on existing data management systems that use graph structures to represent data?

Microblogging Queries on Graph Databases: An Introspection

Oshini Goonetilleke¹, Saket Sathe², Timos Sellis¹, Xiuzhen Zhang¹

¹RMIT University

²IBM Research Melbourne

GRADES 2015

May 31, 2015 – Melbourne, Australia

Background and Motivation

- Increasing applications consume twitter data
 - Requirement of an **integrated** framework
- Need for efficient querying and management of large collections of twitter data modeled as graphs
- In this study:
 - **Model** basic elements of Twittersphere as graphs
 - Data **Ingestion** capabilities working with a large dataset
 - Determine the **feasibility** of running a set of proposed queries
- Twitter Data modeled as **labeled directed multi-graphs**

Graph Data Management Systems

- Natural way to analyze graph data
- Popular open source graph DBMS: **Neo4j** and **Sparksee**
- **Neo4j**
 - Fully transactional, ACID compliant
 - Methods of interaction: Declarative query language, Core API
- **Sparksee**
 - Support for APIs in many languages
 - Bit-map based representation of graphs on-disk
 - Primary navigation operations: `neighbors` and `explode`

Query Translation Example

Neo4j Cypher

```
MATCH (a:user {uid:{505}})-[:FOLLOWS]->(f)
WITH a, COLLECT(f) AS friends
MATCH (a)-[:FOLLOWS]->(:user)-[:FOLLOWS]->(fof)
WHERE a<>fof AND
      NOT (fof IN friends)
RETURN fof.uid, COUNT(*)
ORDER BY COUNT(*)
DESC LIMIT 10
```

Recommendation Query:
Top-n followees of A's followees who A
is not following yet

Sparksee API

```
value.setInteger(input);
long required = findObject(nodeType, attribute, value);
int rettype = findType(nodeType);
int followstype=findType(edgeType);
Objects friends = g.neighbors(required, followstype, EdgesDirection.Outgoing);
it=friends.iterator();

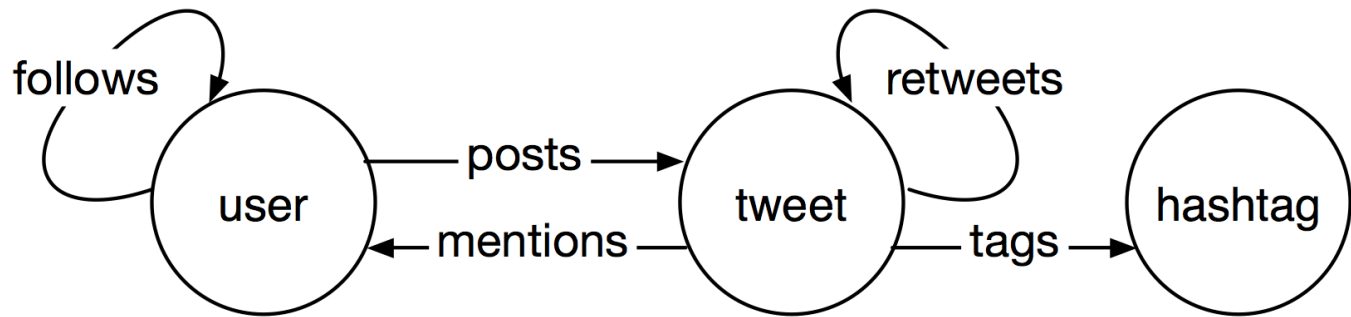
int r1=findAttribute(rettype, returnAtt);
while(it.hasNext())
{
    x=it.next();
    fof = g.neighbors(x, followstype, EdgesDirection.Outgoing);

    if(fof.exists(required)) //removing the original user
        fof.remove(required);

    fof_it = fof.iterator();
    while (fof_it.hasNext())
    {
        y = fof_it.next();
        if (!friends.exists(y))
        {
            g.getAttribute(y, r1, value);
            int key=value.getInteger();
            if (h.containsKey(key)) {
                h.put(key, (Integer) h.get(key) + 1);
            } else {
                h.put(key, 1);
            }
        }
    }
}
fof_it.close();
fof.close();
}
h = sortByValue( h );
for (Map.Entry<Integer, Integer> entry : h.entrySet()) {
    System.out.println(entry.getKey() + "\t" + entry.getValue());
    if(j++==10) break;
}
}
```

Database Schema for Twitter case study

3 Node Types, 4 Relationship types



Initial Dataset contain: **users**, **tweets** and **follows** relationships

Nodes		Relationships	
user	24,789,792	follows	284,000,284
tweet	24,000,023	posts	24,000,023
hashtag	616,109	mentions	11,100,547
		tags	7,137,992
Total	49,405,924	Total	326,238,846

Experiments

- Feasibility Analysis of
 - Data Ingestion
 - Query Processing
- Configuration
 - Standard Intel Core 2 Duo 3.0 GHz
 - 8GB of RAM
 - Non-SSD HDD
- Version of the databases
 - Sparksee 5.1, research license up to 1 billion objects
 - Neo4j 2.2.M03, Community Edition
 - Java APIs for both databases

Data Ingestion

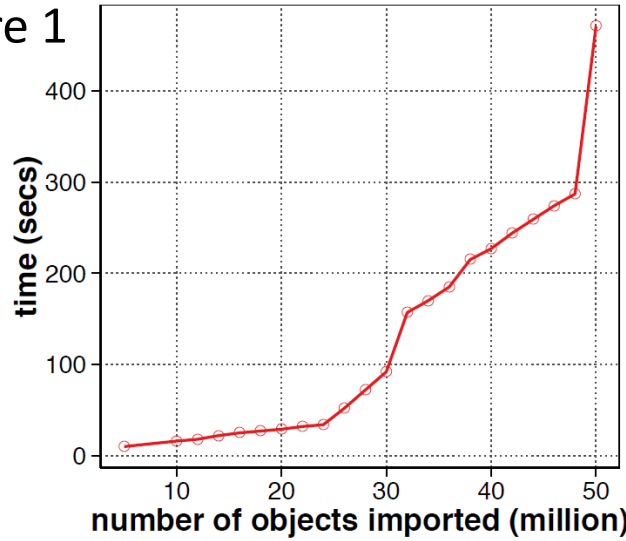
- Simulate batch loading procedures of a graph
 - Neo4j **Import tool**
 - Effectively manages memory
 - Cannot create indexes while loading
 - Sparksee **scripts**
 - Recovery and rollback disabled
 - Cache size: 5GB
 - Extent size: 64KB
 - Neighbors not materialized
- Cannot perform incremental loading

Figure 1

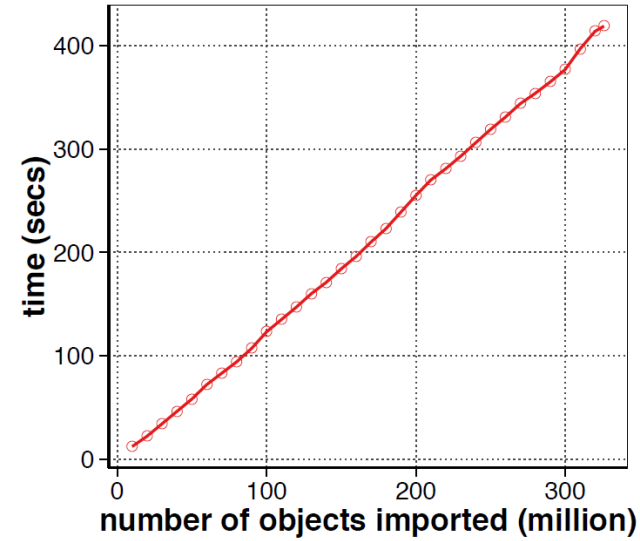
Neo4j

Total Time : 45 min

Size on disk: 20.8 GB



(a) Nodes



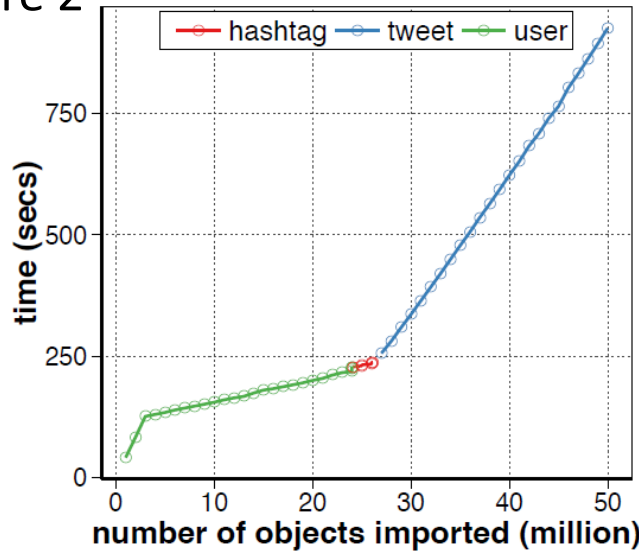
(b) Edges

Figure 2

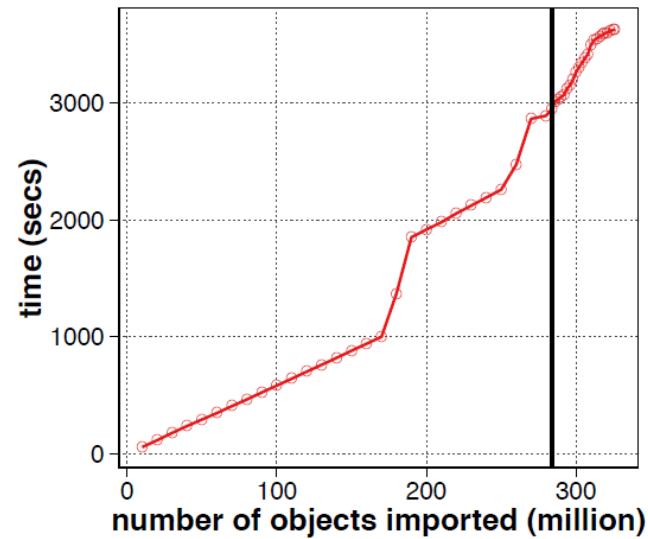
Sparksee

Total Time : 72 min

Size on disk: 15.1 GB



(a) Nodes



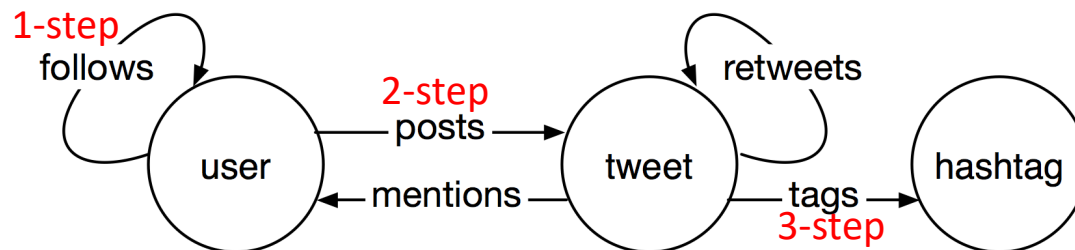
(b) Edges

Query Workloads

- Set of representative queries that are useful in the microblogging context
- Basic
 1. Select
 2. Adjacency
- Advanced
 3. Co-occurrence
 4. Recommendation
 5. Influence
 6. Shortest path

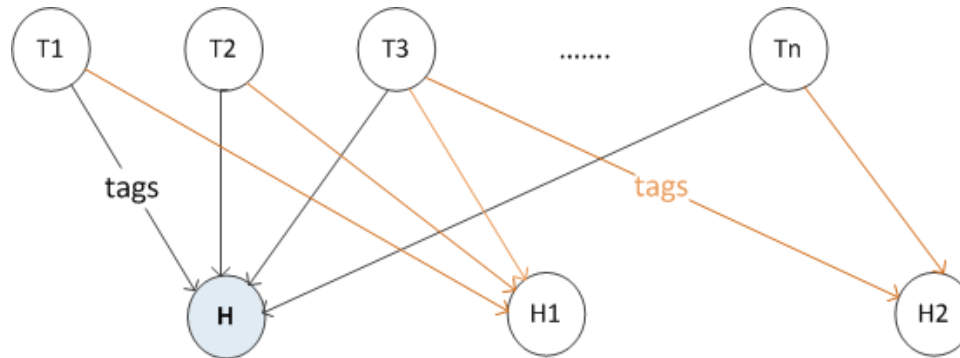
Query Processing - Examples

- Select
 - All Users with a follower count greater than a user-defined threshold
- Adjacency
 - 1-hop: All the followees of a given user **A**
 - 2-hop: All the Tweets posted by followees of **A**
 - 3-hop: All the hashtags used by followees of **A**

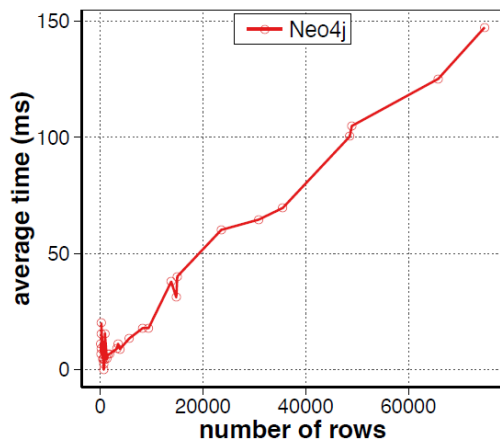


Co-occurrence queries

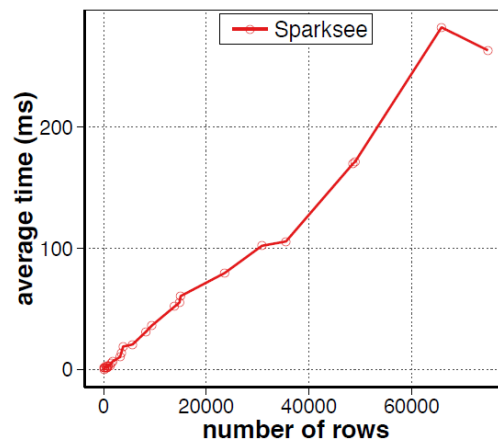
- Co-occurrence of **hashtags** and **mentions**
- Finding co-occurrences translates to a 2-step process:



- **Examples:**
 - Q3.1: Top-n users most mentioned with user **A**
 - Q3.2: Top-n most co-occurring hashtags with hashtag **H**



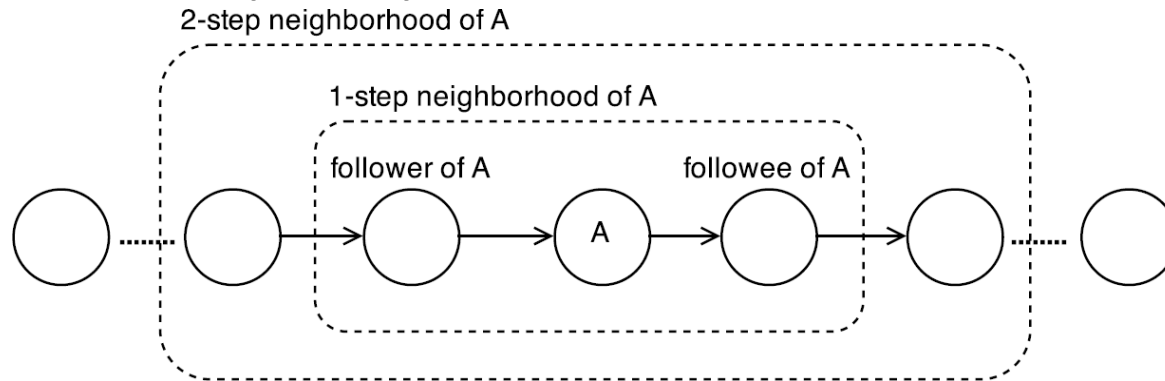
(a) Q3.1 – Neo4j



(b) Q3.1 – Sparksee

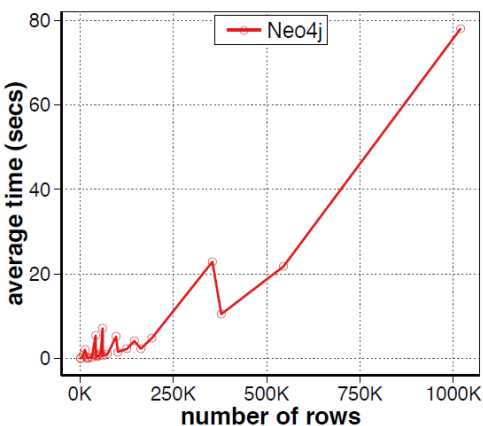
Recommendation Queries

- Recommendation from a user's **local community**
- Only top-n most frequently followed users are considered relevant

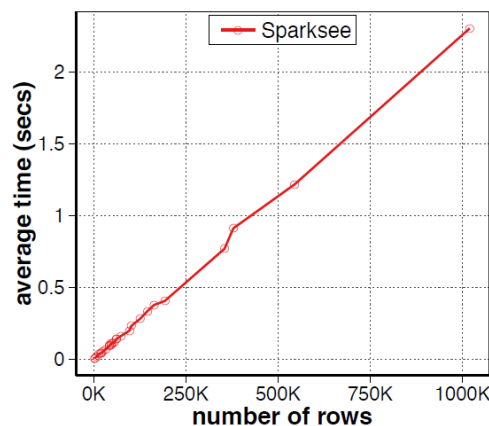


- **Examples:**

- Q4.1: Top-n followees of **A**'s followees who **A** is not following yet
- Q4.2: Top-n followers of **A**'s followees who **A** is not following yet



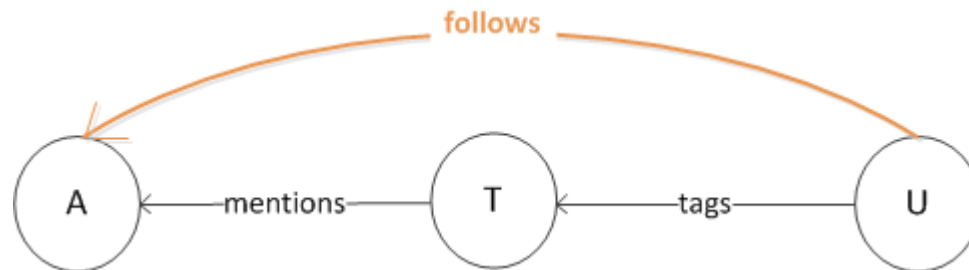
(c) Q4.1 – Neo4j



(d) Q4.1 – Sparksee

Influence Queries

- **Current** and **potential** influence a user has on its community
- An intuitive translation of this query as a function of user **mentions**:
 - Current influence of A is a union of the two sets: Most frequent users who mention A and who are already followers of A
 - Potential influence: Difference of the two sets



- **Examples**:
 - Q5.1: Top-n users who have mentioned **A** who are followees of **A**
 - Q5.2: Top-n users who have mentioned **A** but are not direct followees of **A**

Other queries

- Shortest-path queries
 - Shortest path between two users where they are connected by follows edges
- Deriving new queries
- Suppose you want to answer this question:
 - user A is interested in topic H who should A follow?
- Combine:
 1. Get all the hashtags that co-occur with the given hashtag H
 2. Get the most retweeted tweets mentioning those hashtag
 3. Get the original users of those retweets
 4. Order the users based on the shortest path length from A

Observations

- Alternate Solutions
 - Cypher vs. Neo API
 - Performance differences in Cypher queries
 - Sparksee Traversals vs. Navigation operations
- Overhead of aggregates
 - Retrieving the top-n results
 - factors for increased performance: remove ordering and de-duplication, limiting results
 - Sparksee: no way to limit the result set
- Problems with cold-cache
 - Warm-up time for Neo4j caches

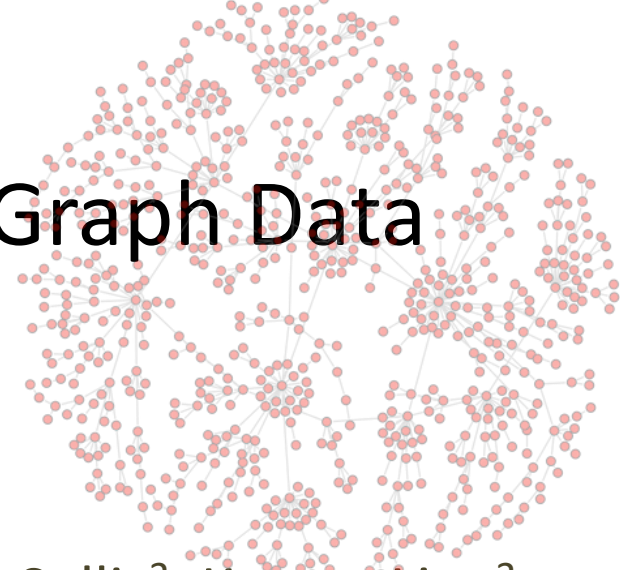
Conclusions and Future Work

- Extension to other domains where a similar schema is applicable (Facebook, LinkedIn etc.)
- Not tested for offline graph analytics
- Database tuning for optimal configuration to suit query workloads
- Generation of the graph on-the-fly
 - Testing for update workloads
- Semantic-aware strategies to speed up the queries by graph partitioning and storage

RQ₃: Storage representation of the graph

- What is the appropriate storage mechanism of a large graph on-disk?
 - How can we **partition** the graph such that the I/O can be minimized for known query workloads?
 - What is a suitable **representation** of a graph on disk?

Edge Labeling Schemes for Graph Data



Oshini Goonetilleke¹, Danai Koutra², Timos Sellis³, Kewen Liao³

¹ RMIT University, Australia

² University of Michigan, USA

³ Swinburne University of Technology, Australia

SSDBM

June 27-29, 2017 – Chicago IL, USA



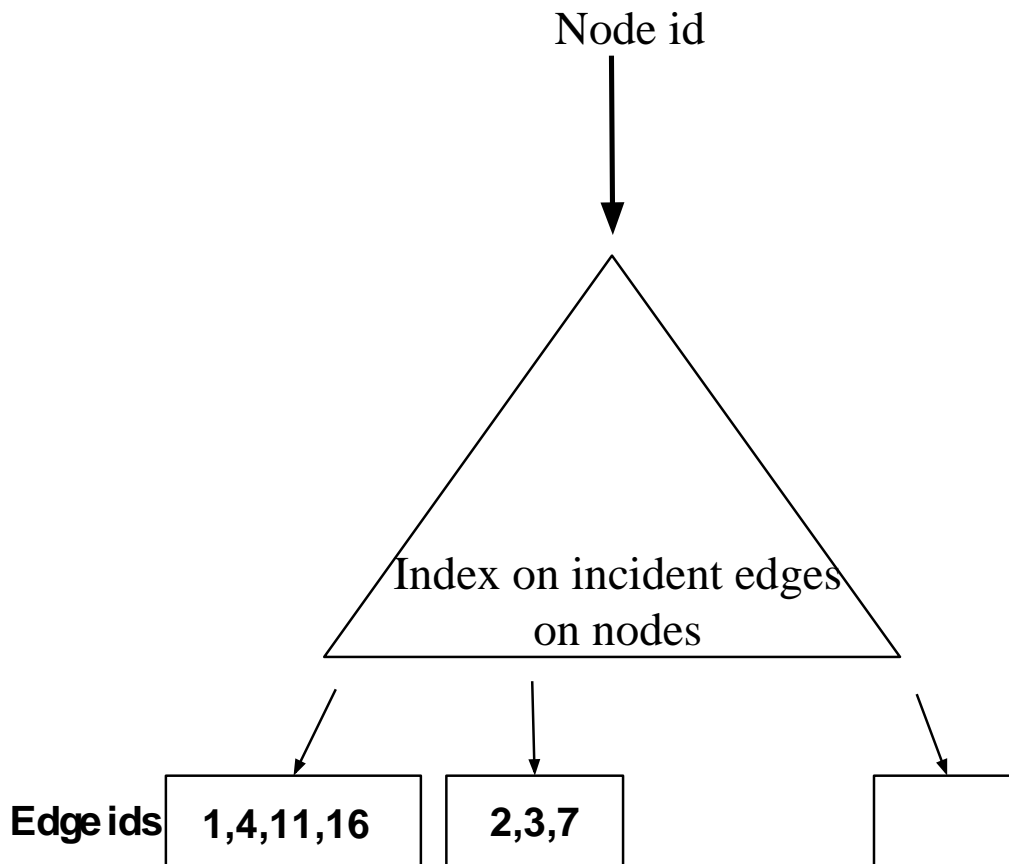
Background

- Graph data is everywhere!
- Management and analysis of networked-data
 - Data storage, management, querying
- Tools
 - GDBMS – Neo4j, Sparksee
 - Network analysis packages – SNAP, NetworkX
- Graph representation
 - Matrix, Adjacency node/edge lists

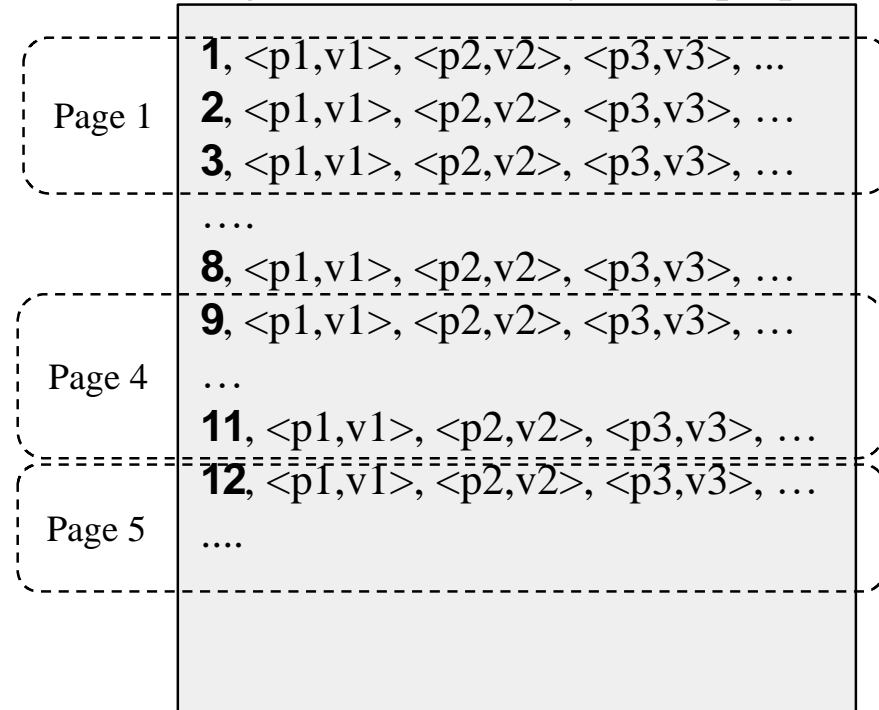
Improving query performance

- Node re-ordering
 - SlashBurn, Shingle
- Graph compression
- Graph partitioning, clustering
 - METIS

Motivation



Data file on disk sorted by edge id
(edge ids followed by all its properties)



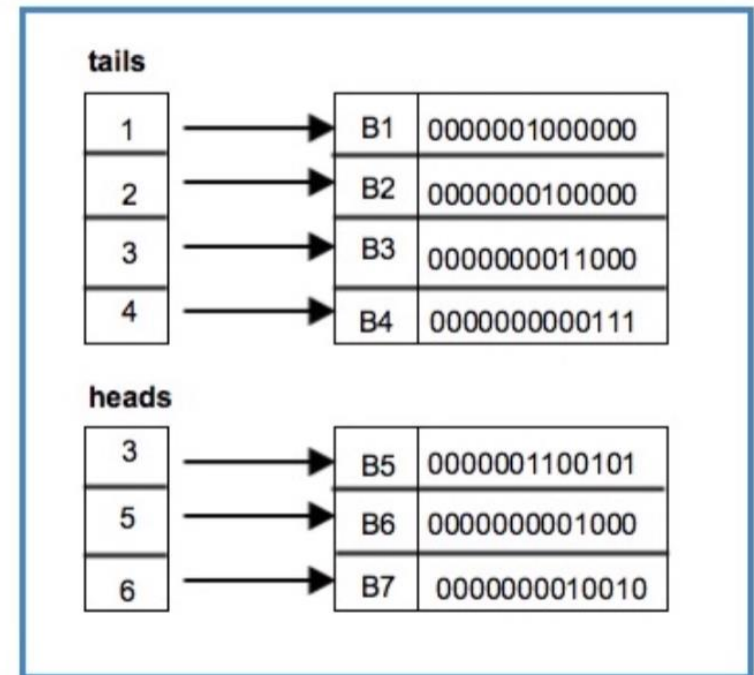
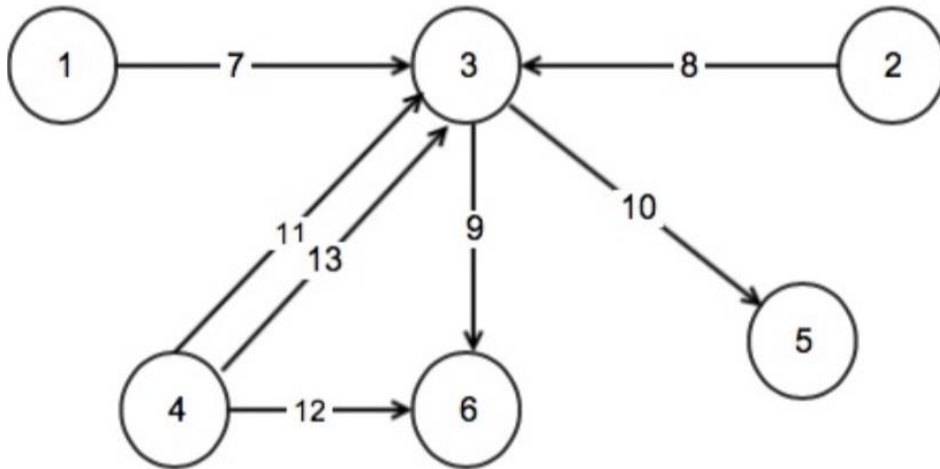
Goal: Assign edge labels to achieve improved disk locality for efficiently answering typical **neighborhood** queries, **without modification** to the storage internals of the graph system.

Objectives

- Given a directed graph, how should we label both its outgoing and incoming edges to achieve **better disk locality** and optimize **neighborhood-related** queries?
 - Can we **speed up** popular edge-based neighborhood graph queries using an edge labeling?
 - Can we observe improved disk I/O as a result of **better locality** when storing graph on disk?
 - Do better edge labeling schemes show **benefit in storage** compared to a random ordering and other baselines
 - Can **streaming** graph partitioning benefit from an already **locality improved** edge-list?

Sparksee

- Bit-map based representation

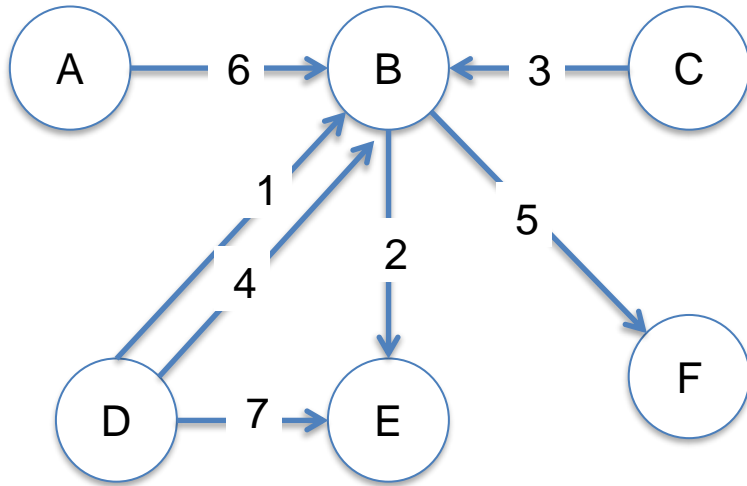


Outline

- Background and Motivation
- Edge consecutiveness
- Baselines and proposed method
- Experiment Results
- Application: Graph streaming
- Conclusions and Future work

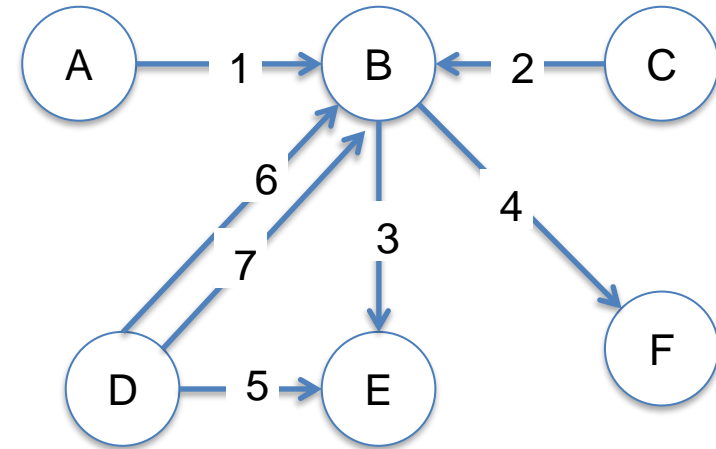
The effect of different ordering strategies

a) Random Ordering



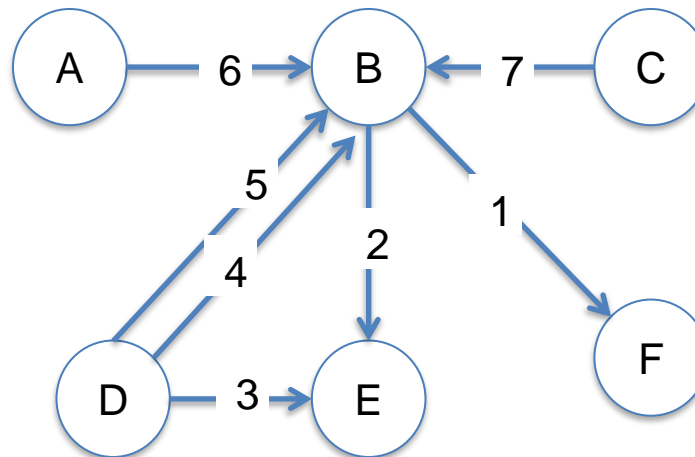
$C_{in}(G) = 0.4, C_{out}(G) = 0.4$

b) Source-based Ordering



$C_{in}(G) = 0.4, C_{out}(G) = 1.0$

c) Perfect Ordering



$C_{in}(G) = 1.0, C_{out}(G) = 1.0$

Edge consecutiveness

- **Consecutiveness:**
 - Directed graph $G(V,E)$ and a mapping $\pi : E \rightarrow \mathbb{Z}$ of edges to integer ids
 - Incoming edge consecutiveness of vertex v ,
 $C_{in}(v)$ = no. of pairs of incoming edges with consec. ids
- **Maximization** of total consecutiveness of graph G :

$$C(G) = \frac{1}{|E| - n_{in}} \sum_{v \in V} C_{in}(v) + \frac{1}{|E| - n_{out}} \sum_{v \in V} C_{out}(v)$$

where $C_{in}(v)$ is the number of pairs of incoming edges with consecutive ids under some order and n_{in} and n_{out} are the number of nodes with > 1 incident edges of their respective edge types.

Edge Labeling Schemes

- **Random**
 - Edges are listed in an order given by a random permutation.
- **consecIN**
 - Incoming edges of each node are labeled consecutively, edge IDs are sorted over the edge target/destination nodes.
- **consecOUT**
 - Outgoing edges of each node are labeled consecutively, edgeIDs are sorted over the source nodes.

Proposed: GrdRandom

- Does not favor a single direction
- Consider a random permutation of the nodes to inform the visit order
- Idea
 - alternate between the edge type we number so that edges in a single direction are not favored

Proposed Method: FlipInOut

- Incorporates 3 ideas:
 - **Alternate**: Flips between numbering incoming and outgoing edges to balance consecutiveness
 - **Prioritize**: High degree nodes are given higher priority and are labeled earlier
 - **Terminate early**: For large graphs, order the last $|E| \times \delta\%$ edges a greedy fashion. Eliminates the frequent restart problem
- Alternate and Prioritize locally maximize the individual consecutiveness at a vertex. i.e. $C(v)$

Evaluation

- Query Performance (runtime and scalability)
 - **FoF Queries**: incoming and outgoing
 - **Shortest Path**: bidirectional bfs
 - **Edge Property**: pattern matching on edges
- Disk I/O performance
- Disk storage benefit

Dataset

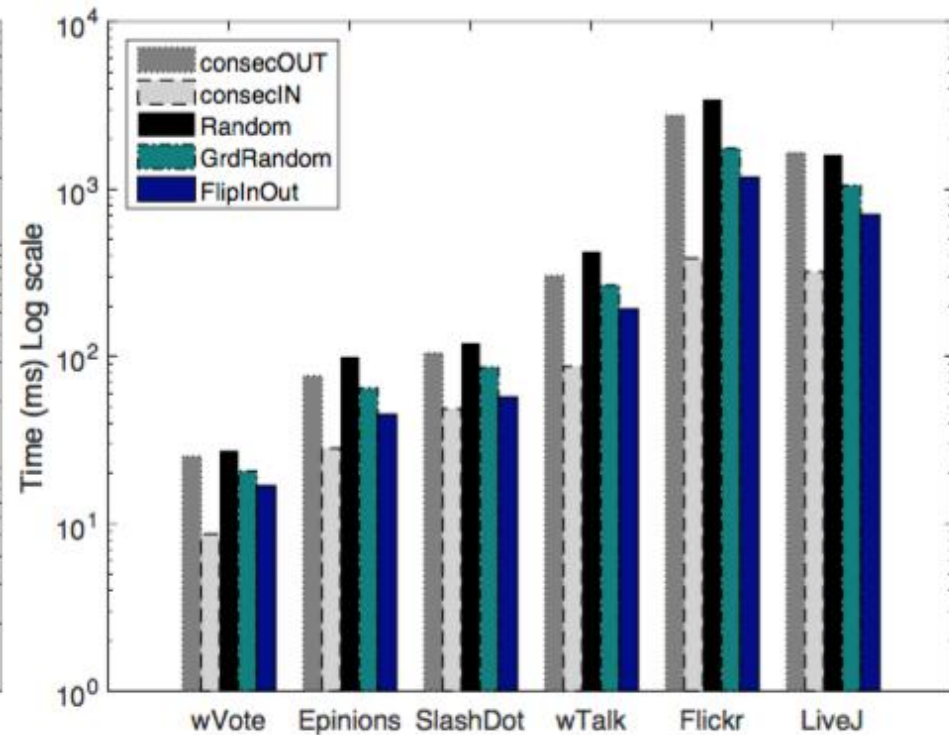
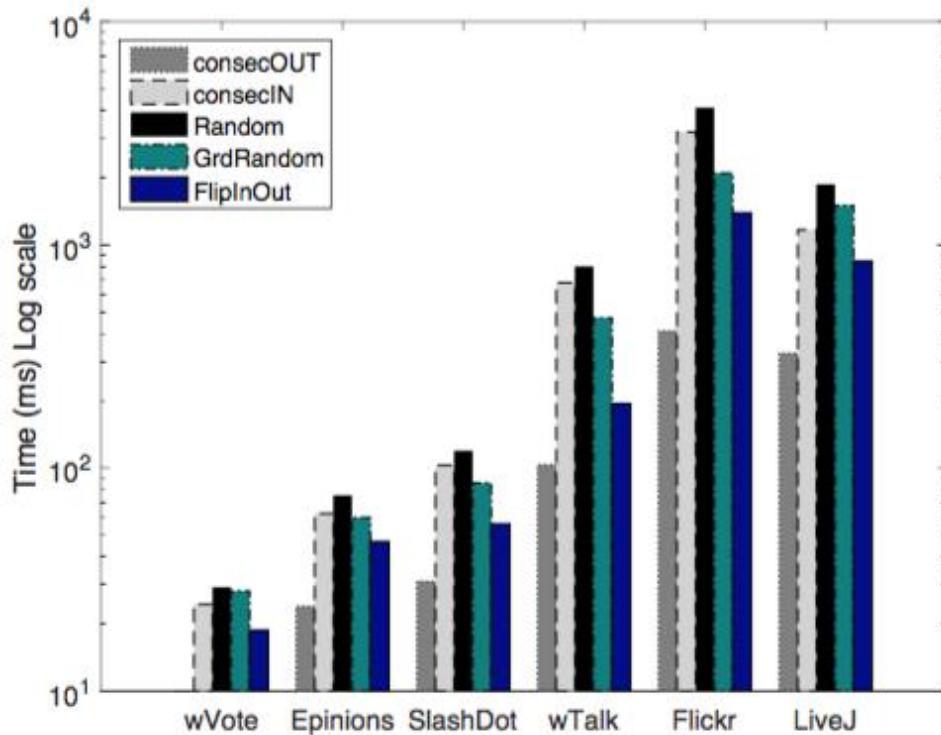
Dataset	Nodes	Edges	Avg. OIR	LCC	ACC	Description
WikiVote	8,297	103,690	0.73 / 0.27	0.38	0.14	who-votes-whom
Epinions	75,879	508,837	0.56 / 0.43	0.87	0.14	who-trusts-whom
Slashdot	82,168	948,464	0.43 / 0.56	0.96	0.06	social network
WikiTalk	2,394,385	5,021,410	0.03 / 0.96	0.29	0.05	Wiki talk network
Flickr	2,585,568	33,140,018	0.42 / 0.57	0.82	0.10	social network
LiveJ	4,847,571	68,993,773	0.49 / 0.50	0.95	0.27	social network

- **OIR**: avg. ratio of outgoing and incoming edges over the total number of edges
- **LCC**: number of edges in the largest strong connected component
- **ACC**: avg. clustering coefficient

Query Performance (1)

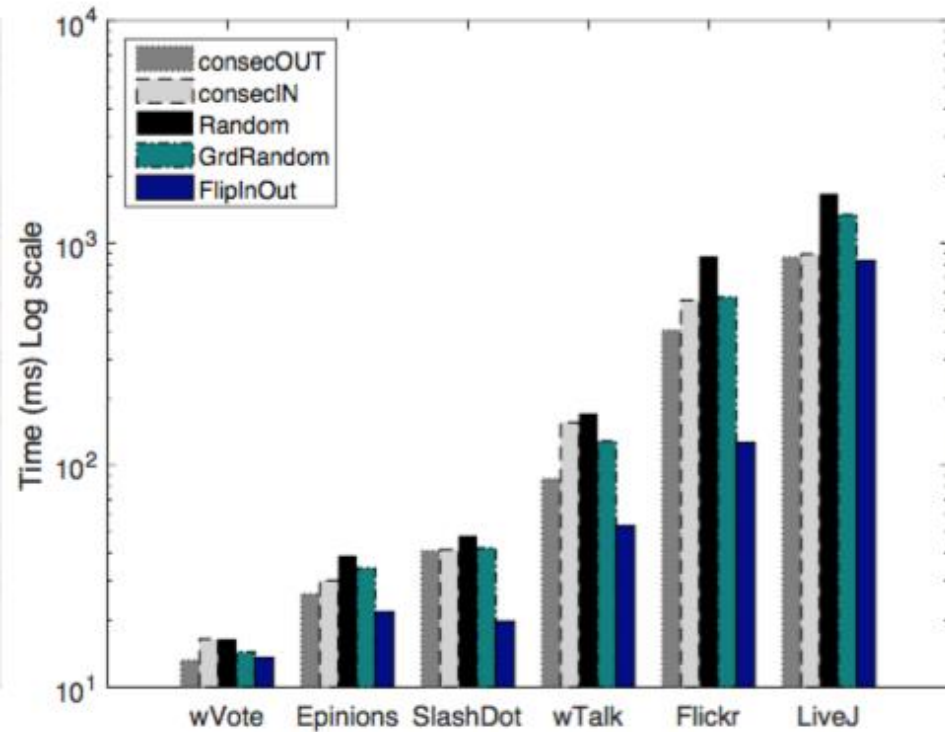
Outgoing friend-of-friend

Incoming friend-of-friend

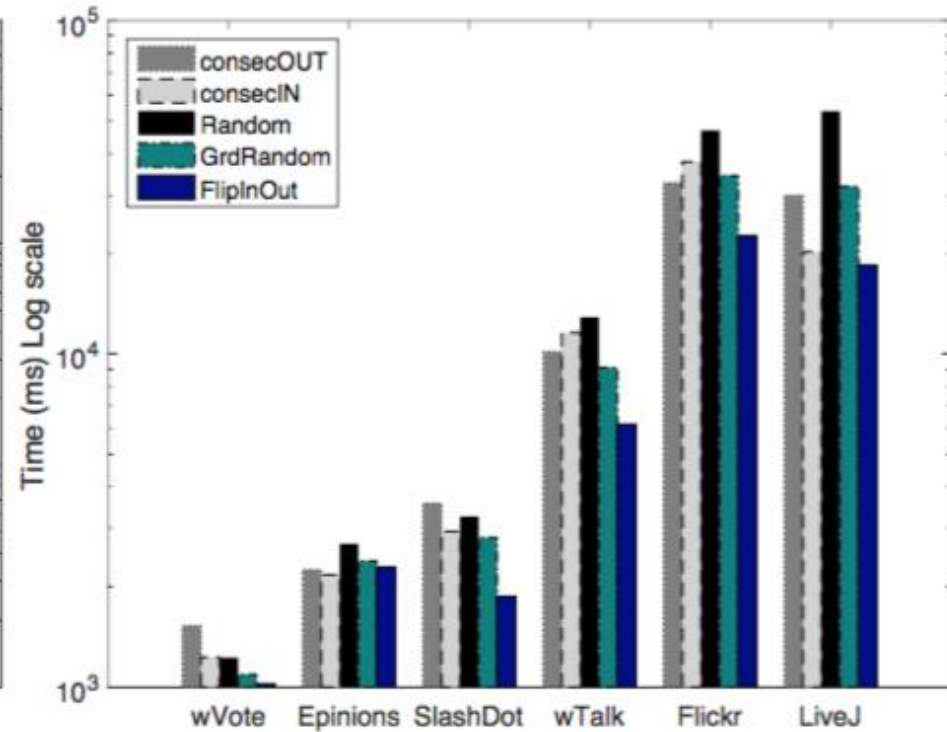


Query Performance (2)

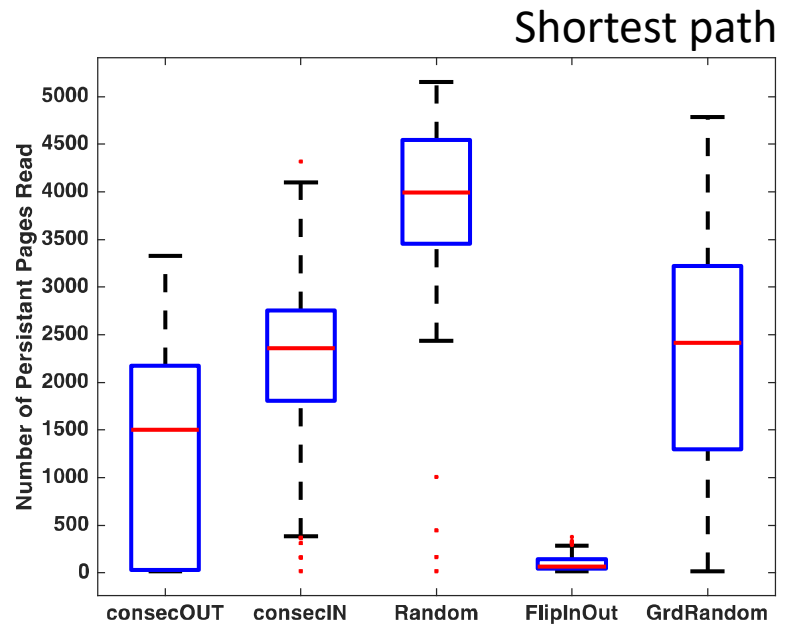
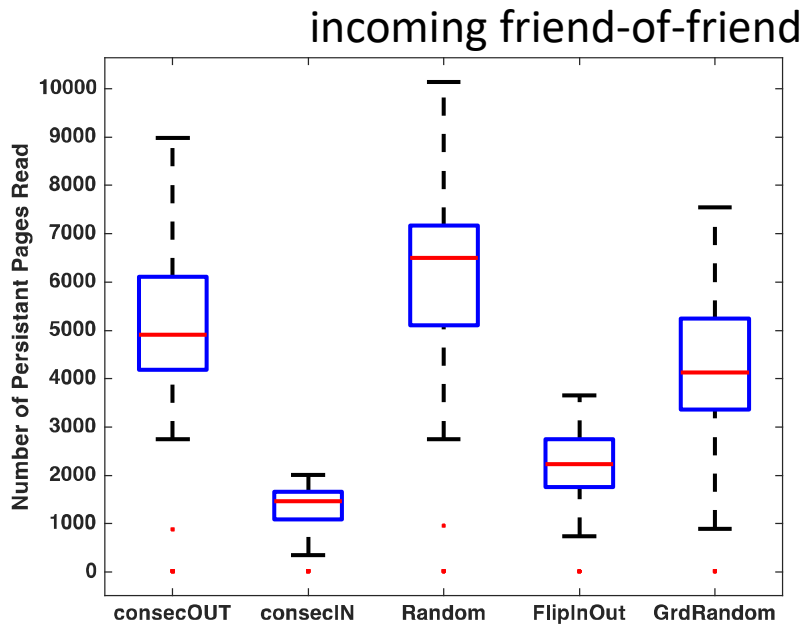
Shortest Path



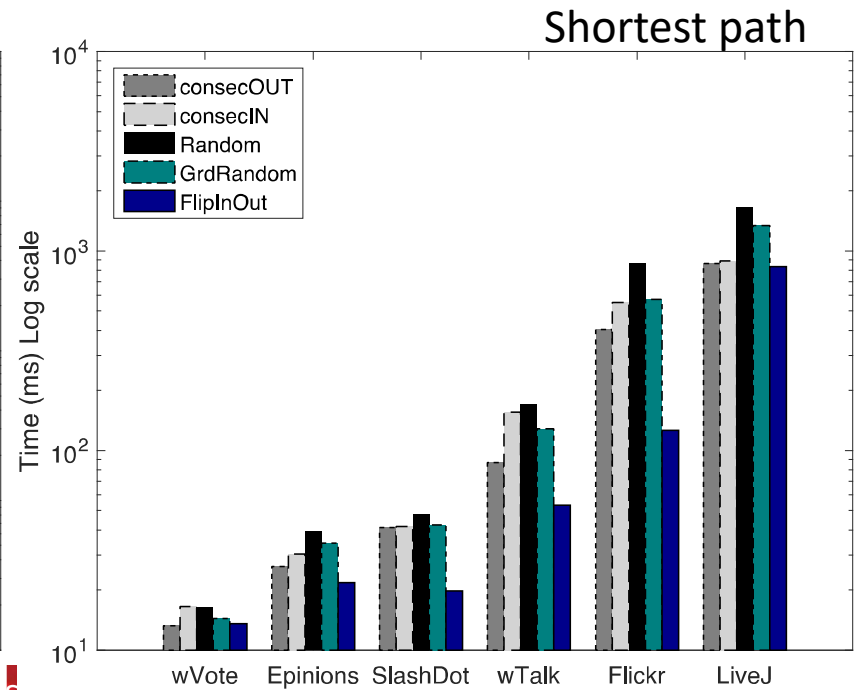
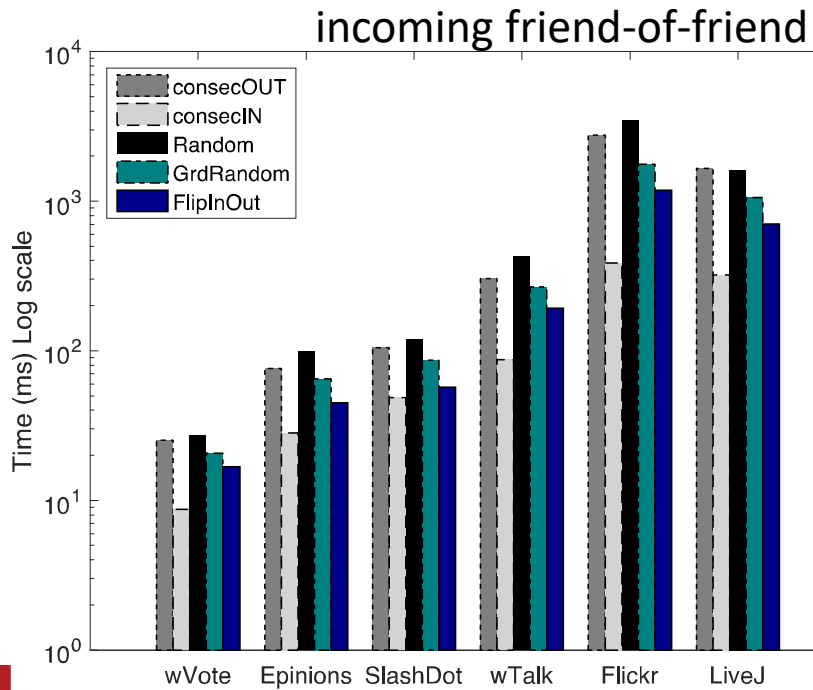
Edge Property Query



Disk I/O performance on Flickr



Query performance on Flickr



Disk storage benefit

- 10%–27% **reduction** compared to a database with random ordering
- The low compression benefit compared to other baselines due to disarray of either out or in bitmaps

Application: Streaming Graph Partitioning

- **FlipInOut** already has improved locality. Can graph partitioning benefit from this?
- Partitioning objective
- **FlipCut** (based on FlipInOut)
 - Considers one edge at a time based on three rules
 - One pass over the edges $O(|E|)$

Baseline methods

- Streaming partitioning algorithms
 - FENNEL, LDG
- Baselines
 - BFS+LDG
 - Random+LDG
 - Hashing

Results

- Smaller graphs, for $k = 4$ partitions edgecuts in FlipCut compared to
 - LDG variants: 8% to 42% reduction
 - Hashing methods: 26%- 50% reduction
- Larger graphs:

Data	k	Natural + LDG	Random + LDG	FLIPCUT	Hashing
Flickr	12	55.85	85.45	27.04	91.7
	24	61.65	89.74	54.12	95.8
LiveJ	24	41.01	87.88	63.67	95.8
	50	46.99	90.56	70.16	98.0
	100	51.74	92.04	75.00	99.0

Summary

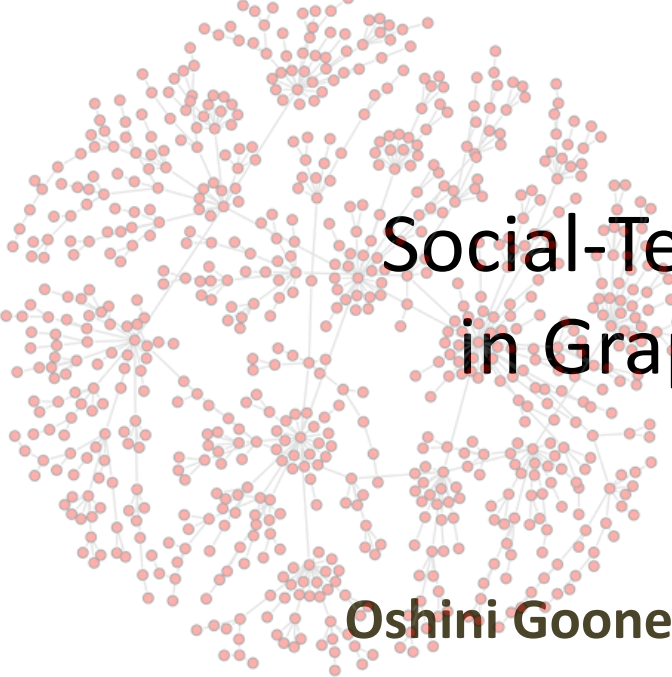
- Proposed two edge labeling schemes
 - GrdRandom and FlipInOut
 - Balances and maximize the edge consecutiveness
- Experimentally evaluated the benefit in
 - Query performance
 - Disk I/O
- Introduced FlipCut
 - Effective one-pass graph partitioning

Future Work

- Proving NP-hardness of edge consecutiveness maximization problem
- Correlation studies between node and edge orderings
- Analyse other less strict measures such as gap minimization
- Online edge-labeling for applications such as graph streaming

RQ4: Socio-Textual Indexing

- How a full-text search can be seamlessly integrated into graph traversals within a graph database system?
- What are the constructs we need to build indices for this?



Social-Textual Query Processing in Graph Database Systems

Oshini Goonetilleke¹, Timos Sellis², Xiuzhen Zhang¹

¹ RMIT University

² Swinburne University of Technology

Australasian Database Conference
May 23-25 2018, Gold Coast, Australia



Graph data

- Graph data from the real-world
 - Combination of connections + *attributes* + *text*



Social connections

aggregated attributes

J.K. Rowling ✓
@jk_rowling

For contact details, see website (sorry, but too many DMs to read)

📍 Scotland
🌐 jkrowling.com

location, position, URL, joined date attributes

Tweets **Tweets & replies** **Media**

📌 Pinned Tweet

J.K. Rowling ✓ @jk_rowling · May 2
It's that anniversary again. This year, I apologize for the deaths of the characters who die during the #BattleofHogwarts, but who laid out the rules for the winners who'd win it. I refer, of course, to Dobby the house-elf.

💬 6.5K 🔄 95K ❤️ 324K

free text

- How can we efficiently query them?

Problem



- GDBMS supports large-scale labeled, attributed multi-graphs from the real-world
 - Neo4j, Titan and many more...
 - They are optimized for **graph traversals**
- Dimensions in a real-world graph are fundamentally supported by **different** models:
 - Graph topology and its traversals: by GDBMS
 - Queries on text: by specialised full-text indexing schemes
- **Our goal:** Investigate how a **full-text search** can be seamlessly integrated into graph traversals within a graph database system.



Motivation

- Well established methods for **graph** and **text** indexing
 - Specialised graph indexing schemes, inverted indexes
 - Different data management goals
 - Do we have convenient ways to seamlessly **integrate** them?
- Sample query examples:
 - **LinkedIn**: Who's most likely in my network to introduce me to someone who is skilled in **python**?
 - **Twitter**: Finding 5 users who are socially close to a query user, and who have used terms relevant to **#AusOpen**?

Outline

- Problem and Motivation
- Problem Definition
- Baseline algorithms
- Proposed Approach
- Experiment results
- Conclusions and future work

Problem Definition

Let $G(V,E)$ be a graph with users represented as $v \in V$ and a social interaction represented as $e \in E$. $D(v)$ denotes the set of keywords associated to v . $V(w)$ denotes the set of vertices containing the keyword w (where $V(w) \subseteq V$).

- Social proximity:

$$s(v_i, v_j) = \frac{sdist(v_i, v_j)}{sdist_{max}}$$

where $sdist(v_i, v_j)$ is the length of the shortest path connecting v_i and v_j .

- Text relevance:

$$t(q.w, D(v)) = \frac{tf(q.w, D(v)) \times idf(q.w)}{tdist_{max}}$$

-- denotes the similarity between a query term $q.w$ and $D(v)$ adopting standard tf-idf model.

- Denominators normalises the respective scores [0,1].
- Combined ranking function: $f(v) = \alpha \cdot (t(q.w, D(v))) + (1 - \alpha) \cdot (1 - s(q.u, v))$
where $0 \leq \alpha \leq 1$ denotes the relative significance of the individual components

Top-k Social Textual Ranking Query (kSTRQ)

- kSTRQ, on a graph $G(V,E)$ can be expressed as a triplet $q=(u,w,k)$ where,
 - $u \in V$ is the query vertex in G ,
 - w is a keyword and,
 - k is a positive integer denoting the number of output records.
- kSTRQ query returns a result set R that contains k nodes $v \in V - \{q.u\}$ with the highest $f(v)$ values.

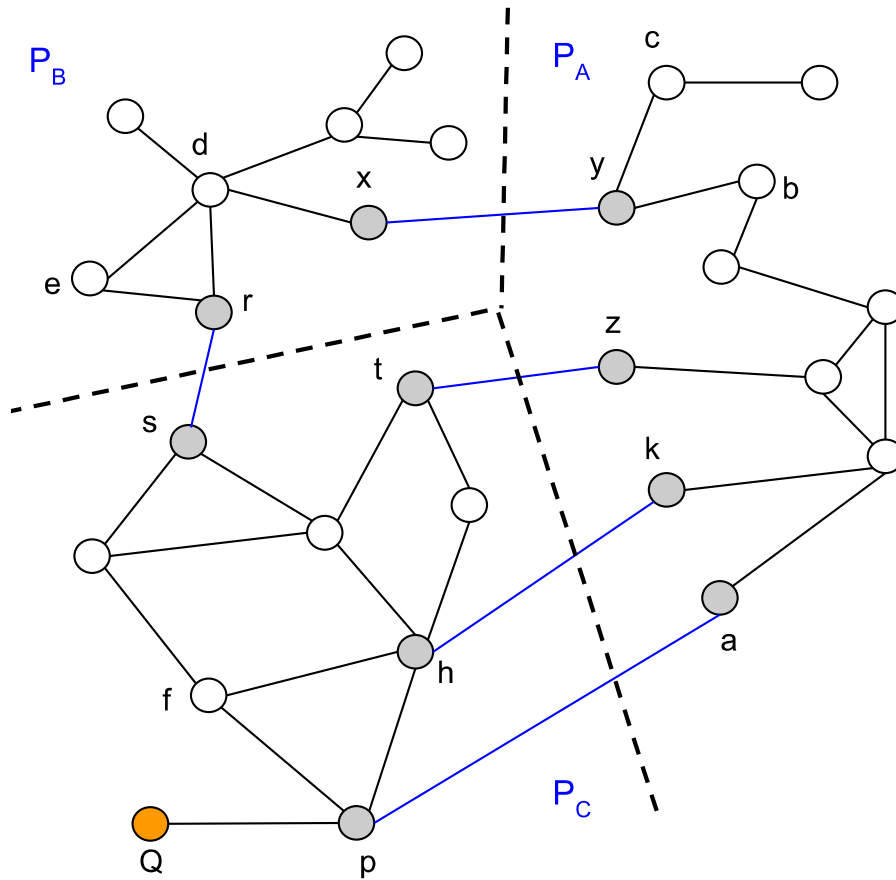
$$f(v) = \alpha \cdot (t(q.w, D(v))) + (1 - \alpha) \cdot (1 - s(q.u, v))$$

Baseline algorithms

- Text-First Algorithm (**TFA**)
 - Iterates through the text list containing q.w
- Social-First Algorithm (**SFA**)
 - Iterates through users in increasing social distance to q.u
- Early termination variations (**TFA_ET** and **SFA_ET**)
 - Enables traversing through respective lists partially
- Threshold algorithm (**TA**) [1]
 - Previous algorithms are ignorant of one dimension; may incur unnecessary traversals
 - TA iterates through both lists simultaneously for efficient pruning
 - Require **sorted** and **random** access to ranked social and text lists

[1] Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci. 66(4), 614–656 (2003)

Proposed PART_TA algorithm



partition	$T(p_i, q.w)$ for keyword w
P_A	a: 5, b: 5, c: 1
P_B	d: 8, e: 2
P_C	f: 2

- Boundary nodes
- Pre-computation

- Partitioned graph + text lists to map to partitions
- Rationale for partitions
- Inspired by the threshold algorithm, running on partitions

Experiments

- Real datasets with different characteristics

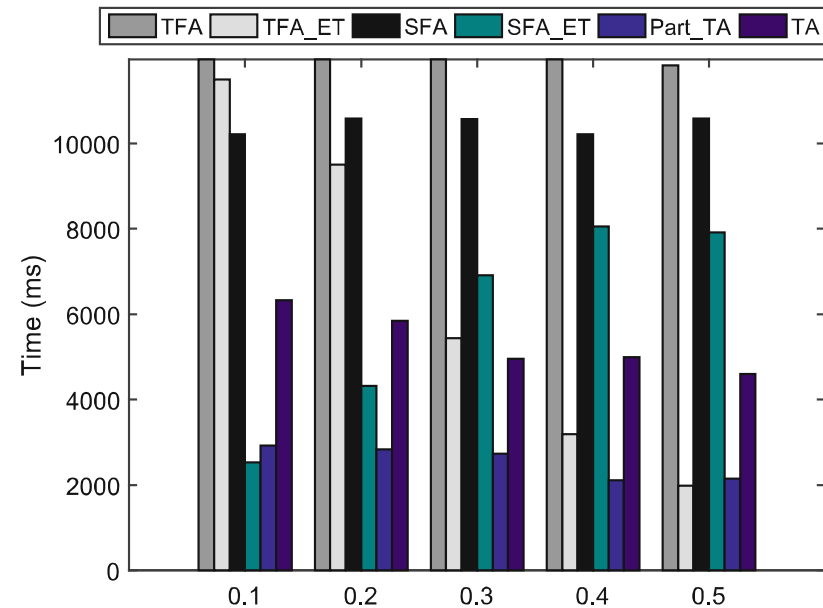
Dataset	Nodes	Edges	Avg/Max Degree	Diameter	No. of unique keywords	Description
Twitter	87,349	306,249	7.8 / 230	15	1.3M	who-follows-whom social network
AMiner	1,057,194	3,634,124	4.9 / 551	24	2.9M	Academic Co-authorship network
Flickr	424,169	8,475,790	39.6 / 11,930	8	340K	Friendship social network

- Neo4j
 - allows manipulation of text via Lucene
- Partitioned with METIS [2]
- Effect of parameters by varying
 - Preference parameter α (default:0.3)
 - number of objects k (default:10)
- Effect of partitions expanded

[2] Karypis, G., Kumar, V.: Multilevel k -way partitioning scheme for irregular graphs. Journal of Parallel and Distributed computing 48(1), 96–129 (1998)

Results: Effects of α (0.1 to 0.5)

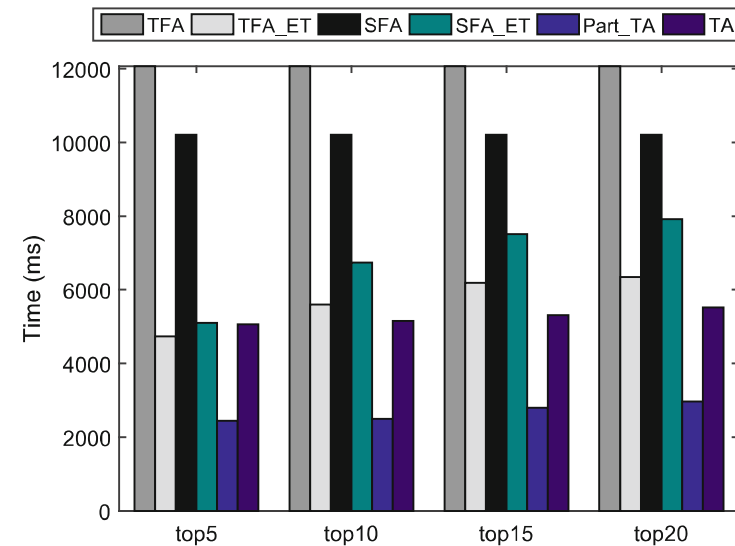
- With increasing α , **TFA_ET** and **SFA_ET** performs better and worse respectively (across the datasets)
- **PART_TA** performs better (76%) compared to **TFA_ET** and **SFA_ET** variations
- Although **_ET** variations are better for edge cases, this unstable behaviour is not suitable for the general case.



(b) AMiner

More results

- Results: Effects of k (5 to 20)
 - More processing and traversals required for increased k
 - Rate of growth higher for early termination variations compared to **PART_TA**
- Percentage of partitions expanded with varying α
 - This is indicative of the fraction of partitions to expand to retrieve top- k
 - Examines the effect of the density of graphs have on expanded partitions
 - Refer to paper for details



(b) AMiner

Conclusions and Future work

- Investigated the **kSTRQ** query that requires combined graph traversal and text search in a graph database system.
- Observed better **performance** and **robustness** of the proposed approach, especially on larger graphs
- Future work:
 - Investigate the effects of different social and text relevance metrics
 - Explore other partitioning strategies that take into account both social connections and node attributes (such as text)
 - Investigate the effect on varying the number of partitions.

References

- O. Goonetilleke, T. Sellis, X. Zhang, and S. Sathe, “*Twitter Data analytics: A Big Data Management Perspective*”, ACM SIGKDD Explorations, Vol. 16, No. 1, June 2014.
- O. Goonetilleke, S. Sathe, T. Sellis, and X. Zhang, “*Microblogging Queries on Graph Databases: An Introspection*”, Proceedings of the 3rd International Workshop on Graph Data Management Experiences and Systems (GRADES 2015), SIGMOD 2015 Workshops, Melbourne, Australia, May 2015.
- O. Goonetilleke, D. Koutra, T. Sellis, and K. Liao, “*Edge Labeling Schemes for Graph Data*”, Proceedings of the 29th International Conference on Scientific and Statistical Database Management (SSDBM 2017), Chicago, United States, June 2017.
- O. Goonetilleke, T. Sellis and X. Zhang, “*Social-Textual Query Processing on Graph Database Systems*”, Proceedings of the 29th Australasian Database Conference (ADC 2018), Gold Coast, Australia, May 2018.
- O. Goonetilleke, D. Koutra, K. Liao and T. Sellis, “*On effective and efficient graph edge labeling*”, Distributed and Parallel Databases, vol 37, no 1, March 2019.