

# Locality Optimization for traversal-based Queries on Graph Databases

**Fabian Klopfer**

Databases and Information Systems Group  
Department of Computer and Information Science  
University of Konstanz, 30.04.2021

---

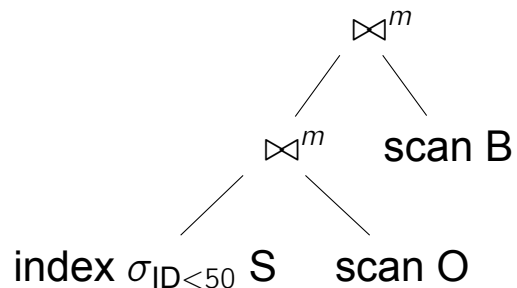
# Motivation

Current state of performance-optimized

- relational databases: accesses are made as sequential as possible.
- graph databases: access is often random.

## Example I

Show me all boats owned by sailors with an ID less than 50:

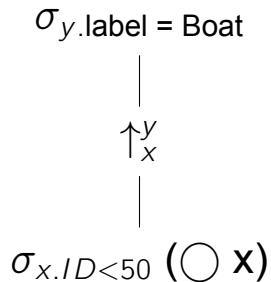


Reads are mostly sequential.

⇒ Prefetch & cache hit.

## Example II

Nodes are Sailors and Boats, relationships “owns”



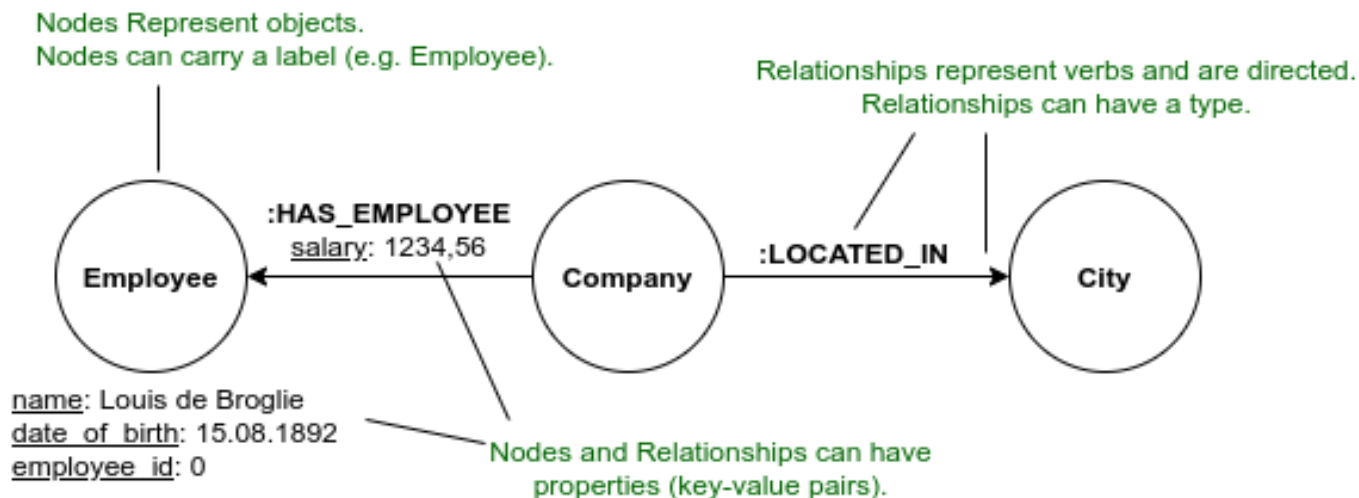
Scanning and filtering is sequential. Expand is not.

⇒ Expand causes prefetch & cache misses.

## Example III

- Especially `Expand` jumps a lot. Potentially back and forth.
- Traversals rely primarily on `expand`.

## Property Graph Model



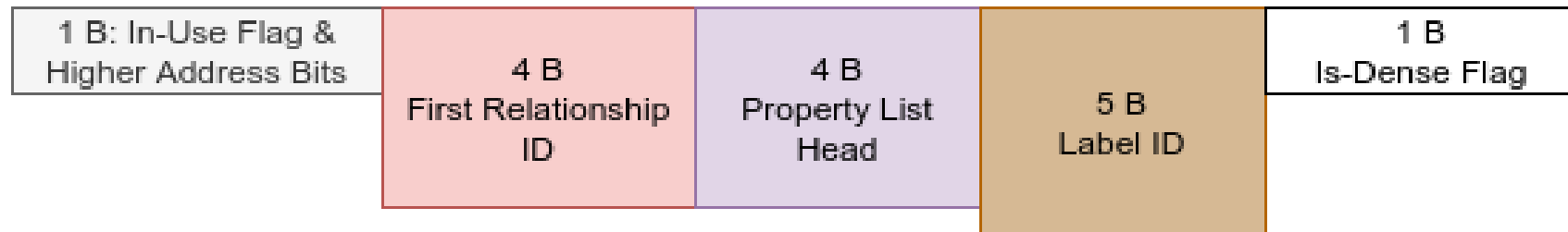
# Data Structures I

Two essential record structures:

1. Node records
2. Relationship records

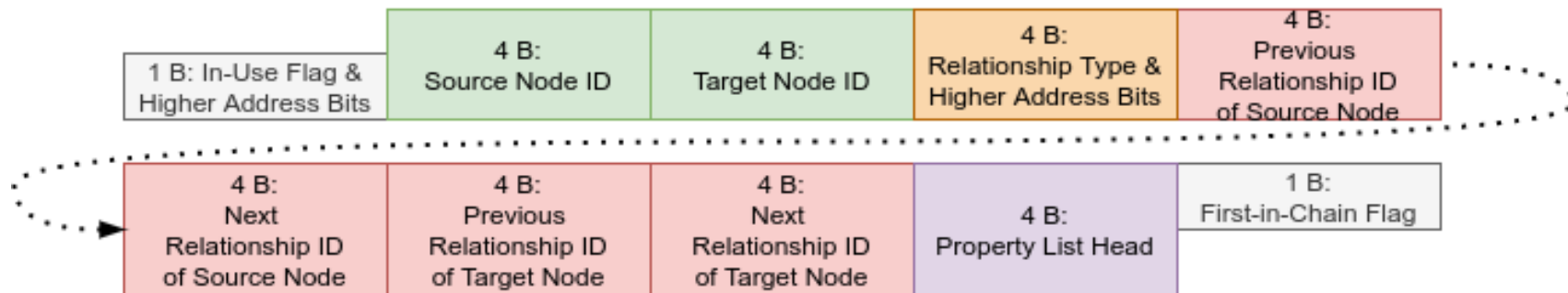
Inspired by Neo4J

## Data Structures II

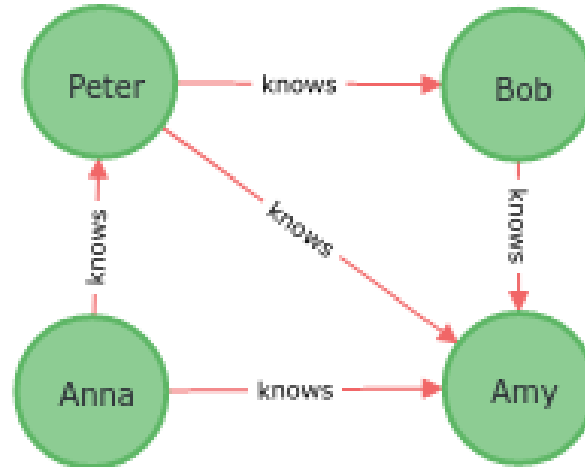




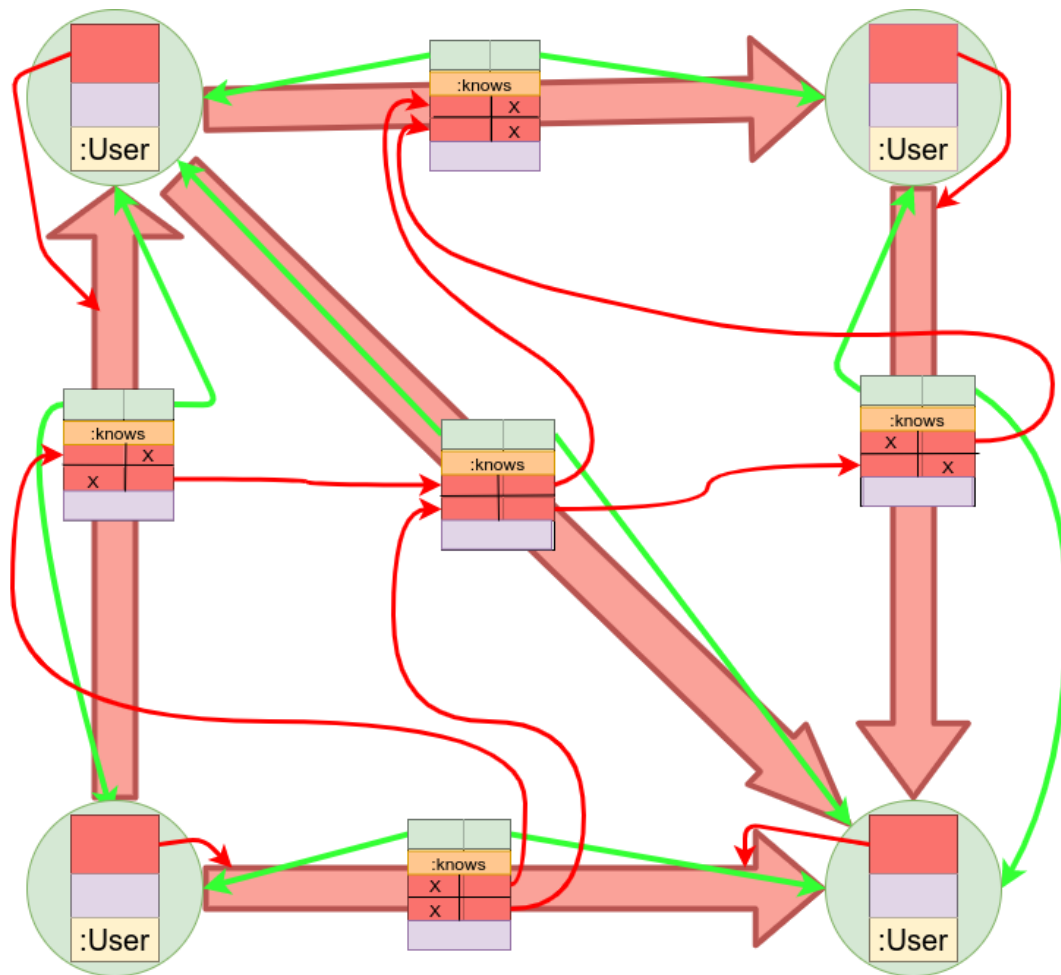
# Data Structures III



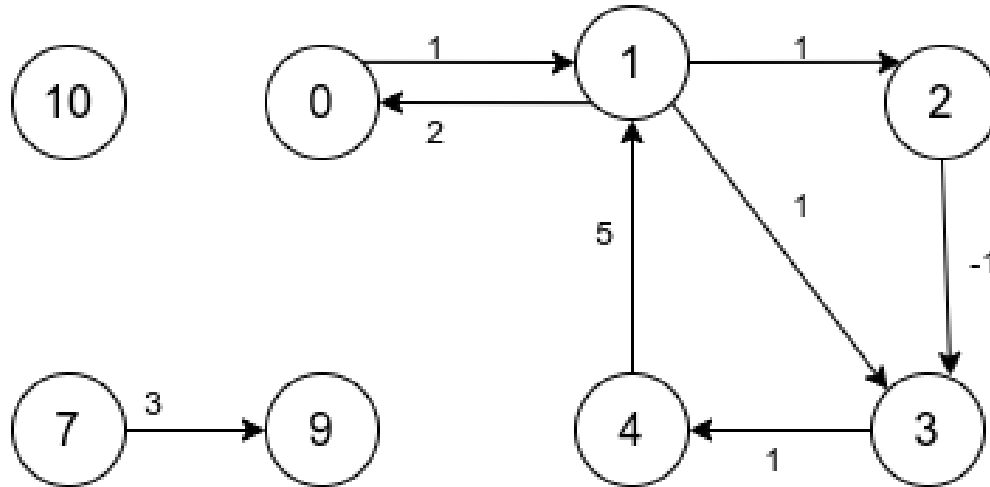
## Data Structures IV



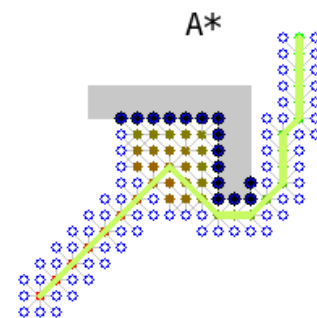
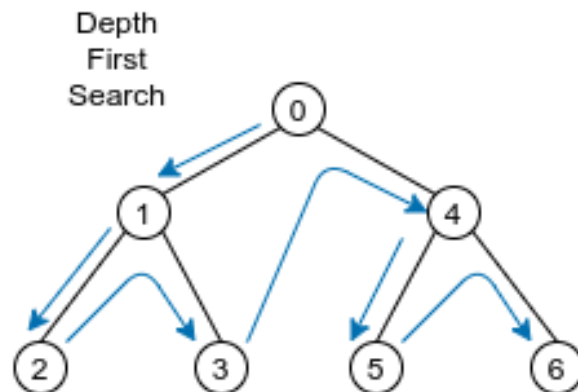
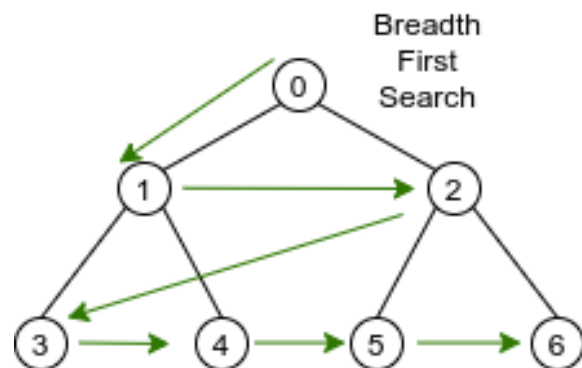
## 2 Background



### Graphs we focus on



# Traversals



## Problem Definition I

Given a graph  $G$ , logical block size  $b$ , page size  $p$ .

Desired is

1. A partition of  $G$  into blocks of vertex records  $V_i$  and  $E_i$  relationship records,
2. permutations  $\pi_v, \pi_e$  of the blocks of vertex and edge records  $V_i, E_i$ ,
3. a reordering of the incidence list pointers

such that spatial locality is as high as possible for traversal-based queries.

## Problem Definition II

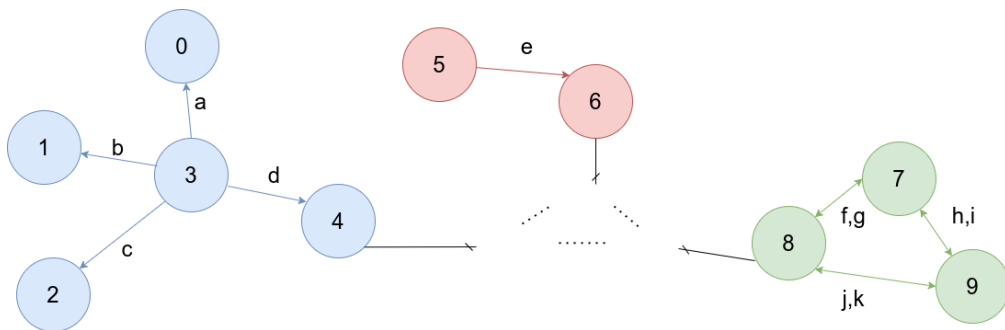
Temporal locality based on blocks.

$$P(X_{t+\Delta} = B | X_t = B)$$

Spatial locality in the same sense:

$$P(X_{t+\Delta} = B \pm \varepsilon | X_t = B)$$

# Problem Definition III

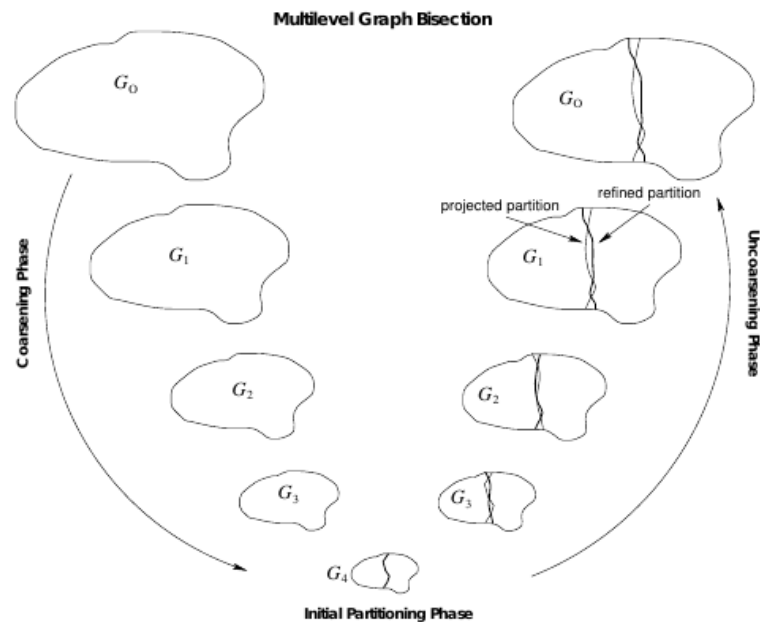


node.db	0, 5, 7	1, 4, 9	2, 6, 8	3		
edge.db	a, f	b, g	c, h	d, i	e, j	k

node.db	7, 8, 9	0, 1, 3	2, 4, 5	6		
edge.db	f, h	g, k	i, j	a, b	c, d	e



# G-Store I

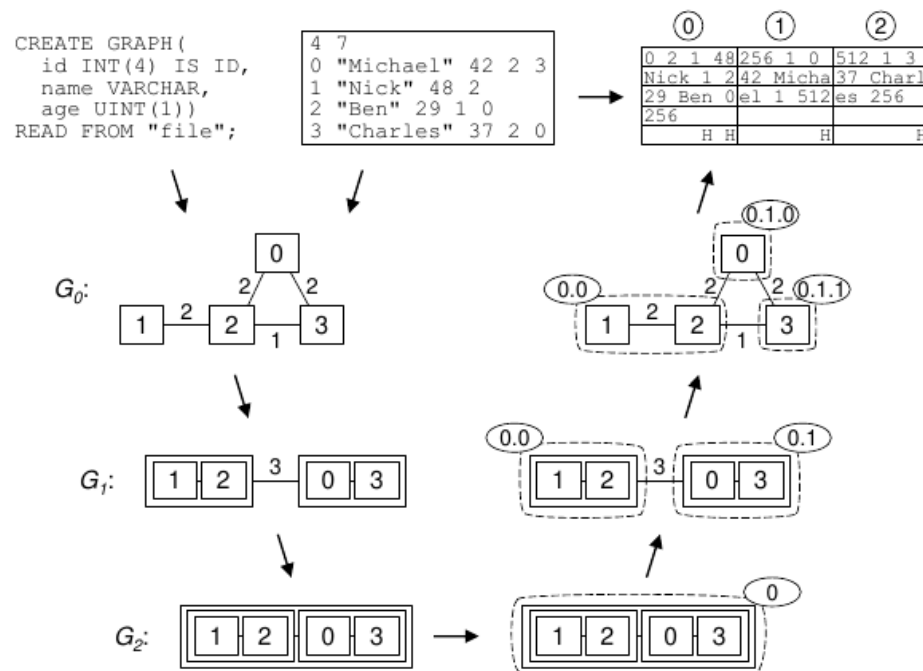


## G-Store II

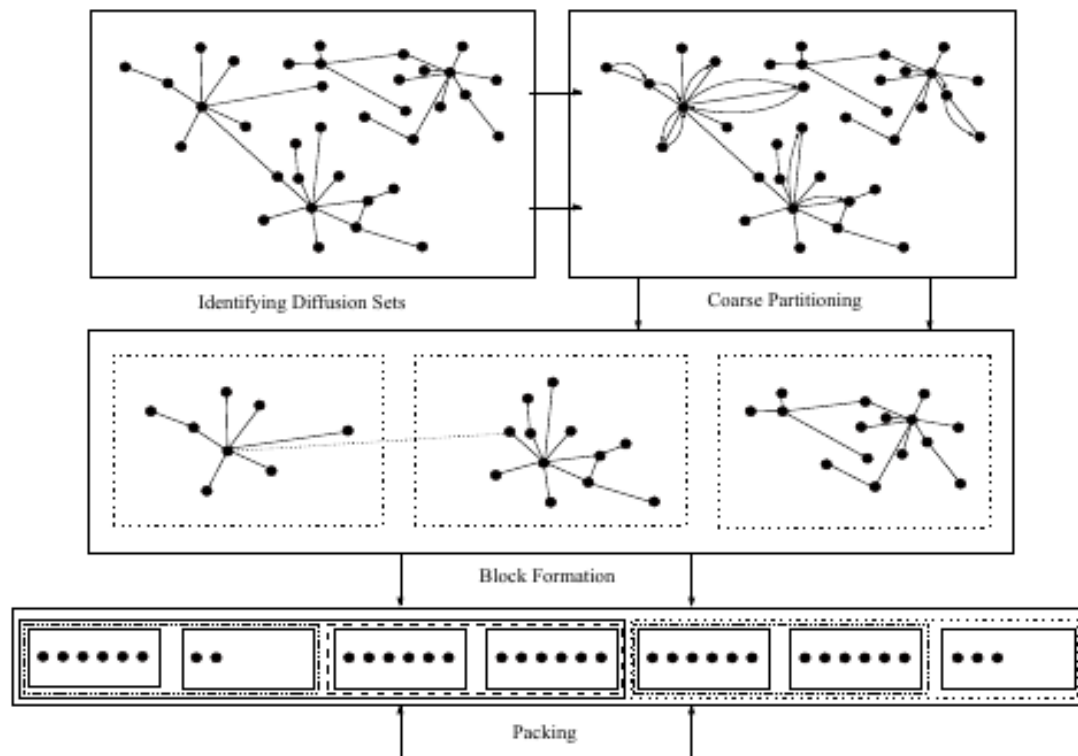
1. Coarsening: Heavy-Edge Matching
2. Turn-around
3. Uncoarsening
  - 3.1 Project
  - 3.2 Reorder
  - 3.3 Refine

$$\min \sum_{(u,v) \in E} |\phi(u) - \phi(v)|$$

# G-Store III



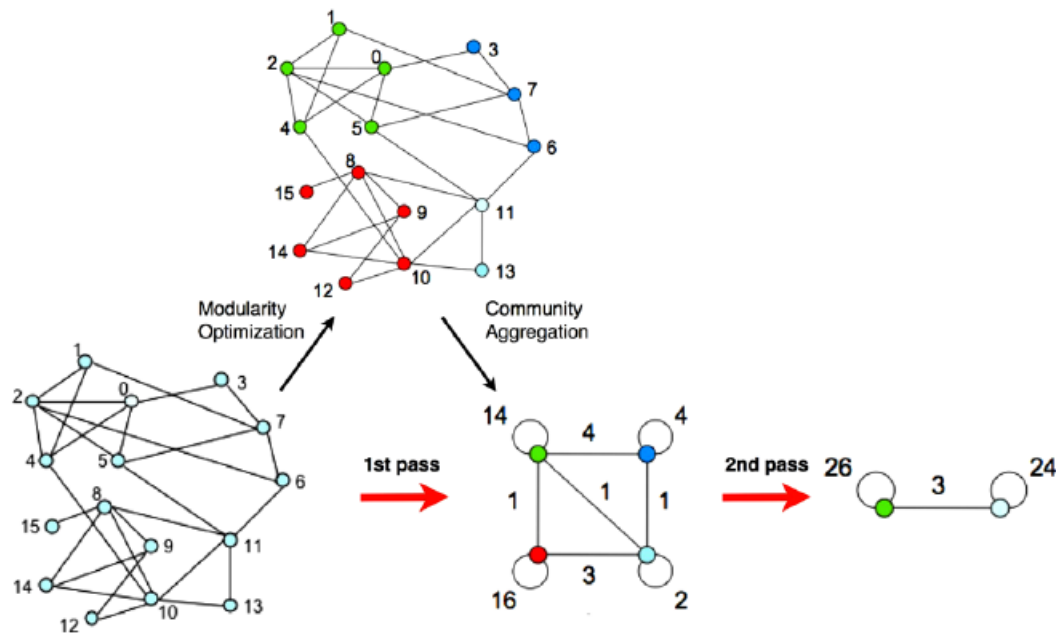
### ICBL I



# ICBL II

- I Feature extraction: Do  $t$  random walks of length  $l$ .
- C Coarse clustering: Adapted K-Means.
- B Block Formation: Agglomerative hierarchical clustering.
- L Layout Blocks: Sort blocks and subgraphs

# Louvain Method I



## Louvain Method II

1. Initialize all nodes in singleton community.
2. Merge community into a neighboring community where modularity gain is maximal, until modularity gain is below threshold.
3. Construct new graph from aggregated communities and go to 1.

$$\frac{1}{2m} \sum_{u,v \in V} \left( w_{(u,v)} - \frac{w_u w_v}{2m} \right) \cdot \delta(c_u, c_v)$$

# Incidence List Rearrangement

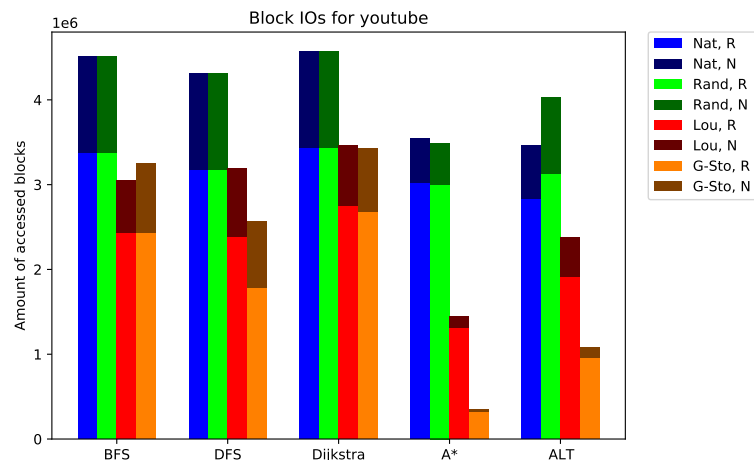
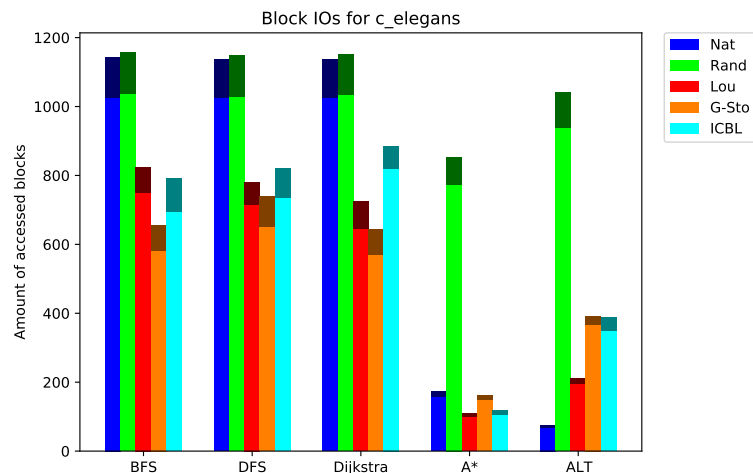
TODO figure



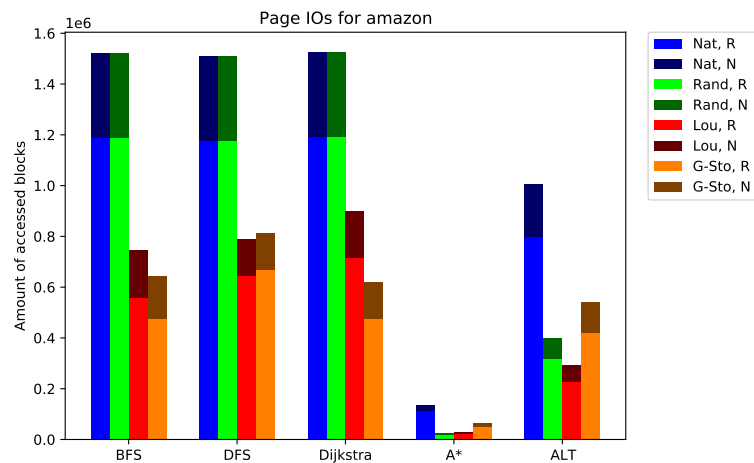
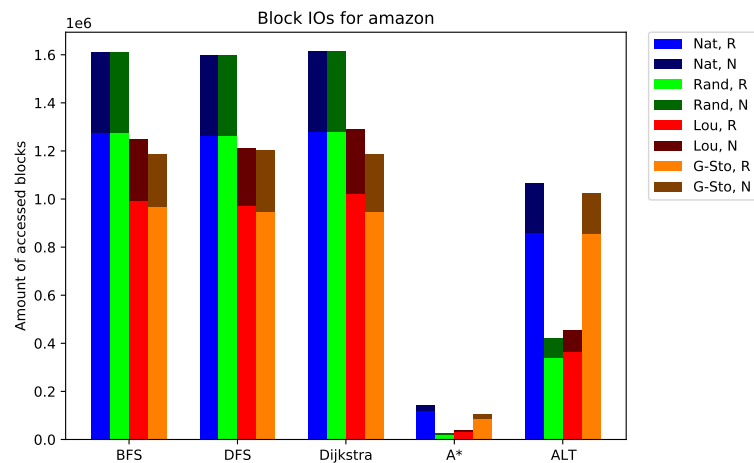
# Setup

- Queries: BFS, DFS, Dijkstra,  $A^*$ , ALT.
- Datasets: [131, 1'134'890] nodes, [764, 2'987'624] edges, average degree [2.6, 25.5]
- Domains include biological neural net, E-Mails, Co-authors, Frequent item sets, Comments.

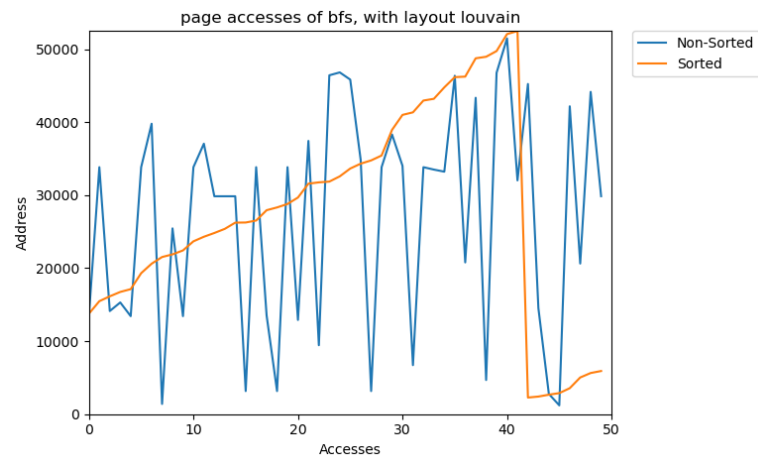
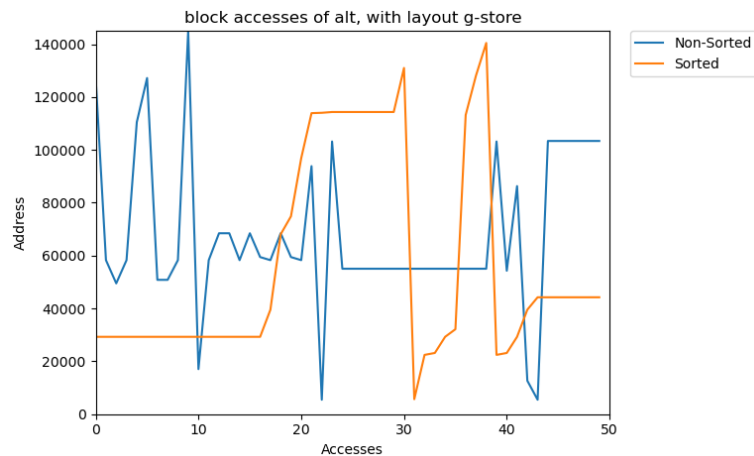
# Results I



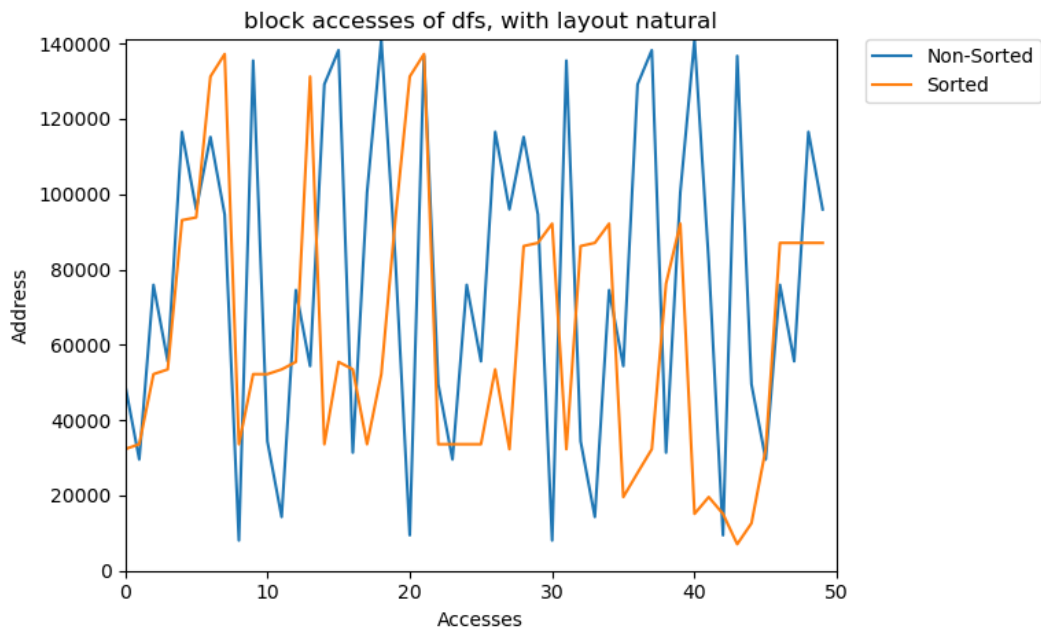
# Results II



# Results III



## Results IV



# Summary

- Static rearrangement methods decrease number of block accesses.  
⇒ increase locality
- Sorting the incidence lists leads to more sequential access sequences.
- Ordering the blocks is crucial for spatial locality.

# Future Work

- Leiden instead of Louvain
- RCM-based rearrangement
- Dynamic Rearrangement — Query-based