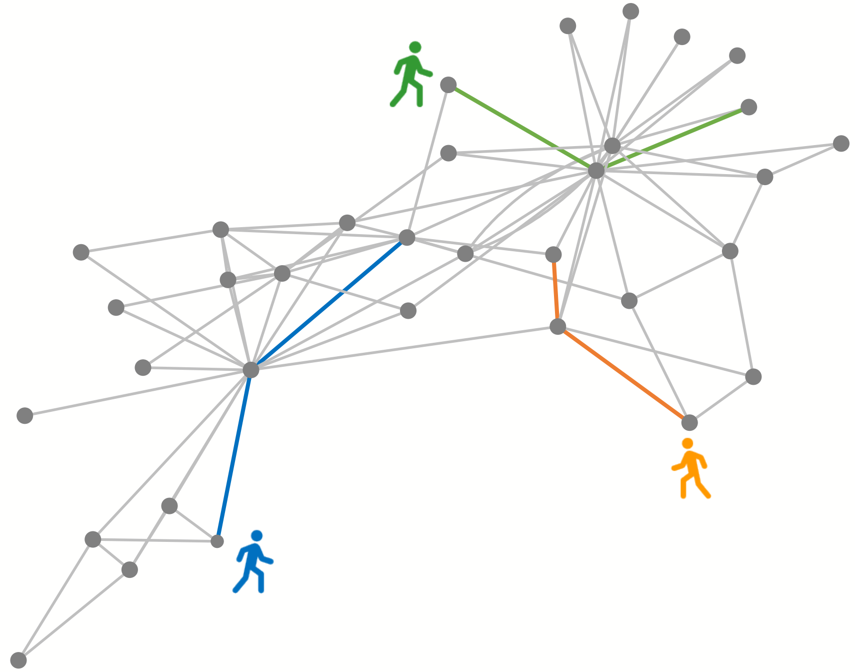
SOWalker: An I/O-Optimized Out-of-Core Graph Processing System for Second-Order Random Walks

Yutong Wu, Zhan Shi, Shicai Huang, Zhipeng Tian, Pengwei Zuo,
Peng Fang, Fang Wang, Dan Feng
Huazhong University of Science and Technology

USENIX ATC 2023

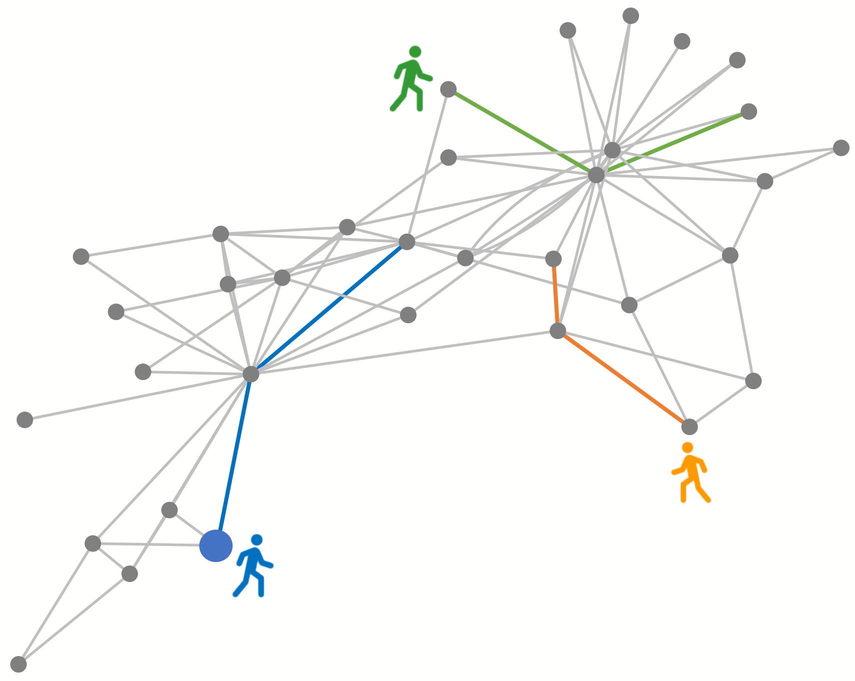
Random Walk in Graph

- Random walk, a powerful tool for extracting information from graphs
  a) Each walker starts from a given vertex
  b) Randomly moves to a neighbor of the current vertex
  c) Repeat b) until a termination condition is satisfied



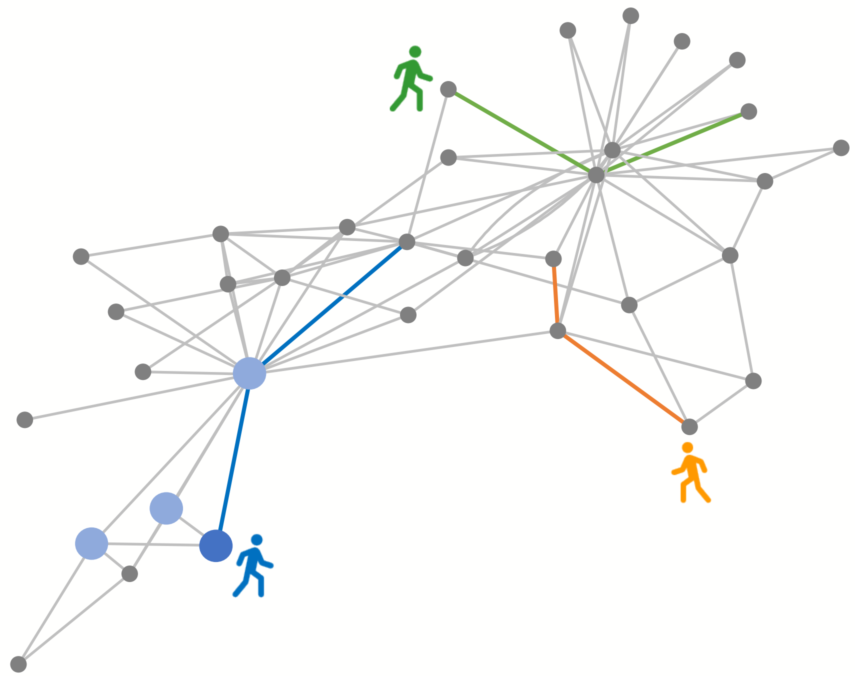
Random Walk in Graph

- Random walk, a powerful tool for extracting information from graphs
  a) Each walker starts from a given vertex
  b) Randomly moves to a neighbor of the current vertex
  c) Repeat b) until a termination condition is satisfied
- First-order random walk
  - Only depends on the current vertex
  - E.g., DeepWalk, Personalized PageRank, SimRank …



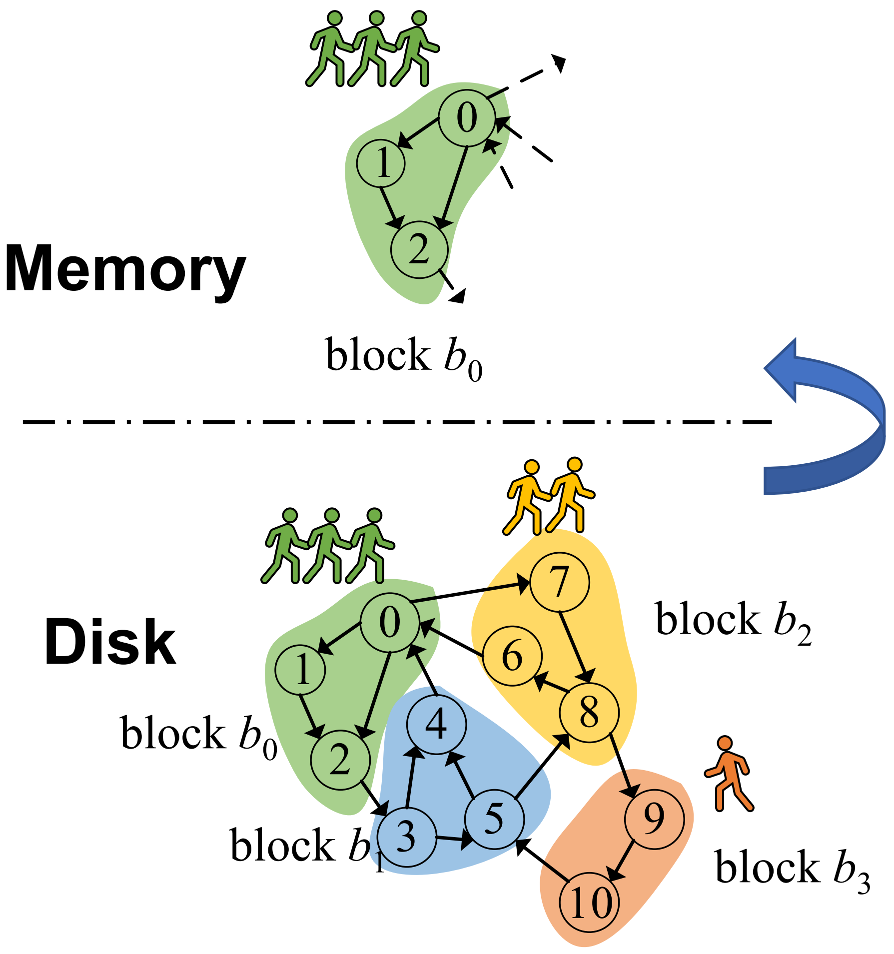
Random Walk in Graph

- Random walk, a powerful tool for extracting information from graphs
  a) Each walker starts from a given vertex
  b) Randomly moves to a neighbor of the current vertex
  c) Repeat b) until a termination condition is satisfied



- First-order random walk
  - Only depends on the current vertex
  - E.g., DeepWalk, Personalized PageRank, SimRank …
- Second-order random walk
  - Consider the previous vertex
  - Facilitate to model higher-order structures
  - E.g., node2vec, second-order PageRank, second-order SimRank …

General Out-of-core Graph Processing Framework



First-order random walk

The current vertex is in the loaded block, the first-order random walk can be immediately updated without I/Os

Second-order random walk

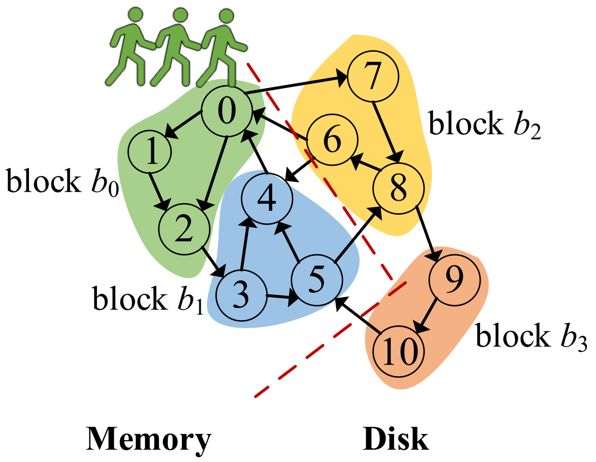
The previous vertex might belong to other blocks on disks, leading to extra I/Os

How to avoid loading non-updatable walks?

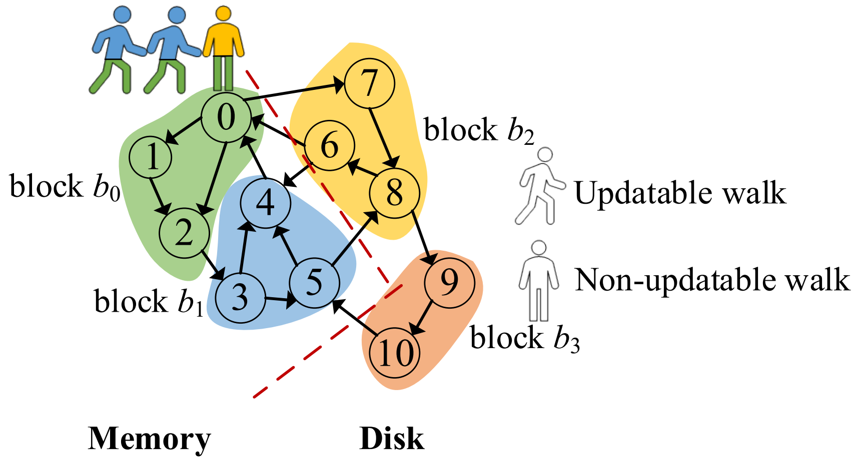
How to load as many updatable walks as possible?

Limitation of Current Solution

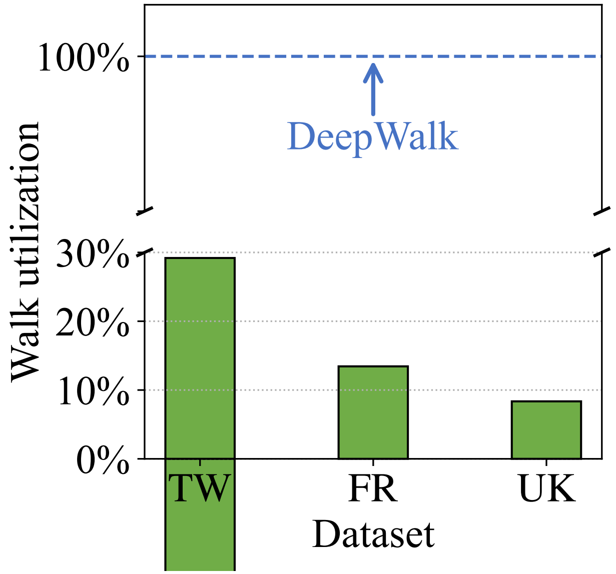
- Load all walks on the current block, but not all second-order walks are updatable



(a) First-order random walk



(b) Second-order random walk



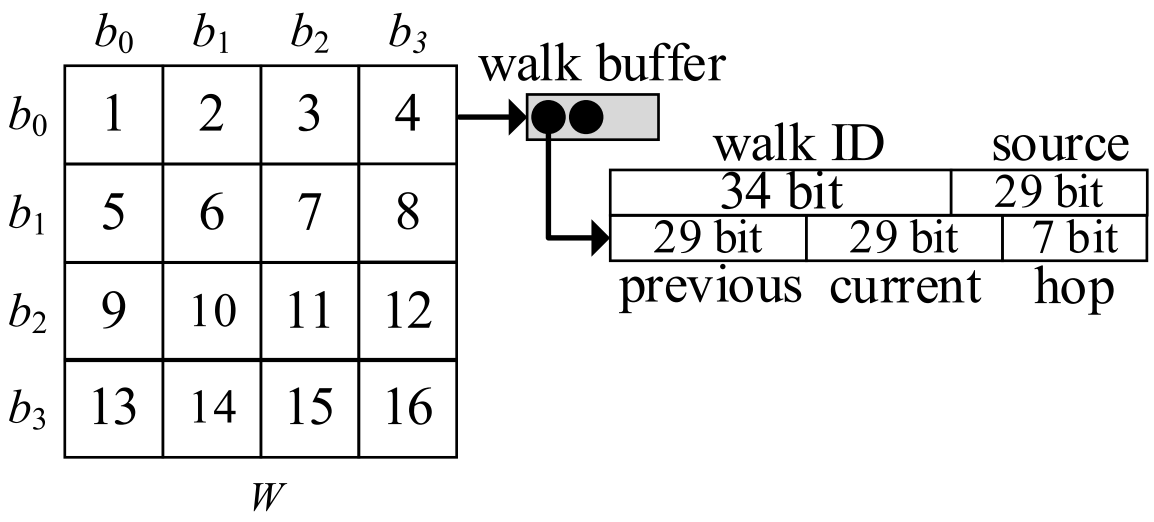
All three first-order random walks are updatable as opposed to only two second-order random walks

$$Walk\ utilization = \frac{\#updatable\ walks}{\#loaded\ walks}$$

Non-updatable walks result in useless walk I/Os

Our solution – Walk Matrix

- $|W| = |B|$
- W_{ij} stores the walks whose previous vertex belongs to block i, and the current vertex belongs to block j
- Check whether a walk can be updated, judging that both the previous and current vertices are in memory
- Encode each walk with 128 bits

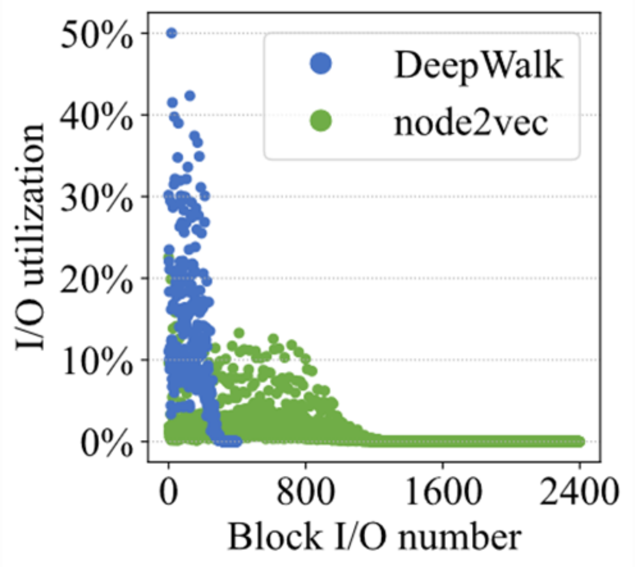


Skip loading non-updatable walks and eliminate useless walk I/Os

Limitation of Current Solution

- Iteratively loads ancillary blocks, unaware of updatable walk states

- Run DeepWalk (i.e., first-order) and node2vec (i.e., second-order) on GraphWalker



$$I/O\ utilization = \frac{\#walk\ steps}{\#total\ edges\ in\ a\ loaed\ block}$$

The non-optimal block scheduling model results in low I/O utilization

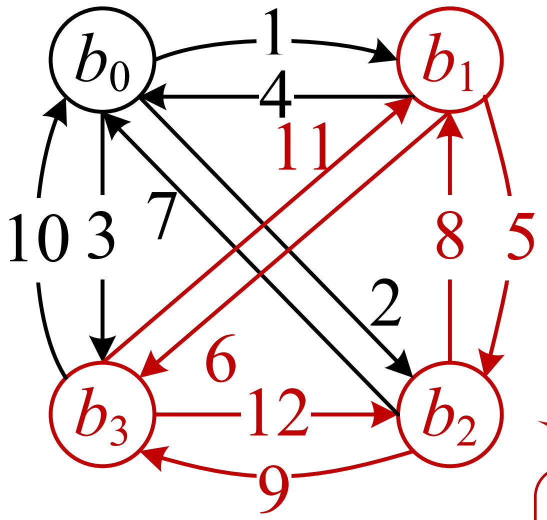
Our solution – Benefit-aware I/O Model

- Load multiple blocks with the maximum accumulated updatable walks

	b_0	b_1	b_2	b_3
b_0	0	1	2	3
b_1	4	0	5	6
b_2	7	8	0	9
b_3	10	11	12	0

W

map



CDG

Maximum edge weight clique

Maximize the I/O utilization in a block I/O

Our solution – Benefit-aware I/O Model

Linear programming method

$$\max \sum_i \sum_j e_{ij} y_{ij}$$

$$\text{s.t. } \sum_{i=0}^{N_B-1} x_i = m$$

$$\sum_{i=0}^{N_B-1} \beta_i x_i = m - k$$

$$y_{ij} \le x_i$$

$$y_{ij} \le x_j$$

$$x_i, y_{ij} \in \{0,1\}$$

Exact but time-consuming

Simulated annealing method

Algorithm 1: SA-based benefit-aware I/O model

Function SelectBlocks (CDG=(B,E), B_0):

- $B_L \leftarrow B_0$ // initial block set
- $t \leftarrow T_0$ // initial temperature
- $i \leftarrow 0$ // iteration counter
- **while** $t \ge T_s$ and $i \le iter_{max}$ **do**
  - $B_c \leftarrow$ CHOOSENEWBLOCK (CDG, B_L)
  - $\Delta S = S(B_c) - S(B_L)$
  - **if** $\Delta S > 0$ **or** $e^{\Delta S/t} >$ random(0, 1) **then**
    - $B_L \leftarrow B_c$
  - $t \leftarrow \gamma t$
  - $i \leftarrow i + 1$
- **return** B_L

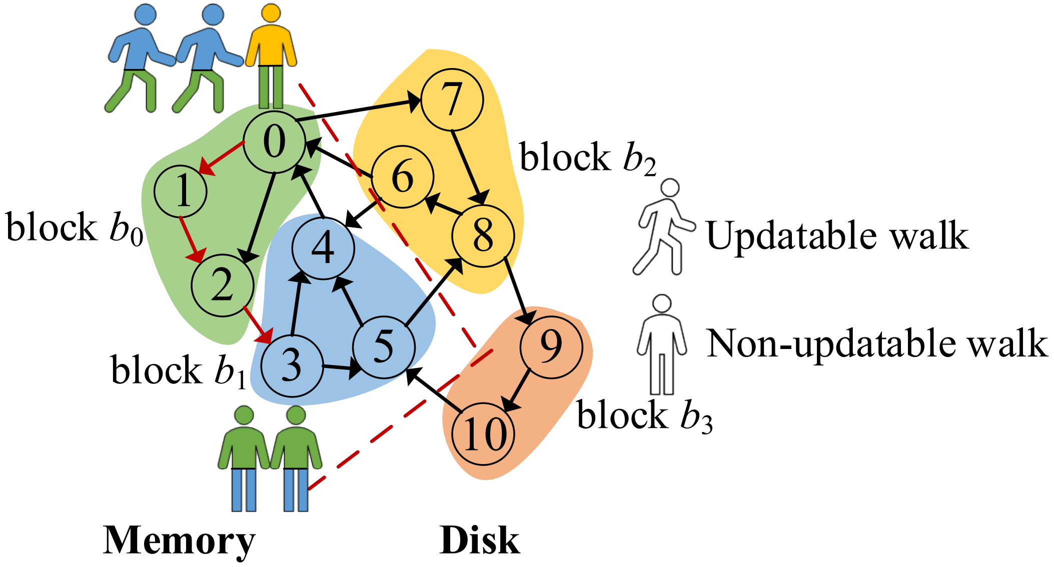
Top-m blocks based on the number of walks in a block

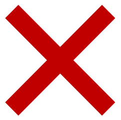
Randomly swap a block between the current and remaining block set

Efficient and effective

Limitation of Current Solution

- Manage walks at a block granularity and restrict walk updating to a block



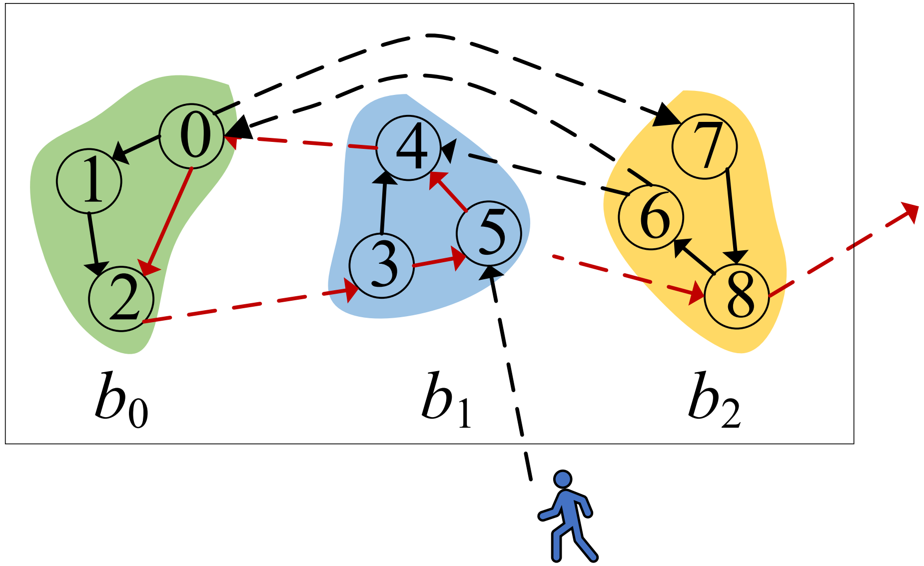


Fail to utilize the vertex information in other blocks residing in memory

If the previous and current vertex information are both available, the walk can further be updated

The block-oriented walk updating scheme brings low walk updating rate

Our Solution – Block Set-Oriented Walk Updating



- Walks can move across blocks via the cut edges between these blocks
- Each walk can be updated as much as possible in the loaded block set

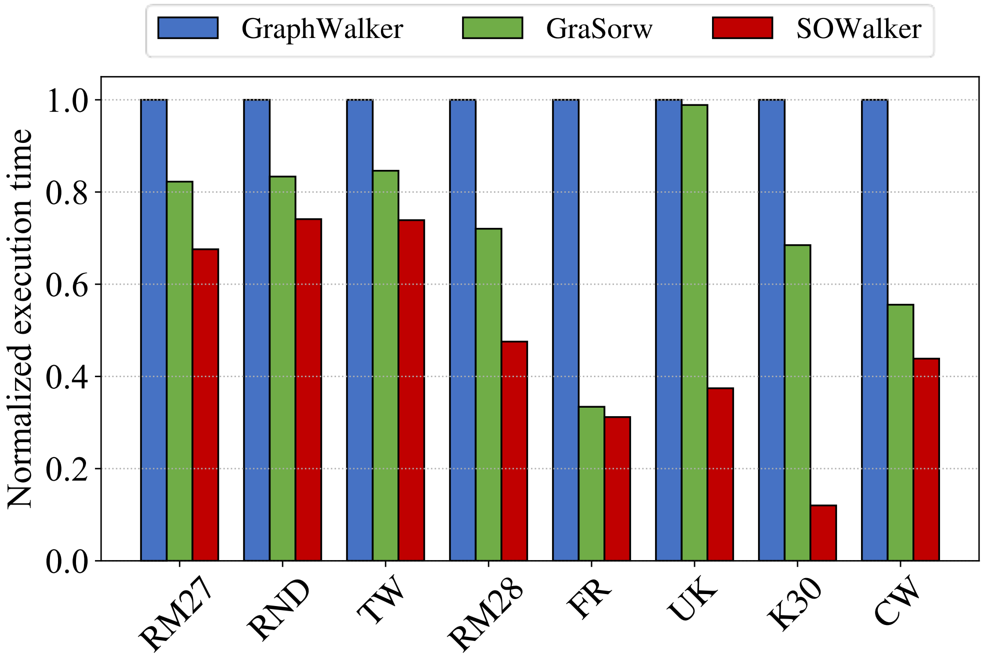
Maximize the walk updating rate of the loaded block set

Evaluation

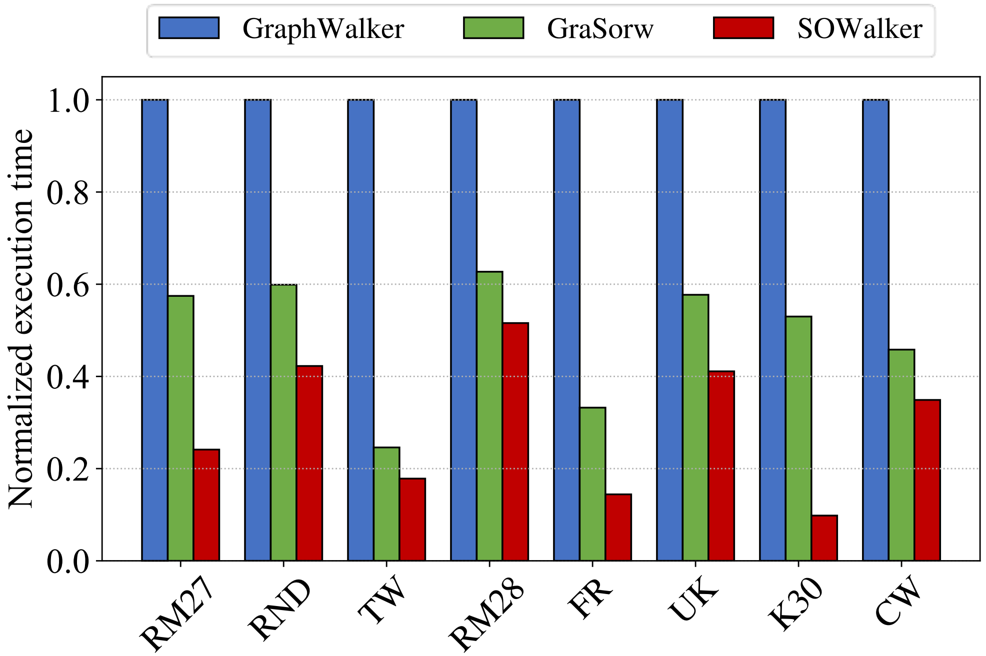
- Environment
  - 32-core 2.10GHz Intel Xeon CPU E5-2620
  - 128GB main memory and 3TB HDD
- Applications
  - Node2vec, second-order PageRank
- Comparison systems
  - GraphWalker [ATC'20], GraSorw [VLDB'22]
- Datasets

Dataset	$\|V\|$	$\|E\|$	Graph Size	CSR Size	Block Size	$\|B\|$
RM27	134.2M	1.1B	18GB	4GB	512MB	9
RND	268.4M	1.4B	24.7GB	5.2GB	512MB	11
TW	61.5M	1.5B	24.4GB	5.5GB	1GB	6
RM28	268.4M	2.1B	34.9GB	8GB	1GB	9
FR	65.6M	3.6B	58GB	13.5GB	1GB	14
UK	133.6M	5.5B	94.6GB	20.4GB	1GB	21
K30	1.1B	33.8B	628.3GB	120GB	8GB	16
CW	3.6B	126B	2.6TB	470GB	8GB	59

Overall Performance



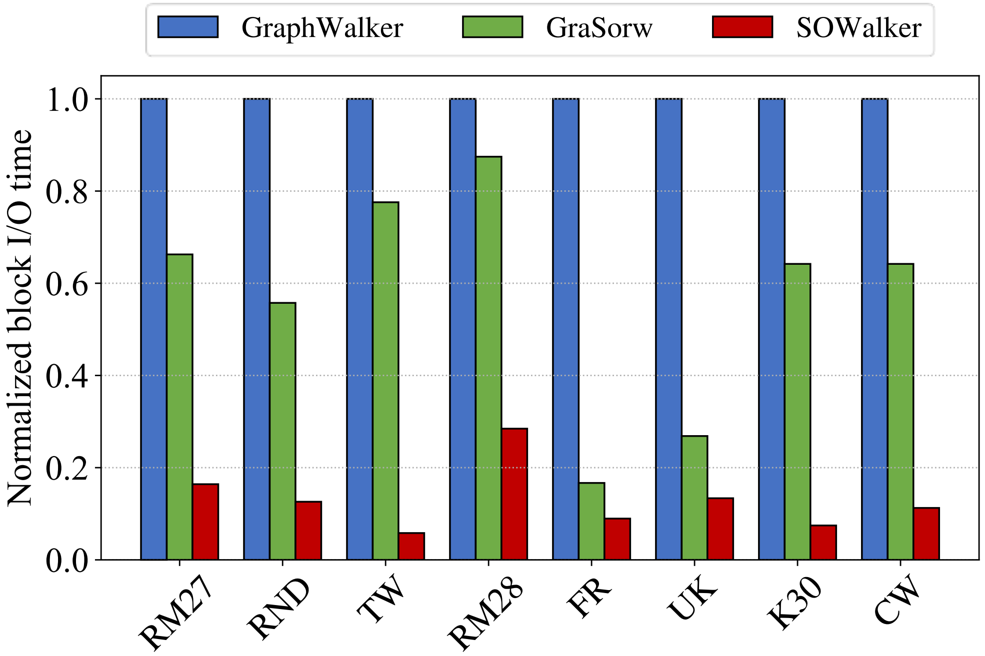
node2vec



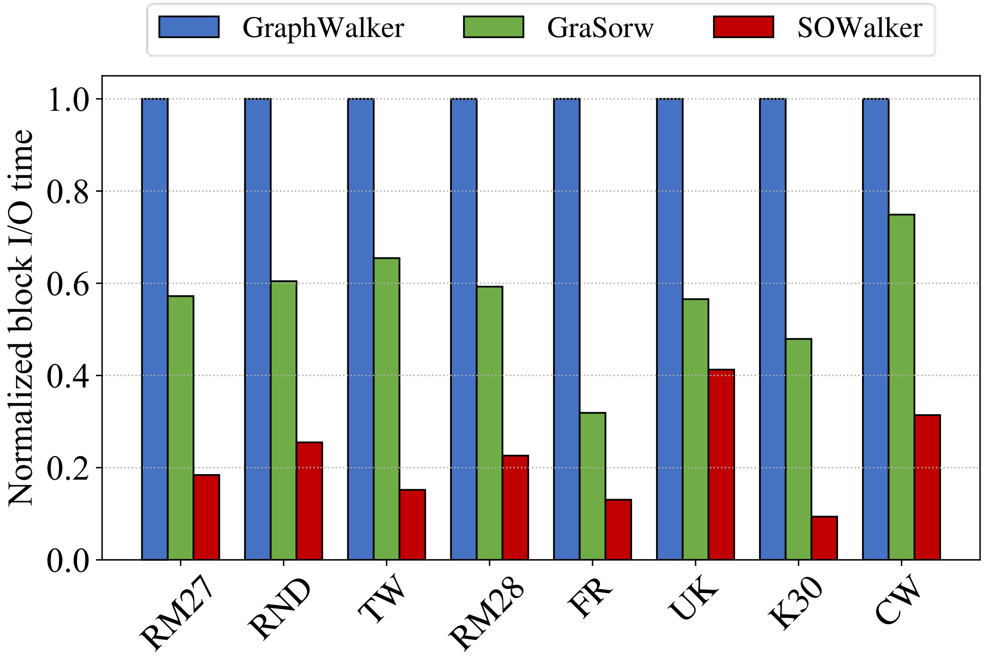
Second-order PageRank

SOWalker achieves 1.4-10.2× speedups over GraphWalker.
SOWalker achieves 1.2-5.7× speedups over GraSorw.

I/O-efficiency Evaluation



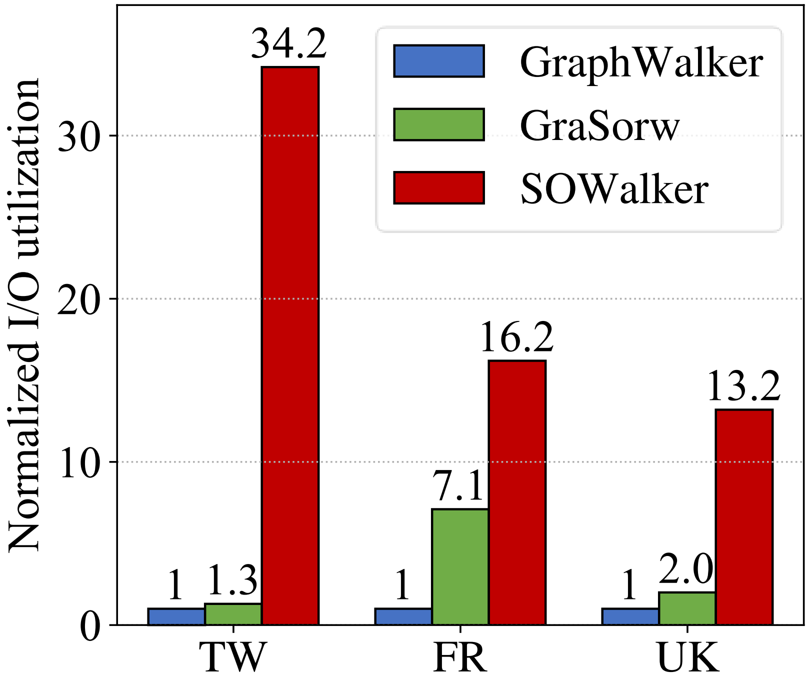
node2vec

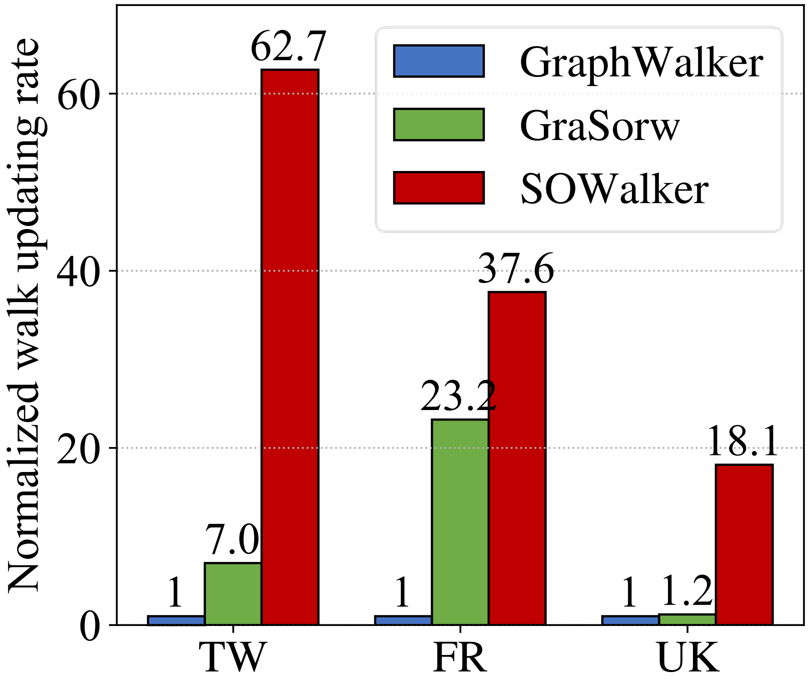


Second-order PageRank

The block I/O time in SOWalker is only 5.8-41.3% of that in GraphWalker, and 7.5-72.9% of that in GraSorw, respectively.

I/O-efficiency Evaluation





I/O utilization of SOWalker is improved by 13.2-34.2× and 2.3-26.4× compared to GraphWalker and GraSorw, respectively.

Design Choices

➢ Scheduling Models

- Random: randomly chooses m blocks to load into memory
- Max-*m*: chooses top-*m* blocks based on the number of walks in a block
- Exact: the exact benefit-aware I/O model according to the linear programming method
- BA: the benefit-aware I/O model according to the simulated annealing method

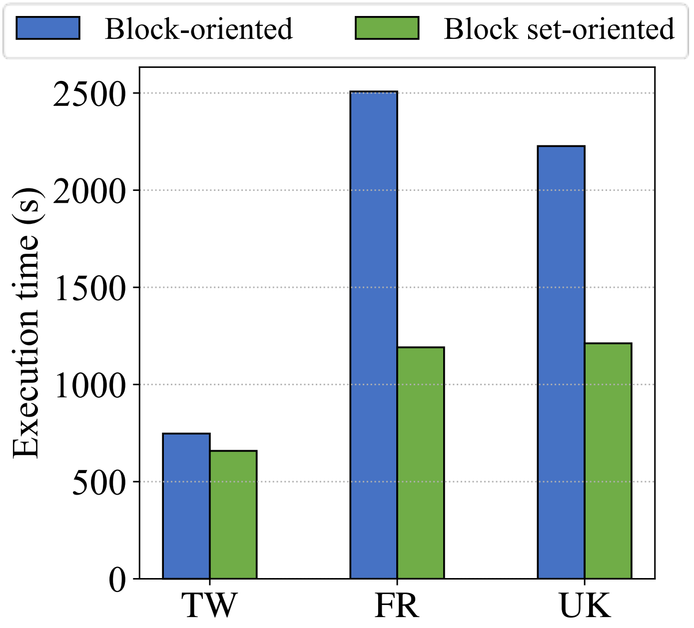
Model	Execution time (s)	Block I/O time (s)	Block I/O number	Computation time (s)
Random	4970	3234	9868	-
Max-*m*	3871	2162	6391	-
Exact	14311	548	1484	12097
BA	2133	575	1537	10

Benefit-aware I/O model (BA) can achieve better I/O performance and faster runtime.

Design choices

➢ Walk Updating Schemes

- I/O model: loads a block with the maximum number of walks as the current block and iteratively loads another block into memory as the ancillary block



Yield up to 2.1× speedups under the block set-oriented scheme.

Conclusion

- SOWalker: An I/O-Optimized Out-of-Core Graph Processing System for Second-Order Random Walks
  - Walk matrix
    - Avoid loading non-updatable walks
  - Benefit-aware I/O model
    - Load multiple blocks with the maximum accumulated updatable
  - Block set-oriented walk updating scheme
    - Allow each walk to move as many steps as possible in the loaded block set
- Result
  - Achieve up to 10.2× speedups compared to two state-of-the-art out-of-core random walk systems

Thanks for your attention!

Email: yutongwu@hust.edu.cn

Open-source code: https://github.com/Teamb507/SOWalker.git