VLDB2023
vancouver

MiniGraph: Querying Big Graphs with a Single Machine

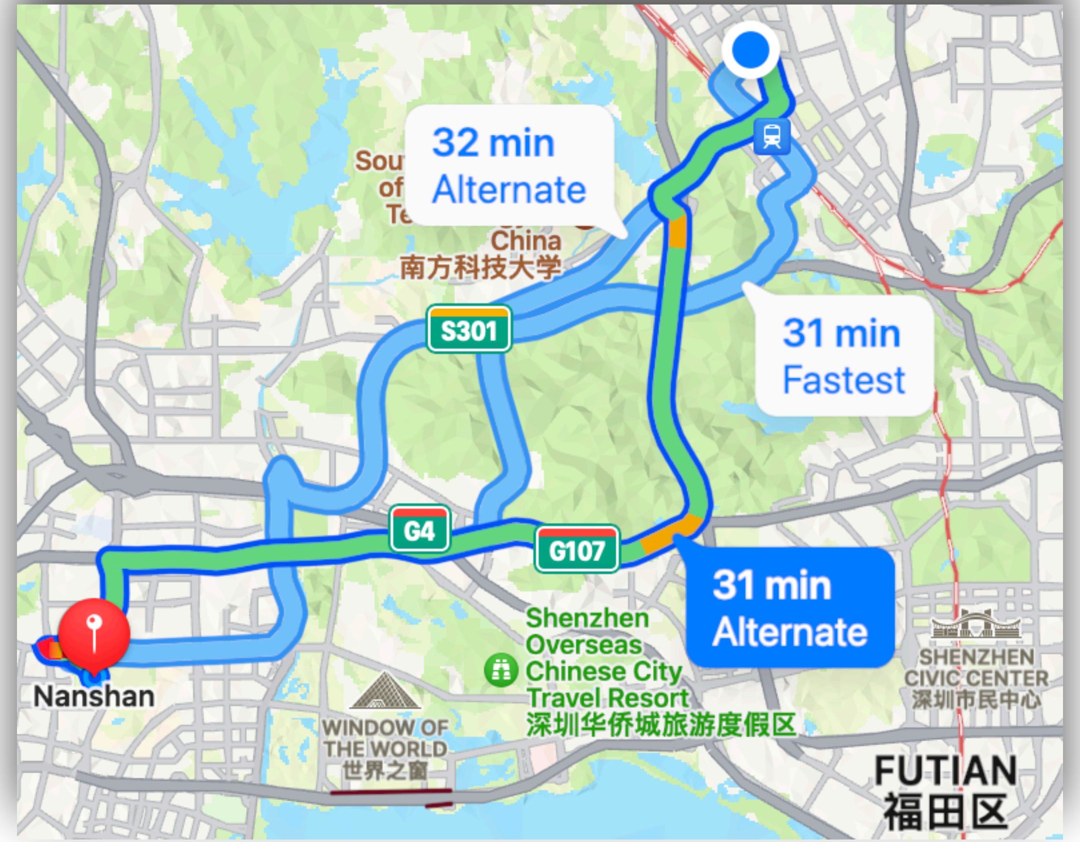
Xiaoke Zhu, Yang Liu, Shuhao Liu, and Wenfei Fan

北京航空航天大学 BEIHANG UNIVERSITY 1952

深圳计算科学研究院
Shenzhen Institute of Computing Sciences

THE UNIVERSITY OF EDINBURGH

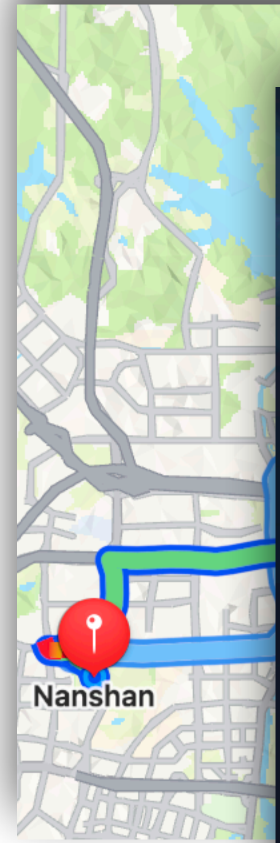
Graphs: Everything is Naturally Connected



Road networks

We use Graphs everyday and everywhere.

Graphs: Everything is Naturally Connected

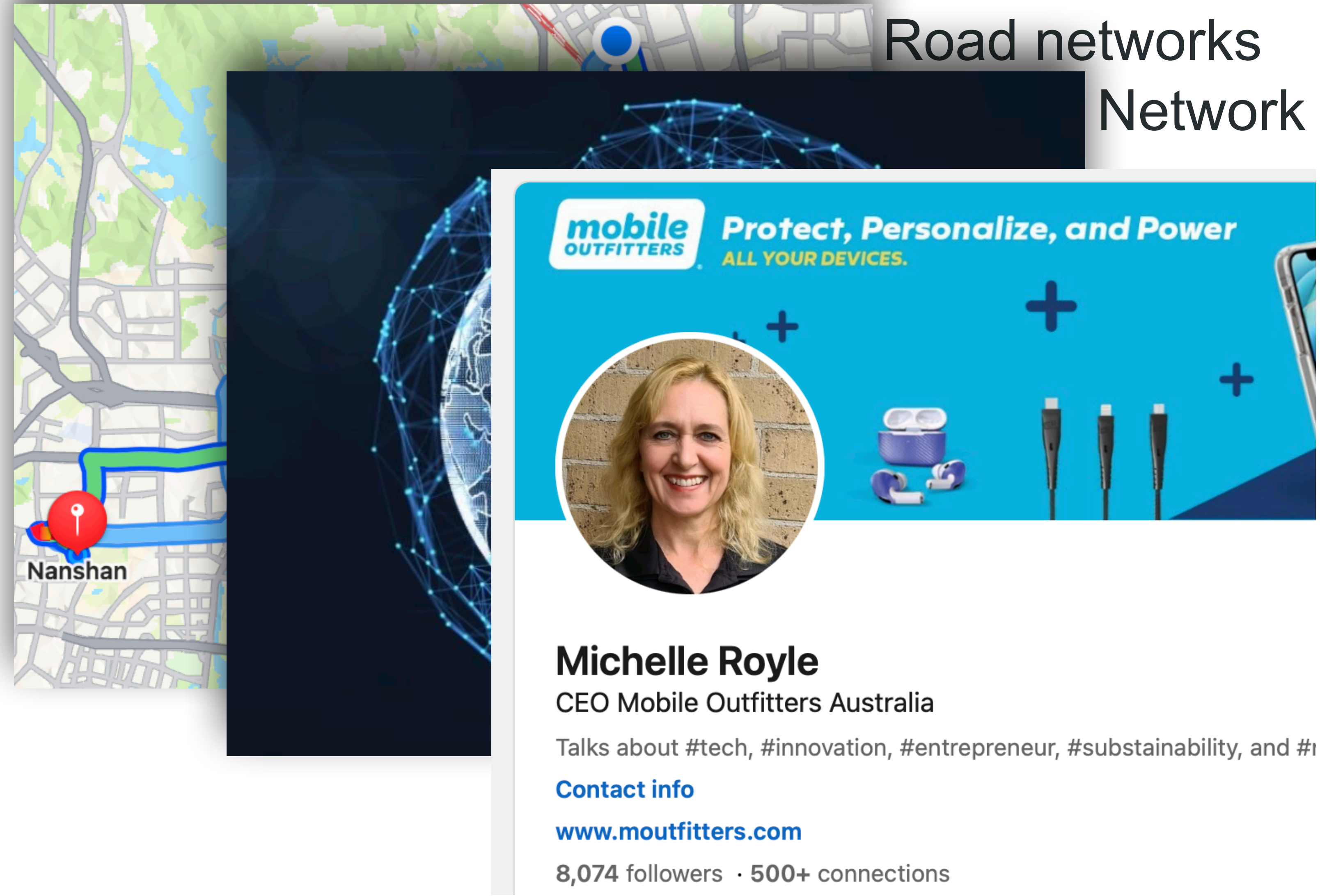


Road networks

Network topology

We use Graphs everyday and everywhere.

Graphs: Everything is Naturally Connected



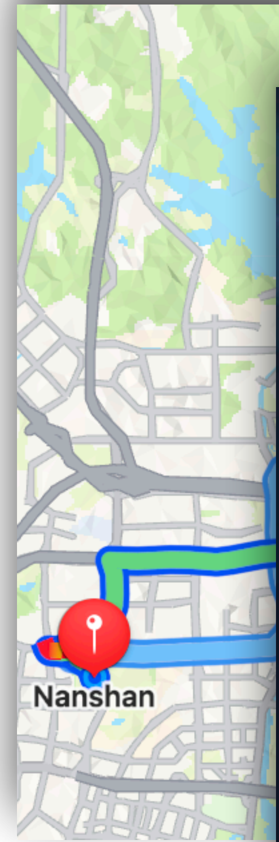
Road networks

Network topology

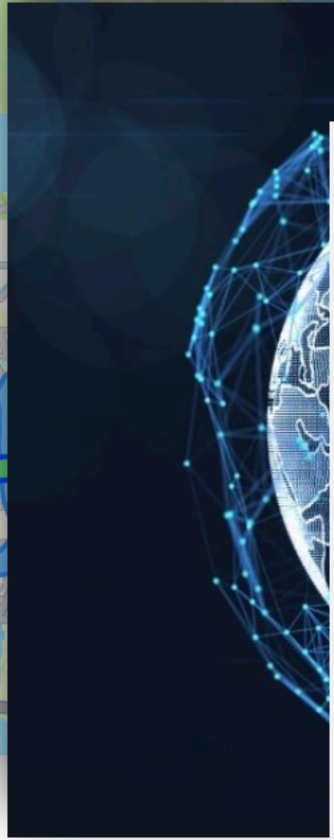
Social networks

We use Graphs everyday and everywhere.

Graphs: Everything is Naturally Connected



Road networks



Network topology

Social networks

KG

We use Graphs everyday and everywhere.

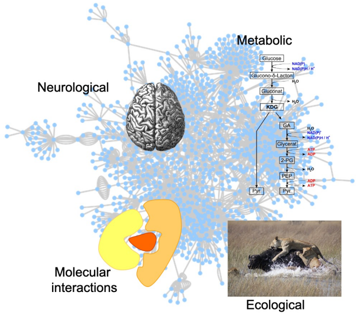
Applications on Graph Data

Map Navigation | Fraud Detection | Recommendation | Protein Interaction

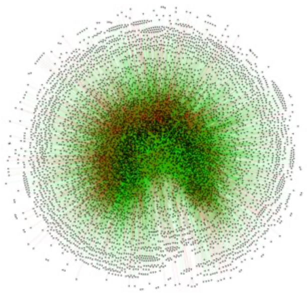
SSSP | *PageRank* | *SubIso* | *WCC* | *Simulation*

Road Network

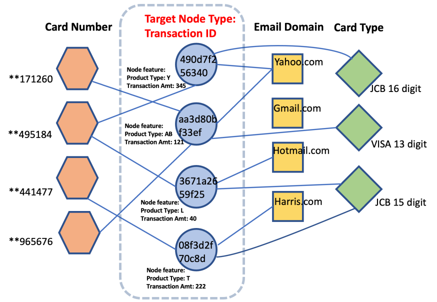
Social Media



Biological Network



Web Graph



Transaction Network

What is graph computing and why is it important?

Existing Solutions

Shared memory

- ✓ Single-node and in-memory
- ✓ Ligra[PPoPP'13],Galois[SOSP'13]

Limited capacity to big graphs

Existing Solutions

Shared memory ✗

- ✓ Single-node and in-memory
- ✓ Ligra[PPoPP'13], Galois[SOSP'13]

Limited capacity to big graphs

Distributed

- ✓ Multi-node and in-memory
- ✓ GraphScope[VLDB'21, SIGMOD'17], Pregel[SIGMOD'10], Gluon[PLDI'18]

Irregular structure, scalability problem
Beyond the reach of small companies

Existing Solutions

Shared memory ✗

- ✓ Single-node and in-memory
- ✓ Ligra[PPoPP'13], Galois[SOSP'13]

Limited capacity to big graphs

Distributed ✗

- ✓ Multi-node and in-memory
- ✓ GraphScope[VLDB'21, SIGMOD'17], Pregel[SIGMOD'10], Gluon[PLDI'18]

Irregular structure, scalability problem

Beyond the reach of small companies

> To compute connected components of a graph with billions of vertices and trillions of edges, Yahoo! employs a 1000-node cluster with 12000 processors and 128 TB of aggregated memory.

Existing Solutions

Shared memory ✗

- ✓ Single-node and in-memory
- ✓ Ligra[PPoPP'13],Galois[SOSP'13]

Limited capacity to big graphs

Distributed ✗

- ✓ Multi-node and in-memory
- ✓ GraphScope[VLDB'21, SIGMOD'17], Pregel[SIGMOD'10],Gluon[PLDI'18],

Irregular structure, scalability problem
Beyond the reach of small companies

Out-of-core

The scope of this work

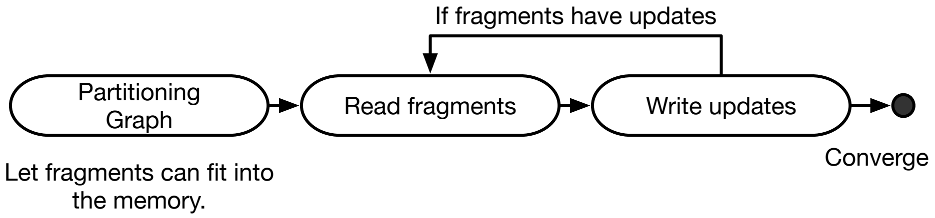
- ✓ Single-node and disk-based
- ✓ GraphChi[OSDI'12], GridGraph[ATC'15], Mosaic[EuroSys'17]

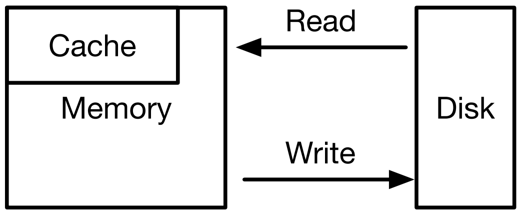
It is feasible due to promise performance of SSD, NVMe el al.
I/O will become the bottleneck

How to improve the performance of out-of-core system?

A review of out-of-core system

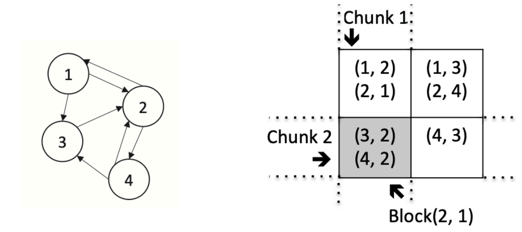
Basic idea





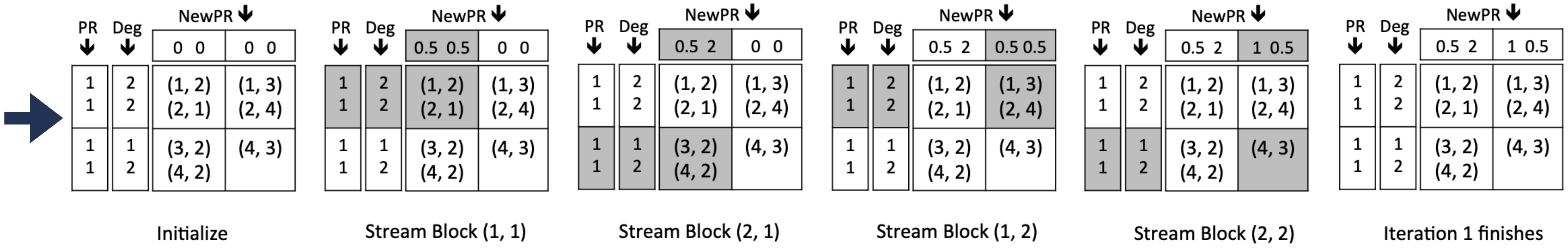
The-state-of-art: GridGraph

- ✓ Vertex-centric model and BSP model.
- ✓ ***Read*** from <u>source</u> vertices, ***Write*** to <u>destination</u> vertices.
- ✓ 2-level hierarchy partitioning and skip block with no active edges.



(a) An example graph (b) Grid representation

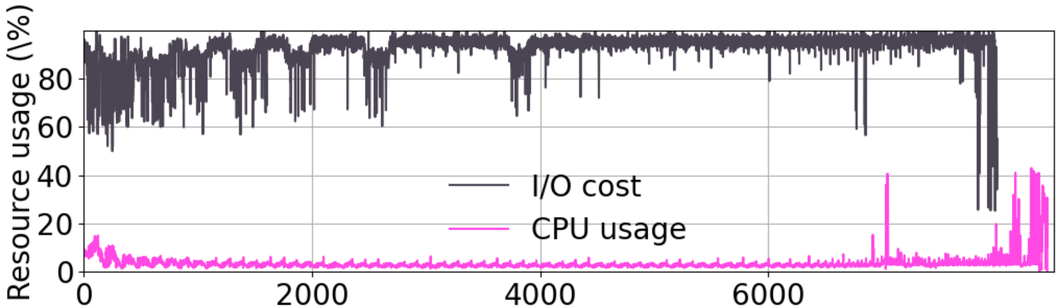
2D-partitioning

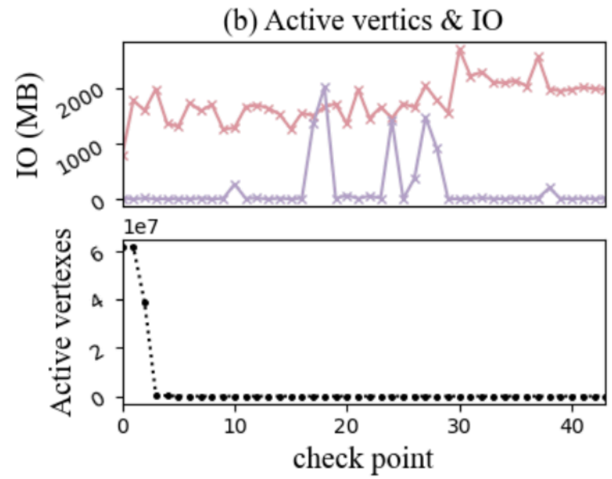


Workflow

Findings after Profiling GridGraph

- ***Setting:***
- WCC task.
- A machine powered with 20 cores and SSD.
- A graph with over 50 Millions edges (50% data out of memory).



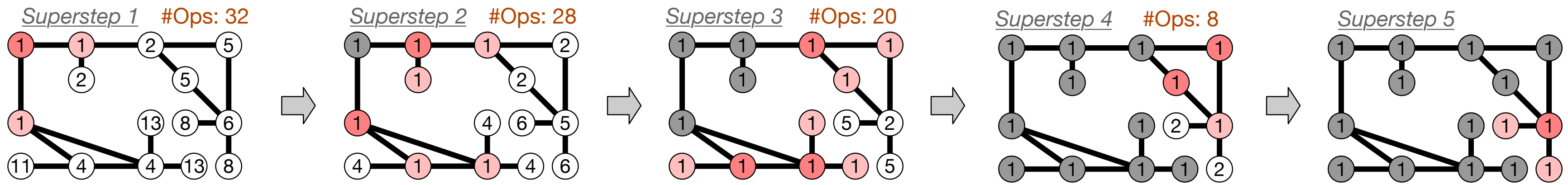


- ***Findings:***
- The rate at which a task is limited by the speed of the I/O.
- Unnecessary I/O caused by less and scattered active vertices.

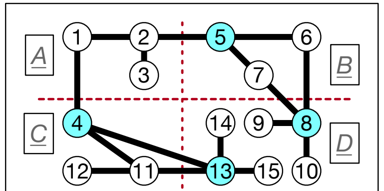
If there is opportunity for improving the SOTA?

Motivation

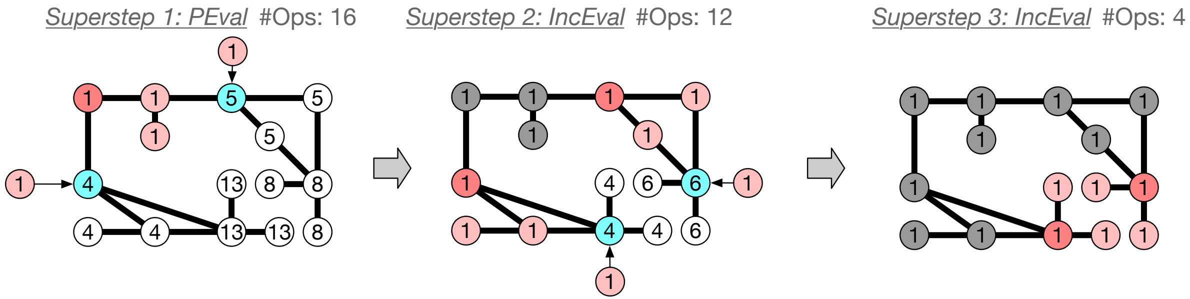
Graph-centric (GC) vs Vertex-centric (VC)



(b) VC execution in 5 supersteps.



(a) Input graph G.



(c) GC execution in 3 supersteps.

- ✓ **VC** takes many computations steps to propagate a piece of information from a <u>source</u> to a <u>destination</u>, even if both appear in the same partition.
- ✓ **GC** allows information to flow freely inside a partition.

How to leverage GC?

Challenges & Opportunities

Parallelism

- ✓ GC exploits data-partitioned parallelism only. With limited memory capacity, it would result in either **underutilization of the CPU** or **graph fragmentation.**

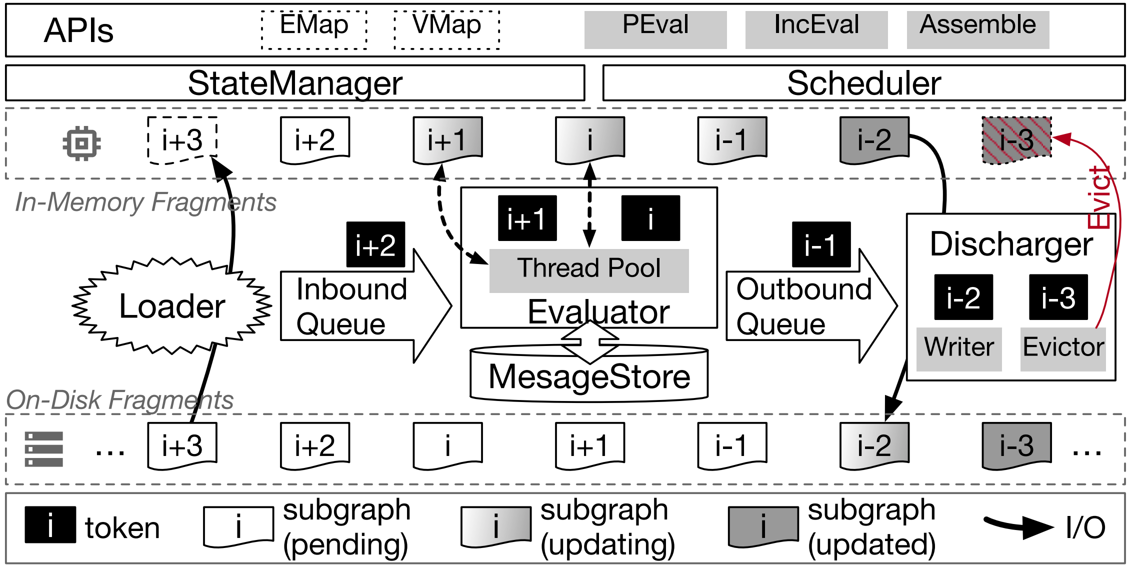
Out-of-core computation

- ✓ A out-of-core system has to resort to secondary storage. Managing the in-memory and the on-disk parts of an input graph is crucial to performance.

Require solutions.

The characters of MiniGraph

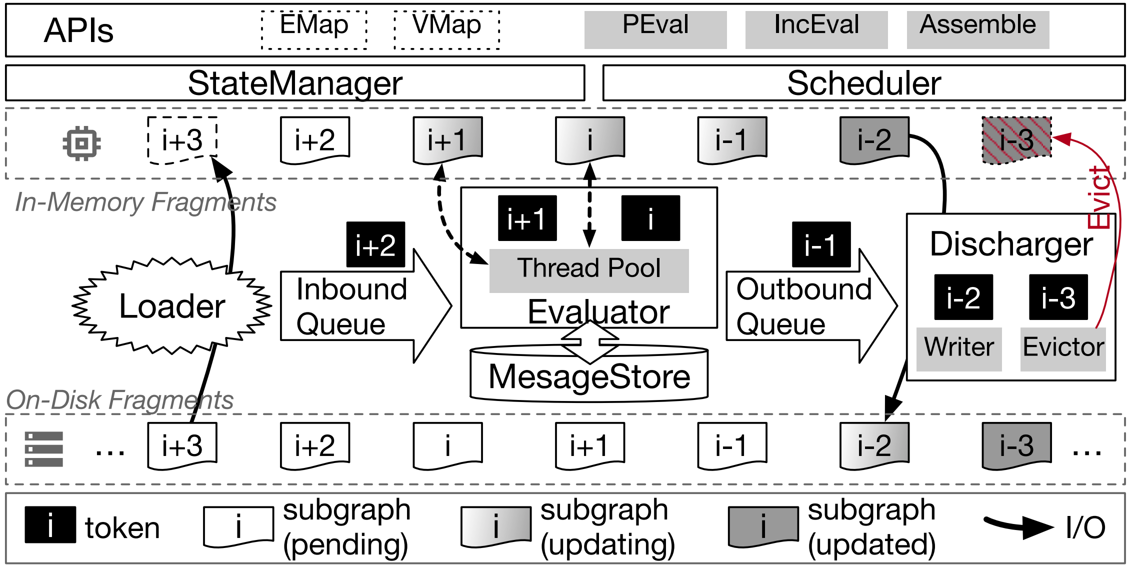
- √ **A pipelined architecture** to overlap I/O and CPU operations.



MiniGraph Architecture

The characters of MiniGraph

- √ **A pipelined architecture** to overlap I/O and CPU operations.
  - **Loader** continuous reads a memory absent subgraphs from disk.
  - **Evaluator** is responsible for execution of an application.
  - **Discharger** writes the data back to the disk.

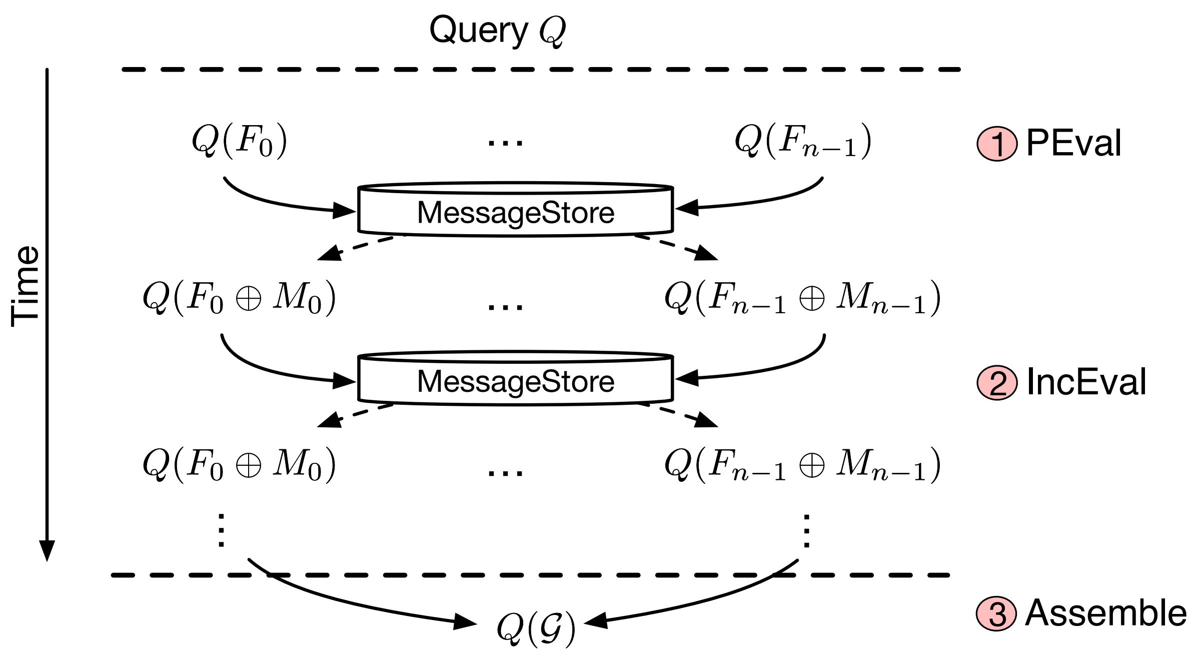


MiniGraph Architecture

The characters of MiniGraph

- √ **A pipelined architecture** to overlap I/O and CPU operations.
- √ **A hybrid parallel model** to support both the *data-partitioned parallelism of **GC*** and the *operation-level parallelism of **VC.***

PIE model



Enrich inter-G. GC parallelism with intra-G. VC parallelism

MiniGraph Architecture

The characters of MiniGraph

√ **A pipelined architecture** to overlap I/O and CPU operations.

√ **A hybrid parallel model** to support both the *data-partitioned parallelism of* ***GC*** and the *operation-level parallelism of* ***VC.***

PEval + EMap/VMap (VC)

HashMin algorithm.
- Init: each vertex is assigned a distinct numeric label
- Run: each vertex collects the labels from its neighbors and update its own label with minimum
- Border vertices: with an edge to another fragment.

Push updates to border vertices.

IncEval + EMap/VMap (VC)

Incremental HashMin algorithm.
- Run: each vertex collects the labels from its neighbors and update its own label
- Messages M_i: changed for border vertices of F_i.

pull M_i from

Assemble

The union of all partial results.

Enrich inter-G. GC parallelism with intra-G. VC parallelism

MiniGraph Architecture

The characters of MiniGraph

- **A pipelined architecture** to overlap I/O and CPU operations.
- **A hybrid parallel model** to support both the *data-partitioned parallelism of **GC*** and the *operation-level parallelism of **VC.***
- **Two-level parallelism:** inter-subgraph parallelism via high-level GC abstraction, and intra-subgraph parallelism for low-level VC operations.

PEval + EMap/VMap (VC)

HashMin algorithm.
- Init: each vertex is assigned a distinct numeric label
- Run: each vertex collects the labels from its neighbors and update its own label with minimum
- Border vertices: with an edge to another fragment.

Push updates to border vertices.

IncEval + EMap/VMap (VC)

Incremental HashMin algorithm.
- Run: each vertex collects the labels from its neighbors and update its own label
- Messages M_i: changed for border vertices of F_i.

pull M_i from

Assemble

The union of all partial results.

Enrich inter-G. GC parallelism with intra-G. VC parallelism

MiniGraph Architecture

The characters of MiniGraph

- ✓ **A pipelined architecture** to overlap I/O and CPU operations.
- ✓ **A hybrid parallel model** to support both the *data-partitioned parallelism of* ***GC*** and the *operation-level parallelism of* ***VC.***
- ✓ **Two-level parallelism:** inter-subgraph parallelism via high-level GC abstraction, and intra-subgraph parallelism for low-level VC operations.
- ✓ **A learned scheduler:** to further improve hardware utilization.

PEval + EMap/VMap (VC)

HashMin algorithm.
- Init: each vertex is assigned a distinct numeric label
- Run: each vertex collects the labels from its neighbors and update its own label with minimum
- Border vertices: with an edge to another fragment.

Push updates to border vertices.

IncEval + EMap/VMap (VC)

Incremental HashMin algorithm.
- Run: each vertex collects the labels from its neighbors and update its own label
- Messages M_i: changed for border vertices of F_i.

pull M_i from

Assemble

The union of all partial results.

Enrich inter-G. GC parallelism with intra-G. VC parallelism

Learned Scheduling

The scheduling problem

- ✓ When to load and process a subgraph?
- ✓ How to allocate resources to maximize two-level parallelism?

A learned model

$$\mathcal{C}_{\mathcal{A}_{PIE}}(F_i) = \sum_{u \in F_i} h_{\mathcal{A}_{PIE}}(\overline{x_i}(u))$$

- ✓ Where $\overline{x_i}(u)$ takes into account the **average in/out-degree** of all vertices and the **number of u's mirror** across all fragments.
- ✓ Collecting training data from log.

Goal

$$\arg\min_{\mathcal{S}} \max_{i \in [0,n)} \{t_i + \mathcal{C}_{\mathcal{A}}(F_i, p_i)\}$$

- ✓ It is in NPC.

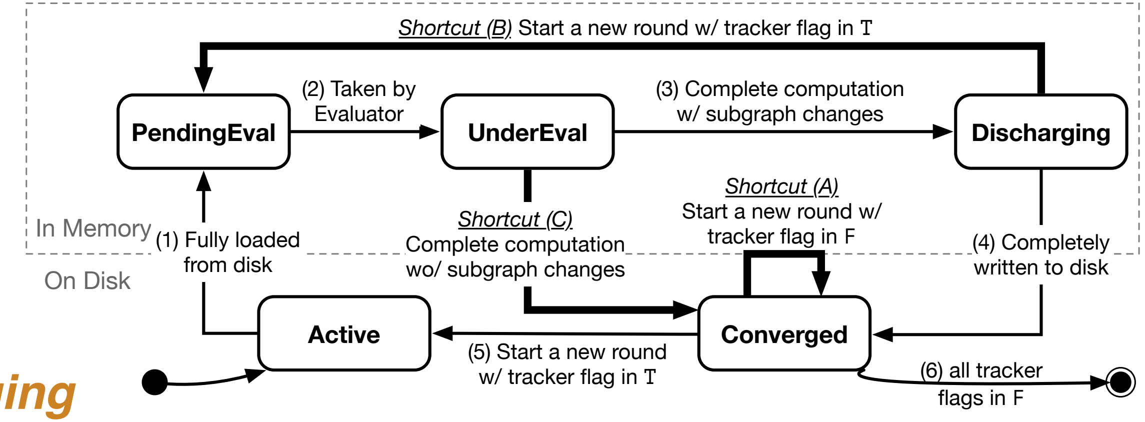
Scheduling strategy

- ✓ **Tentative resource allocation**: allocates resources based on the subgraph size and the memory size.
- ✓ **Greedy subgraph processing:** Scheduler keeps track of a list of pending subgraphs, sorted by $\mathcal{C}_{\mathcal{A}}(F_i, \hat{p}_i)$.

StateManager: a light weight state machine for optimization

Subgraphs states management

- ✓ Targets: Manage subgraphs, determine if the program is finished, and optimize I/O.
- ✓ At any point of time, F_i is in one of the five states: *Active*, *PendingEval*, *UnderEval*, *Converged*, *Discharging*。
- ✓ In-memory: *PendingEval, UnderEval, Discharging*
- ✓ On-disk: *Active, Converged.*



A state machine

I/O optimization

- ✓ ShortCut A: If F_i requires no further processing, we can skip handling subgraph F_i in the round. **(Avoid both Read&Write)**
- ✓ ShortCut B: F_i is set to *PendingEval* directly, such that it starts the new round without going through the disk. **(Avoid Read)**
- ✓ ShortCut C: IncEval(F_i) completes with no changes, F_i skips *Discharging* and is set to *Converged* directly. **(Avoid Write)**

Experimental setting

Datasets

Name	Type	$\|V\|$	$\|E\|$	MaxDegree	Raw Data
roadNetCA [1]	road network	2M	2.7M	23	83MB
skitter [42]	network topology	1.6M	11M	35455	142MB
twitter [8, 40]	social network	41.6M	1.5B	3M	25GB
friendster [5]	social network	65.6M	1.8B	5124	30.14GB
web-sk [55]	Web	50M	1.9B	8.5M	32GB
clueWeb [55]	Web	1.7B	7.9B	6.4M	137GB

Testbed

- ✓ Ubuntu Server 20.04 LTS
- ✓ Intel Core i9-7900X CPU @3.30GHz
- ✓ 13.75MB LLC
- ✓ 10 cores (20 hyper threads)
- ✓ 64GB of DDR4-2666 memory
- ✓ 1TB WD blue SATA SSD, whose read throughput is 560MB/s.

Baseline

Out-of-core

- ✓ GridGraph[ATC'15],GraphChi[OSDI'12], XStream[SIGOPS'13]

Distributed

- ✓ GraphScope[VLDB'21],Gluon[PLDI'18]

Applications

- ✓ WCC
- ✓ PageRank
- ✓ SSSP
- ✓ BFS
- ✓ Random Walk
- ✓ Simulation

Result

Experimental results overview

Data	Memory Budget	#Partitions (PR/Others)	SSSP				WCC				PR			
			MiniGraph	GraphChi	GridGraph	XStream	MiniGraph	GraphChi	GridGraph	XStream	MiniGraph	GraphChi	GridGraph	XStream
roadNetCA	100%	1/1	8.66	22.5 (2.6×)	10.55 (1.2×)	**2 (0.2×)**	**2.76**	17.2 (6×)	18.22 (6.6×)	2.93 (1.1×)	**0.25**	0.91 (3.5×)	0.71 (2.7×)	2.34 (2.6×)
skitter	100%	1/1	0.53	1.64 (78.5×)	**0.35 (0.67×)**	0.69 (1.3×)	**0.16**	18.43 (115.2×)	0.33 (2.1×)	0.59 (3.9×)	**0.27**	1.27 (4.7×)	0.82 (3.0×)	0.98 (3.6×)
twitter	50% (12.5GB)	4/10	**150.8**	802.8(5.3×)	195.4(1.29×)	2365(15.6×)	**159.5**	594.8(3.7×)	186(1.2×)	1983(12.4×)	**224.2**	782.1(3.5×)	371.3(1.7×)	2183(9.7×)
friendster	50% (15.07GB)	4/10	**201.8**	535(2.7×)	293.1(1.45×)	3061(15.2×)	**171.8**	1636(9.5×)	204.7(1.2×)	2037(11.8×)	**190.104**	450.7(1.9×)	485.3(1.9×)	2685(11.3×)
web-sk	50% (16GB)	4/10	**326.4**	1140(3.5×)	917.9(2.8×)	9437 (28.9×)	**172**	620.1(3.6×)	704.6(4.1×)	4056(23.5×)	**248.3**	2288(9.2×)	395(1.6×)	2903(11.7×)
clueWeb	47% (64GB)	4/10	**2514**	/	11534 (4.59×)	/	**2742**	/	11665 (4.25×)	/	**2022**	/	3803(2.1×)	/
	10% (13.7GB)	20/50	**5871**	/	/	/	**7486**	/	/	/	**2979**	/	/	/

Findings

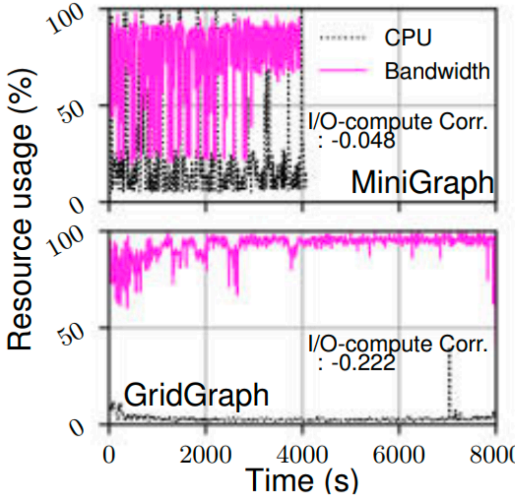
- ✓ MiniGraph consistently outperforms the prior single-machine systems under all out-of-core workloads. It is up to **4.6×**, **9.5×** and **28.9×** faster than GridGraph, GraphChi and XStream, respectively.

Result: Runtime statistics and comparison over resource usage

Runtime statistics for SSSP, WCC and PR

Dataset	Metric	SSSP		WCC		PR	
		MiniGraph	GridGraph	MiniGraph	GridGraph	MiniGraph	GridGraph
friendster	# Supersteps	**8**	32	**6**	21	8	10
	Disk Read (GB)	**78**	115.1	**74**	135	**107**	160
	Shortcut I/O (GB)	**-12**	N/A	**-12**	N/A	**-10.4**	N/A
	Avg. CPU Util.	**33.74%**	4.45%	**48.2%**	6.83%	**68.46%**	62.38%
	I/O-Compute Corr	**0.095**	-0.113	**0.163**	-0.202	**0.185**	-0.156
	Cache Hits	**45.33%**	9.59%	**48.25%**	12.04%	34.8%	**36.2%**
web-sk	# Supersteps	**10**	63	**9**	120	15	20
	Disk Read (GB)	**112.5**	232	**81.9**	367	**87**	232
	Shortcut I/O (GB)	**-30.9**	N/A	**-6.1**	N/A	**-20.9**	N/A
	Avg. CPU Util.	**15.76%**	5.83%	**25.04%**	5.16%	42%	**42%**
	I/O-Compute Corr	**0.008**	0.003	**0.013**	0.009	**0.082**	-0.039
	Cache Hits	**50.89%**	6.37%	**37.42%**	11.63%	**50.22%**	46.04%

CPU & I/O utilization: WCC over clueWeb

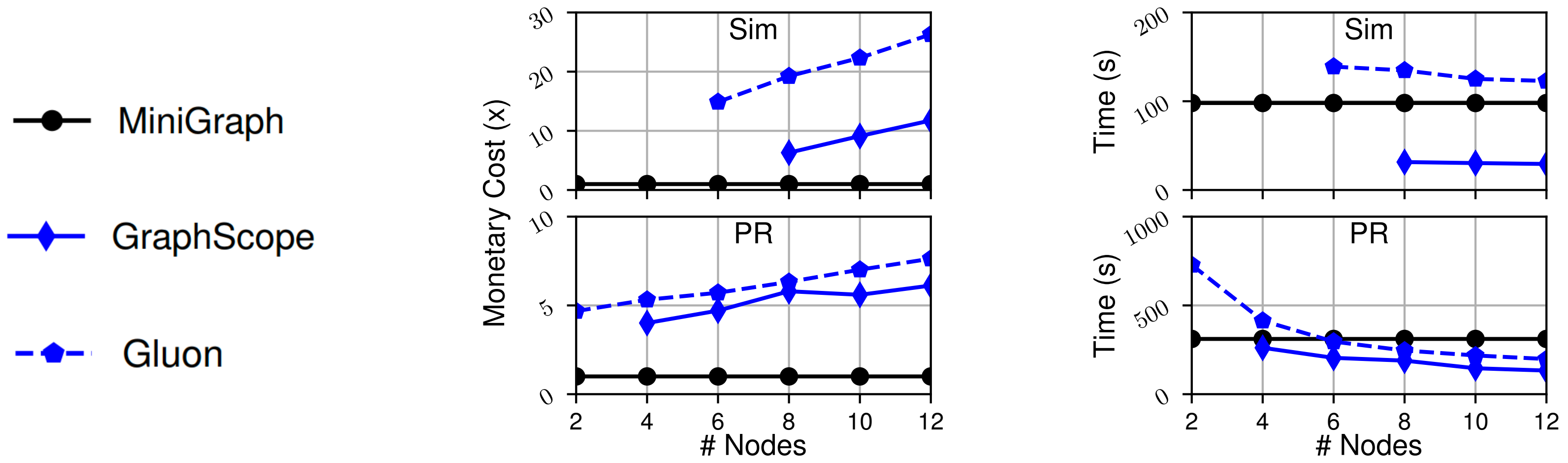


Findings

- ✓ Under BSP, MiniGraph requires only a fraction of supersteps (<29%) and disk read traffic (<53.3%) of GridGraph for SSSP and WCC.
- ✓ MiniGraph improves the CPU utilization of GridGraph, the best-performing baseline, by up to 41.4%.
- ✓ MiniGraph's shortcut optimization effectively reduces I/O cost, especially

Result: Runtime statistics and comparison over resource usage

MiniGraph VS distributed systems



Findings

√ MiniGraph works better than Gluon, a distributed graph analysis system, with 12 machines on a graph simulation task, and saves the monetary cost of multi-machine systems from 3.0× to 13.9×.

Conclusion

MiniGraph is an out-of-core system for graph computations. It is the first single-machine system that extends graph-centric (GC) model from multiple machines to multiple cores.

It shows that GC speeds up beyond-neighborhood and reduces I/O.

https://github.com/SICS-Fundamental-Research-Center/MiniGraph

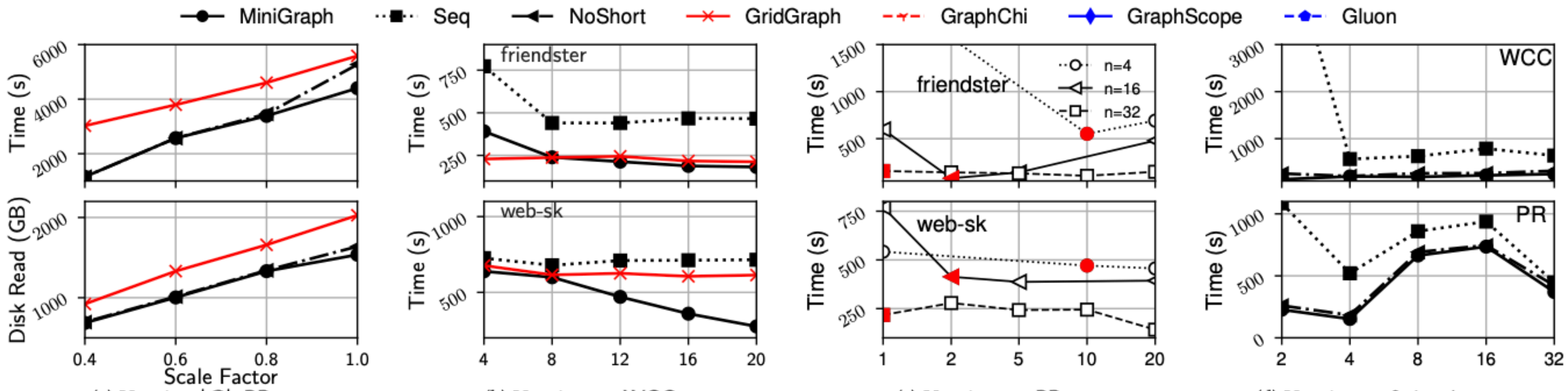
Thanks!

I am looking for postdoctoral position. Please contact me if you are interested.

Email: zhuxk@buaa.edu.cn

Result: other results II

Scalability of MiniGraph



(a) Varying $|G|$: PR. (b) Varying m: WCC. (c) Varying p_i: PR. (d) Varying n: friendster.

Accuracy and effectiveness of cost model formulations

Cost model	$C_{\mathcal{A}}$	Model (a)	Model (b)
Normalized loss over S_{test}	0.16	0.22	0.22
Normalized loss over S'_{test}	0.40	0.50	0.43
Improvement web-sk (%)	39.0%	27.2%	27.3%
Improvement clueWeb (%)	30.0%	16.5%	17.1%