Non-blocking Dynamic Unbounded Graphs with Worst-case Amortized Bounds

Bapi Chatterjee
IIIT Delhi

Sathya Peri,
Komma Manogna
IIT Hyderabad

Muktikanta Sa
Telecom SudParis, IP Paris

OPODIS 2021

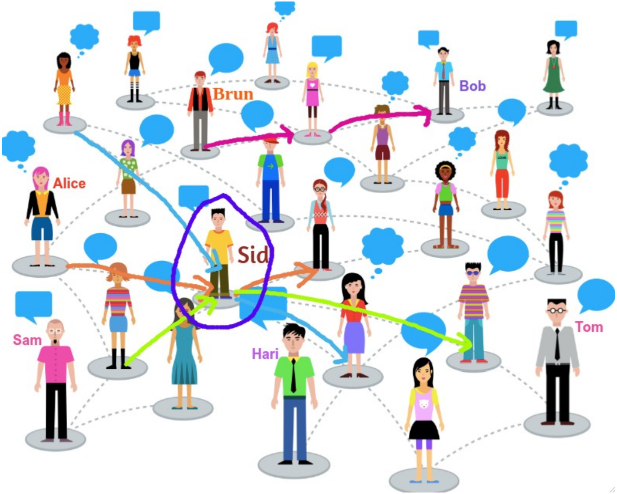
Motivation

Dynamic Graph

Motivation

Dynamic Graph

Motivation



- *reachable* path
- *shortest path*
- *influential* person

The ADT Operations

1. AddVertex
2. RemoveVertex
3. ContainsVertex
4. AddEdge
5. RemoveEdge
6. ContainsEdge
7. BFS
8. SSSP
9. BC

The Graph Data Structure

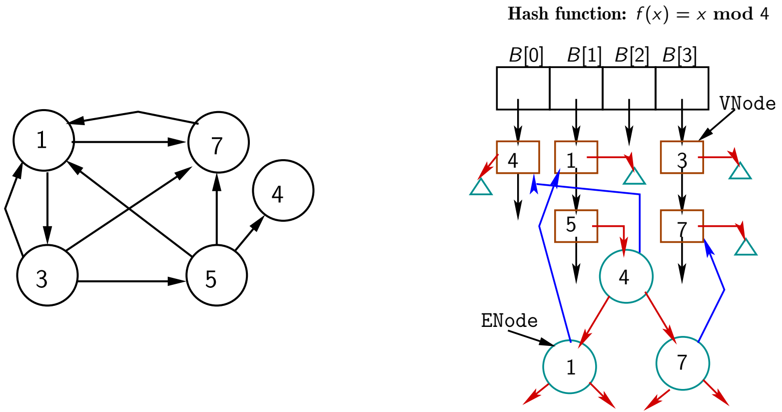


Figure 1: A directed graph and its representation. Graph composition of lock-free sets: a lock-free hash-table and multiple lock-free binary search trees (BSTs).

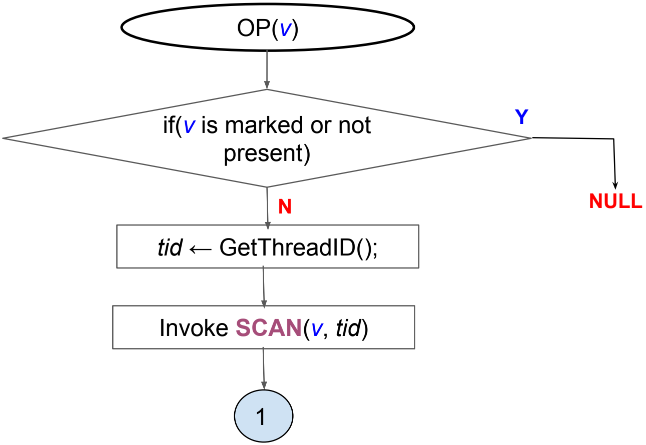
The Graph Framework

1. Three practical operations/queries:
   - Breadth First Search (BFS)
   - Single Source Shortest Path (SSSP)
   - Betweenness Centrality (BC)
2. Dynamic updates of edges and vertices:
   - AddVertex
   - RemoveVertex
   - AddEdge
   - RemoveEdge
3. Non-blocking progress with linearizability.
4. A light memory footprint.

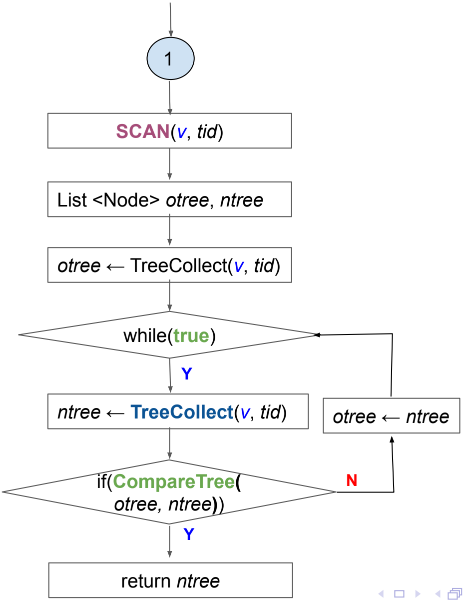
We call it **PANIGRAHAM** [a]: **Pr**actic**a**l **N**on-blocki**n**g **Gra**ph Algorit**hm**s.

[a] Panigraham is the Sanskrit translation of Marriage, which undoubtedly is a prominent event in our lives resulting in networks represented by graphs.

Working Flow of Graph Queries



Working Flow of Graph Queries



Working Flow of Graph Queries

1. During the edge modification operations the atomic counter `ecnt` at each vertex is necessarily incremented.
2. The **TreeCollect** method returns the BFS-tree of VNode in the BFS traversals.
3. The comparison of the two BFS-trees is done in the procedure **CompareTree** along with the counters `ecnt` of the VNodes contained in them.
4. Until **CompareTree** method returns true, the **TreeCollect** method is invoked by copying *ntree* to *otree*. The time when the **CompareTree** method returns true, the SCAN method returns *ntree*.

Correctness and Progress Guarantees

Theorem 1:

1. The ADT operations are linearizable.

Theorem 2:

1. The queries are individually obstruction-free.
2. The algorithm that implements the ADT is lock-free.

Proofs of Theorem 1 and 2 and complexity analysis are shown in the technical report.[b]

[b] https://arxiv.org/abs/2003.01697

Experiments

- Intel(R) Xeon(R) E5-2690 v4 CPU containing 14 cores running at 2.60GHz on two sockets. Each core supports 2 logical threads.
  - A total of 56 logical cores.
- Implementation in C++ without any garbage collection. Multi-threaded implementation is based on Posix threads.
- We loaded a R-MAT graph, thereafter performed warm-up operations, followed by an end-to-end run of 10^4 operations in total.

Workload Distributions

Graph Operations: OP, ADDVERTEX, REMOVEVERTEX, ADDEDGE, and REMOVEEDGE

- **2/49/49:** (2%, 24.5%, 24.5%, 24.5%, 24.5%)
- **5/47.5/47.5:** (5%, 23.75%, 23.75%, 23.75%, 23.75%)
- **10/45/45:** (10%, 22.5%, 22.5%, 22.5%, 22.5%)

Workload Distributions

Graph Operations: OP, ADDVERTEX, REMOVEVERTEX, ADDEDGE, and REMOVEEDGE

- **2/49/49:** (2%, 24.5%, 24.5%, 24.5%, 24.5%)
- **5/47.5/47.5:** (5%, 23.75%, 23.75%, 23.75%, 23.75%)
- **10/45/45:** (10%, 22.5%, 22.5%, 22.5%, 22.5%)

We have compared the following cases.

S. No	Label	Explanation
1	**PG-Cn**	Linearizable PANIGRAHAM
2	**PG-Icn**	Inconsistent PANIGRAHAM
3	**Ligra**	Supports BFS, SSSP, and BC
4	**Stinger**	REMOVEVERTEX, ADDEDGE, REMOVEEDGE, and BFS

Results: BFS

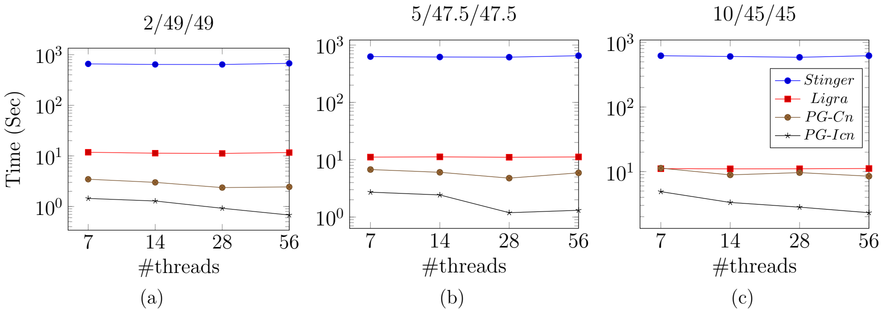
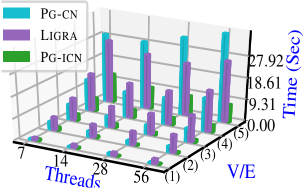
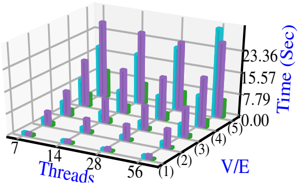


Figure 2: Latency of the executions containing OP: BFS on a R-MAT graph of size $|V| = 131K$ and $|E| = 2.4M$. A total of 10^4 operations were performed. Workload Distributions: BFS, ADDVERTEX, REMOVEVERTEX, ADDEDGE, and REMOVEEDGE :**2/49/49:** (2%, 24.5%, 24.5%, 24.5%, 24.5%)

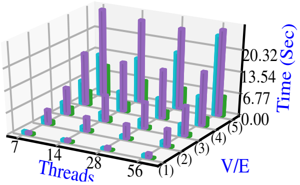
More BFS Results



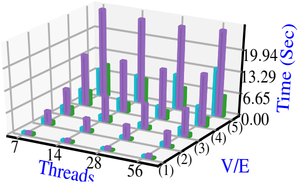
(a) 60/20/20



(b) 40/40/20



(c) 30/50/20



(d) 20/60/20

Figure 3: The dataset sizes as labeled on the y-axis are $\{(V/E), \{(1): 1K/10K, (2): 8K/80K, (3): 16K/160K, (4): 32K/320K, (5): 65K/500K\}$.

Results: SSSP

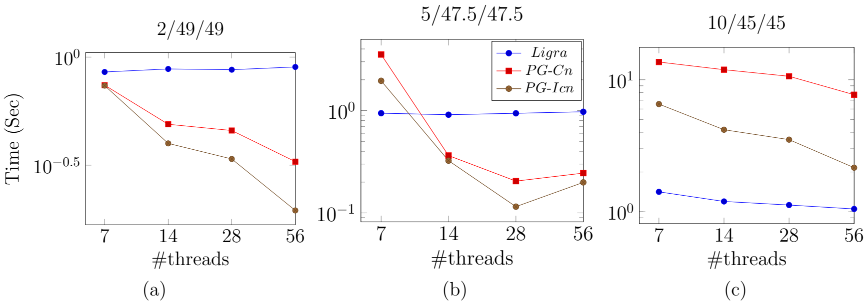
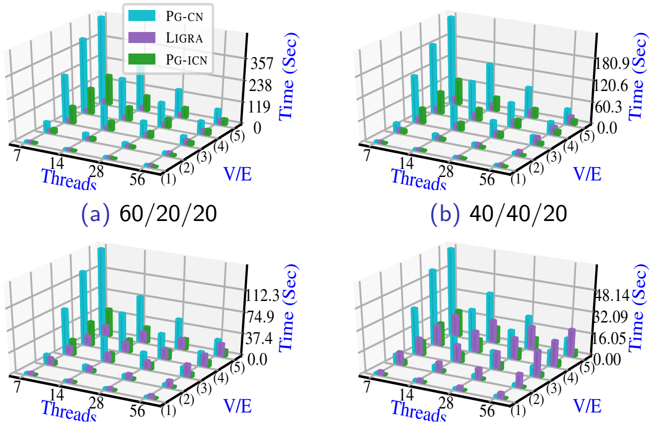


Figure 4: Latency of the executions containing OP: SSSP on a R-MAT graph of size $|V| = 8K$ and $|E| = 80K$. A total of 10^4 operations were performed.



(a) 60/20/20
(b) 40/40/20
(c) 30/50/20
(d) 20/60/20

Figure 5: The dataset sizes as labeled on the y-axis are $\{(V/E), (1): 1K/10K, (2): 4K/30K, (3): 8K/50K, (4): 8K/70K, (5): 8K/80K\}$.

Results: BC

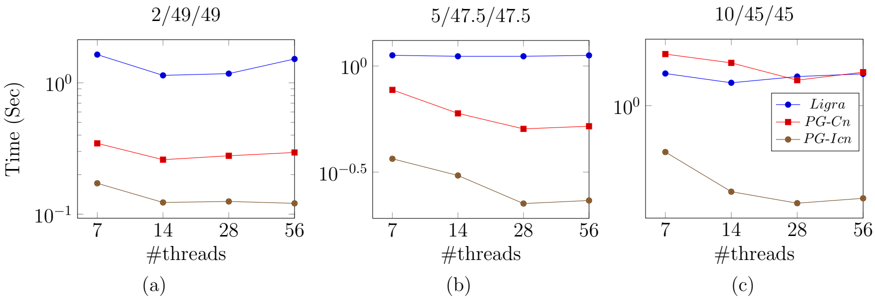
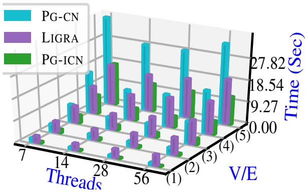
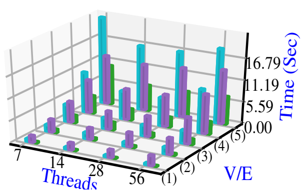


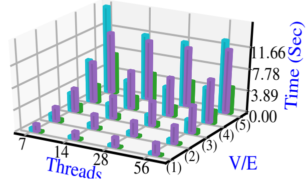
Figure 6: Latency of the executions containing OP: BC on a R-MAT graph of size $|V| = 16K$ and $|E| = 160K$. A total of 10^4 operations were performed.



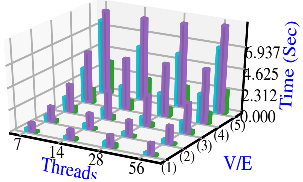
(a) 60/20/20



(b) 40/40/20



(c) 30/50/20



(d) 20/60/20

Figure 7: The dataset sizes as labeled on the y-axis are $\{(V/E), (1) : 1K/10K, (2) : 2K/20K, (3) : 4K/40K, (4) : 8K/80K, (5) : 16K/120K\}$.

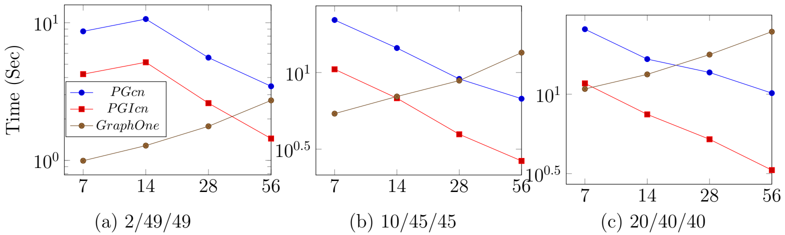


Figure 8: OP: BFS on a graph of size $|V| = 65K$ and $|E| = 500K$. Total 10^4 operations were performed with given distributions. The distributions for each cases is: BFS/ADDEDGE/REMOVEEDGE, e.g., 2/49/49 : $\{$BFS : 2%, ADDEDGE : 49%, REMOVEEDGE : 49%$\}$. X-axis unit is the number of threads.

Memory Footprint

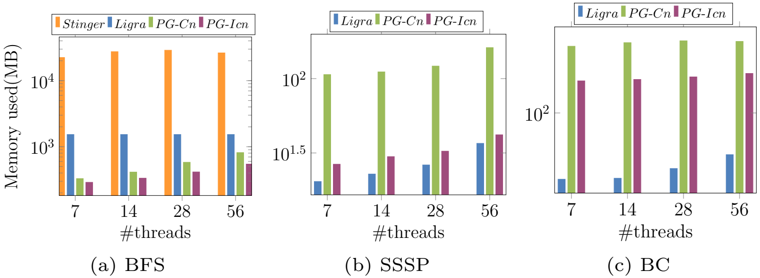


Figure 9: The memory footprint during the run-time corresponding to the workload distribution 10/45/45. BFS: $|V| = 131K$ and $|E| = 2.4M$. SSSP: $|V| = 8K$ and $|E| = 80K$. BC: $|V| = 16K$ and $|E| = 160K$.

Average number of Scans

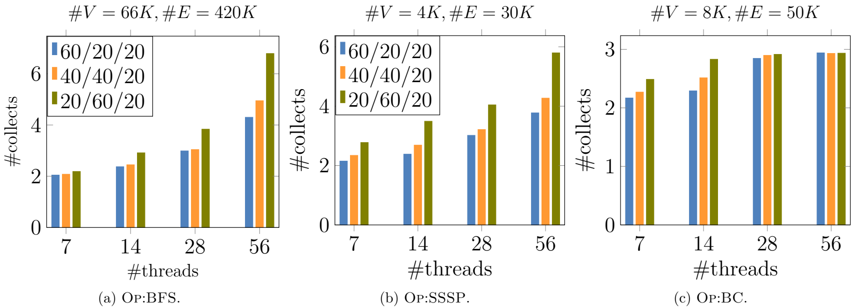


Figure 10: Average number of scans during a query.

Conclusions

1. We implemented a concurrent graph with queries: BFS, SSSP, and BC.
2. We compared these results with Ligra, Stinger, and GraphOne.
3. We extensively evaluate a sample C++ implementation of the algorithm through a number of micro-benchmarks.
4. Non-Blocking implementations handsomely outperform Ligra, Stinger and GraphOne.
5. However, as graph size increases, Ligra starts taking advantage of the parallel implementation.

For More Information

1. The Technical Report is available at:
   https://arxiv.org/abs/2003.01697

2. And the complete source code is available at:
   https://github.com/PDCRL/PANIGRAHAM

Thank You!