INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Indian Institute of Technology Jodhpur

GVE-LPA: Fast Label Propagation Algorithm (LPA) for Community Detection in the Shared Memory Setting

Subhajit Sahu,[1] Kishore Kothapalli,[1] Dip Sankar Banerjee.[2]

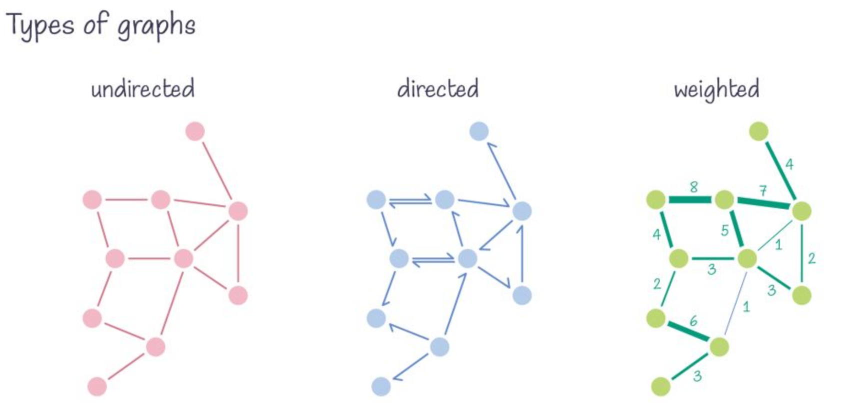
1. IIIT Hyderabad, India
2. IIT Jodhpur, India

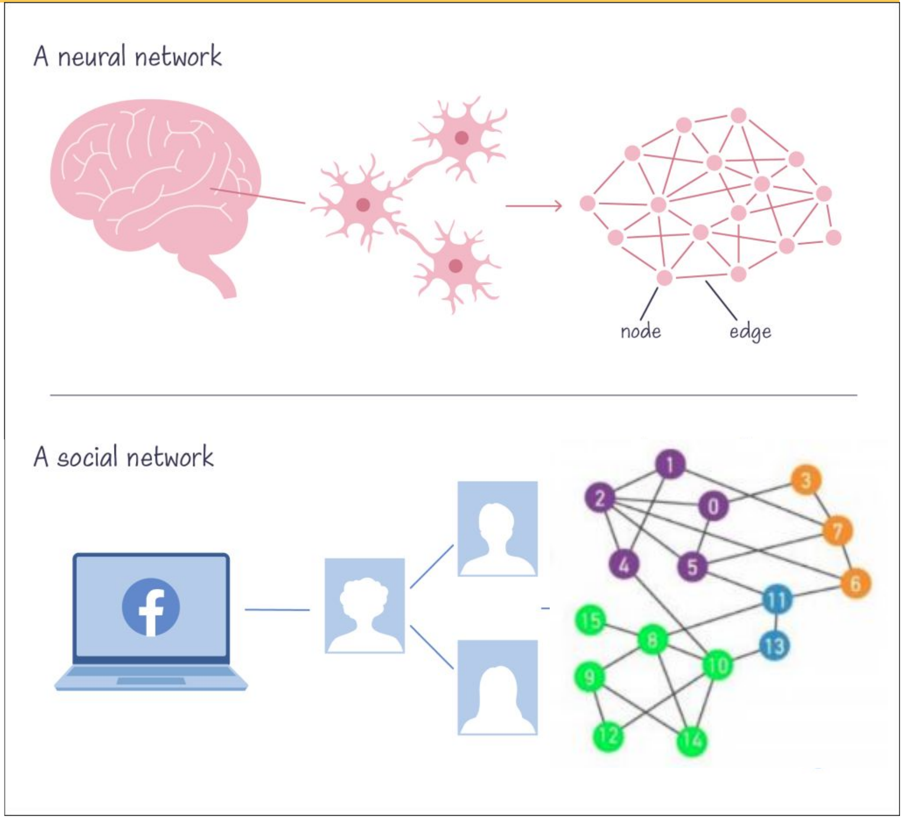
Graphs are everywhere

A **graph** is an abstraction for representing data and relationships between them.

Set of **vertices** and inter-connecting **edges**.

- Web graphs
- Social networks
- Road networks





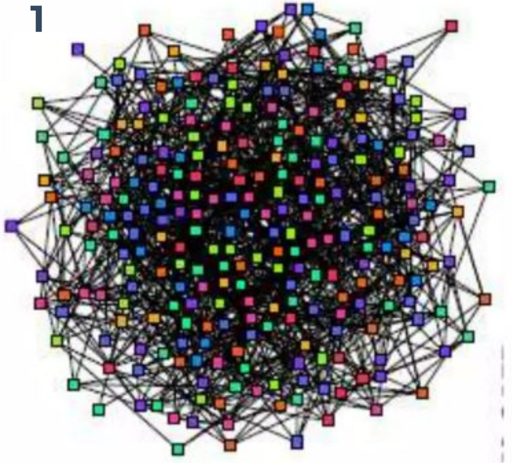
Haque & Andrejevic (2021)

- **Introduction**
- Related work
- Approach
- Evaluation
- Conclusion

What is Community Detection?

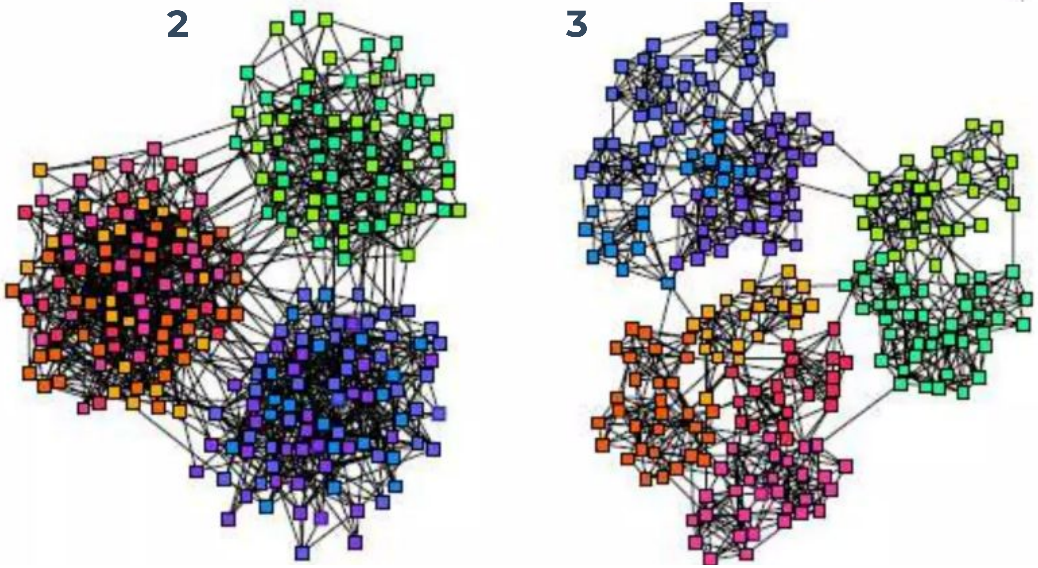
Identifying the **inherent structure** of a graph, which implies its **function**.

In contrast to **classification** / supervised learning, which has known fixed classes.

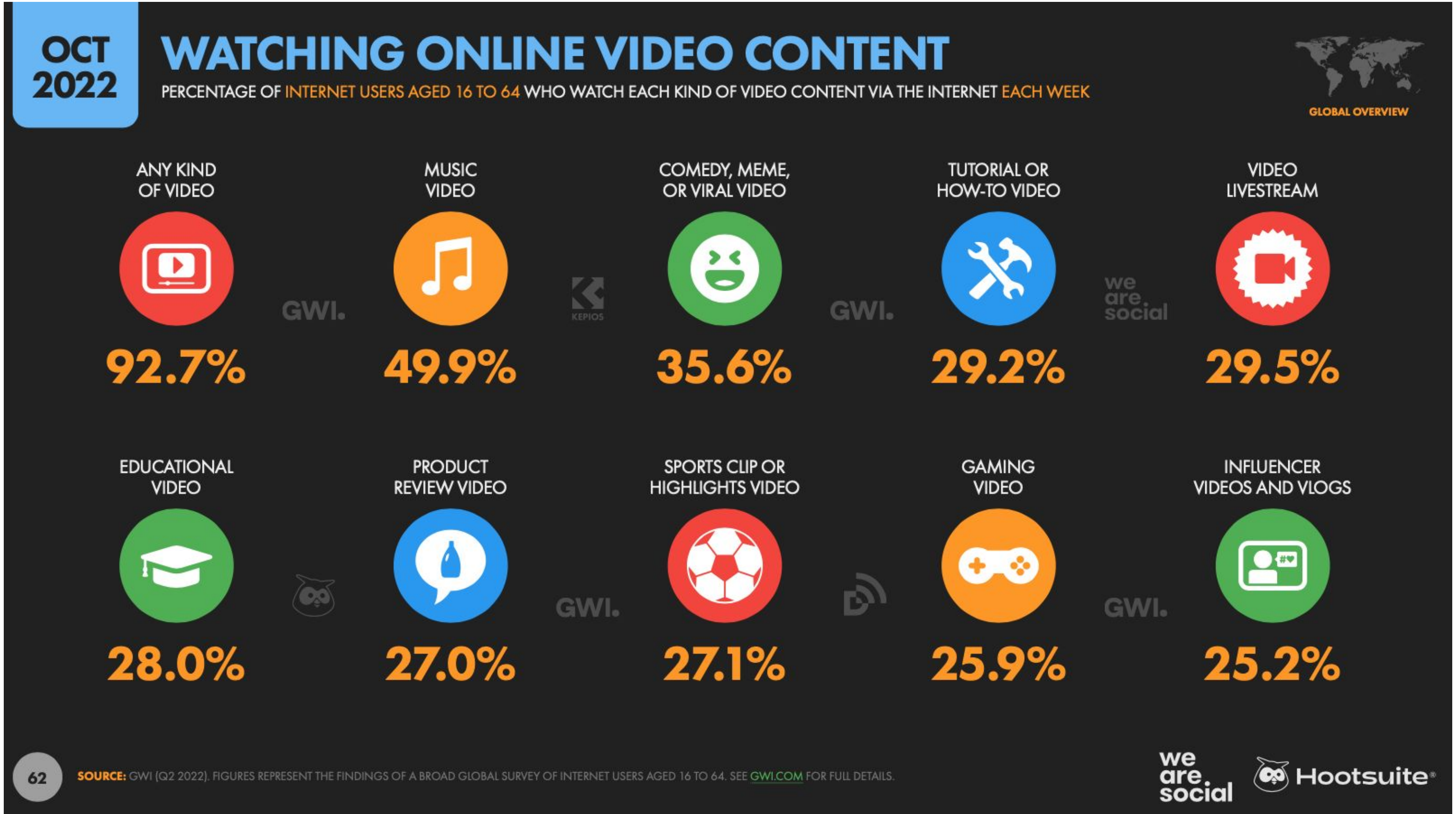


Applicable in many problems.

- Customer segmentation
- Image segmentation
- Anomaly detection
- Graph compression, Partitioning
- Message compression (IoT)



Applications - Customer / content segmentation



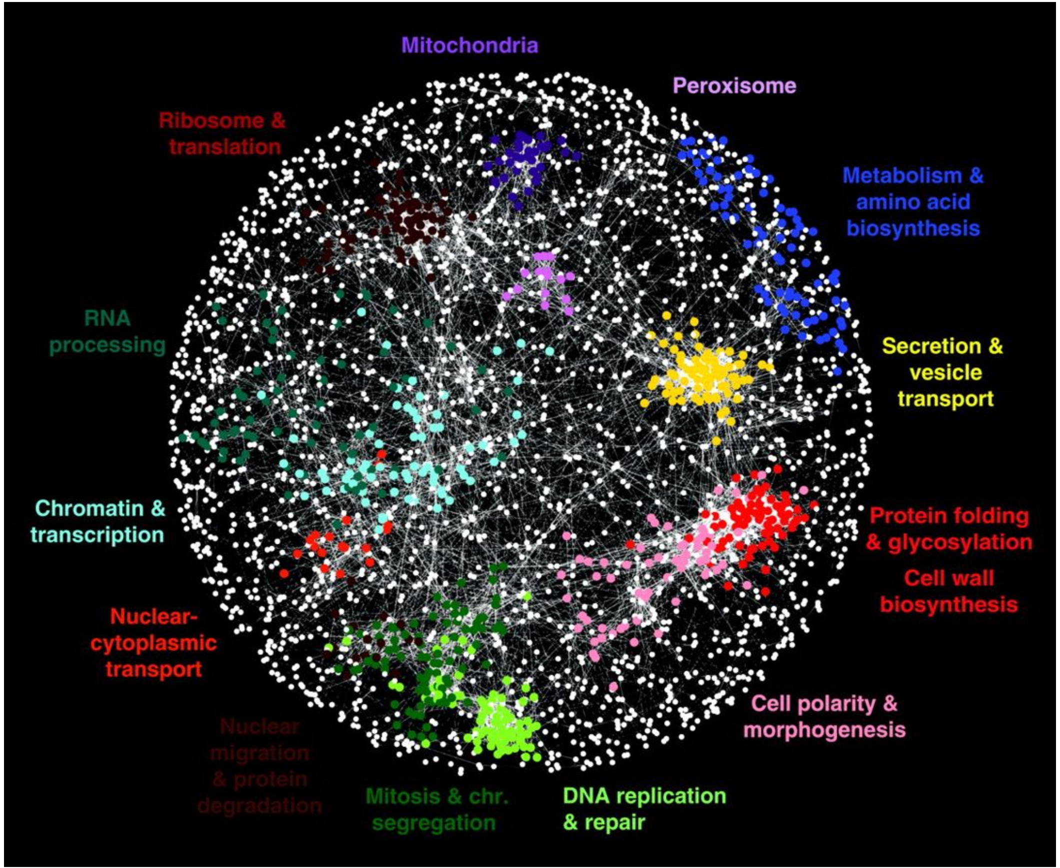
Applications - Gene / protein function prediction

Node:

- Protein

Edge:

- Protein interaction



Applications - Document classification

Node:

- Document

Edge:

- Shared words / phrases

Applications (contd.)

Drug discovery:

Identify groups of **similar compounds** or **target proteins**, facilitating the discovery of new therapeutic agents.

Health domain:

Understanding the dynamics of groups susceptible to epidemic diseases, detecting diseases like lung cancer, and categorizing tumor types using genomic datasets.

Understand the **structure and evolution of metabolic networks**, Gene Regulatory Networks (GRNs), and Lateral Gene Transfer (LGT) networks.

Analysis of **human brain networks**.

Ecological studies:

Determine if **food webs** are organized into **compartments**, where species within the same compartment frequently interact among themselves but have fewer interactions with species in different compartments.

Others:

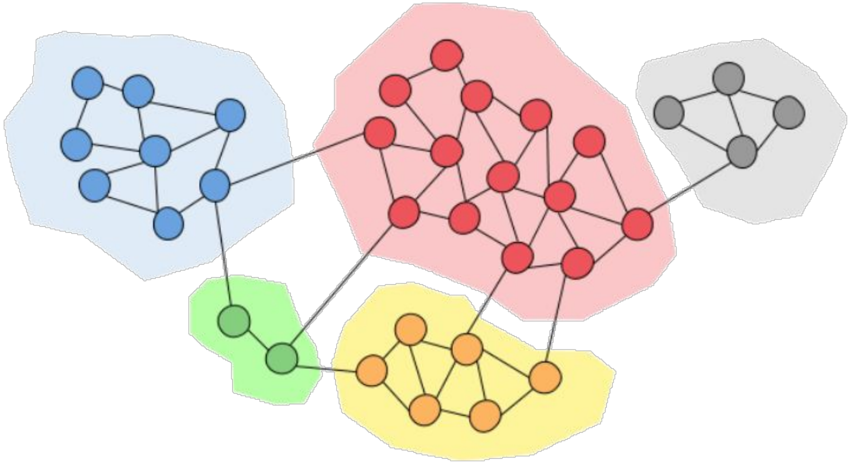
Vertex reordering, graph coarsening, sectionalizing power system (faulty).

What are communities?

A **community** is a subset of a network whose members are *highly connected*, but *loosely connected* to others outside their community.

Neither the number of output communities nor their size distribution is known a priori.

Different community detection methods can *return different communities* these algorithms are **heuristic-based**.



Community types:
- **Disjoint**
- Hierarchical
- Overlapping
- Seed-set expansion

Heuristics:
- Random walk
- **Label propagation**
- Divisive
- Agglomerative

How do you define community quality?

Newman and Girvan introduce **modularity metric** - a **fitness function** that measures **relative density of edges inside** vs outside **communities**.

Between **−0.5** (non-modular clustering) and **1.0** (fully modular clustering).

Optimizing this theoretically results in the best possible **grouping**.

The problem of community detection is then reduced to the problem of modularity maximization which is **NP-hard**.

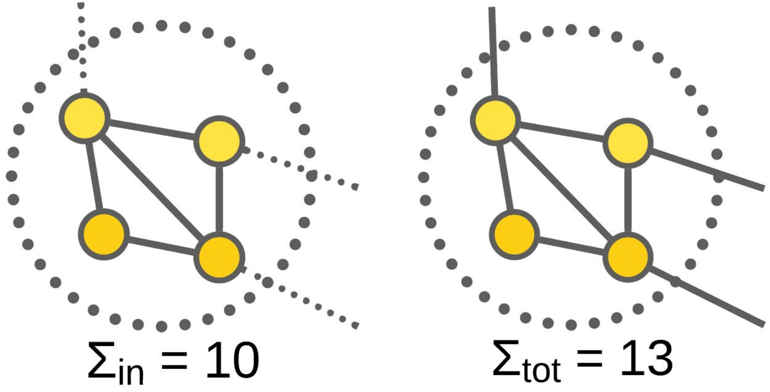
links within community

total links

$$Q(C) = \frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2$$

modularity

community weight



- Introduction
- Related work
- Approach
- Evaluation
- Conclusion

Label Propagation Algorithm (LPA); Raghavan et al. (2007)

This is an implementation of a popular label-propagation based community detection algorithm called Raghavan Albert Kumara (RAK).

Here, every node is **initialized** with a **unique label** and at every step **each node adopts the label that most of its neighbors currently have**.

In this iterative process densely connected groups of nodes form a consensus on a unique label to form communities. The algorithm converges when **n% of vertices don't change** their community membership (**tolerance**).

Semi-supervised learning.

Community Overlap PRopagation Algorithm (COPRA); Gregory (2010)

- Each vertex initializes as its own community (belonging=1).
- Each iteration, a vertex collects labels from its neighborhood.
- It excludes a vertex's own labels, although not explicitly mentioned in paper.
- The collected labels are scaled by edge weights for weighted graph.
- Each vertex picks labels above a certain threshold.
- This threshold is inversely proportional to the max. number of labels.
- If all labels are below threshold, pick a random best label.
- I make a vertex join its own community if it has no labels (not mentioned).
- Selected labels are normalized such that belonging coefficient sums to 1.
- Repeat from 2 until convergence.

Speaker-Listener Propagation Algorithm (SLPA); Xie & Szymanski (2012)

- Each vertex is initialized such that it remembers itself as popular.

- Each neighbor speaks one of the random labels in its memory.

- The vertex (listener) adds the most popular label to its memory.

- Repeat from 2 until a fixed number of iterations is performed (labels - 1).

- I allow early convergence if at least n% of vertices remember their previous label.

- For each vertex, i pick the most popular label in its memory as its community.

LabelRank; Xie & Szymanski (2013)

LabelRank is an iterative algorithm that is based on the concept of propagation of weighted labels on a weighted (directed) network, where the highest weight label determines the community membership of each vertex.

Our implementation of LabelRank differs from the original algorithm in that there is a fixed upper limit on the number of labels per vertex (labelset capacity). Therefore we do not use the cutoff operator (which removes low-weighted labels), but instead trim-off labels if they do not fit within labelset capacity. Labels are sorted by weight such that only low-weighted labels are eliminated.

Csardi and Nepusz (Complex systems 2006) - igraph LPA (seq.)

- **Shuffles node order** each iteration. [Can get expensive]
- Track **dominant labels**, select one randomly. [Given multiple dominant labels; RNG slow]
- **Clear hashtable** by iterating neighbors.
- **No restrict** iteration to a set of active nodes.
- Alternate **label updating** and **control** iters. [Control iter.: check if current label of node is not dominant]
- Converged if labels of **all** nodes dominant. [Large no. of iterations; minimal gain in quality]

Traag and Šubelj (Scientific Reports 2023) - FLPA (seq.)

- Does **not shuffle** node processing order.

- Uses **deque** for managing active nodes.

- Converged when **deque empty**.

- Selects **random dominant label**. [Given multiple dominant labels; RNG slow]

- Converged when **no active nodes**. [Large no. of iterations; minimal gain in quality]

- Label changes? Process neighbors diff. label.

Staudt and Meyerhenke (IEEE TPDS 2016)

- Avoid randomizing processing order (use ||). [Existing; cost; negligible effect]
- **Unnecessary** to **recompute** labels of nodes. [Vertex pruning]
- Restrict iteration to a set of **active nodes**. [Within threshold; majority iterations of very small frac. of high-deg. nodes]
- **Asynchronous** updating of labels. [Has race conditions; beneficial – random variations, solution diversity; avoid oscillations on bipartite structures]
- OpenMP **guided** thread **scheduling**. [Handle power-law graphs; assign node ranges of decreasing size to threads]

Also propose || Louvain + with refinement, and ensemble processing (merge base algs.).

[Re-evaluate node assignments in view of changes in the next coarser level]

[In ML, weak classifiers are combined to form a strong classifier, find commonality of multiple base algs., coarsen, and apply and final alg.]

Staudt, Sazonovs, and Meyerhenke (Network Science 2016) - NetworKit LPA

- **static, 1** || label initialization. [False sharing - consecutive writes]
- **std::map** for weights linked to labels. [Quite inefficient]
- Convergence **tolerance** of **10^{-5}**. [We use lower with similar quality]
- **Atomically count** updated vertices. [Contention; can use || reduce instead]
- **Boolean vector** to track active nodes. [8-bit integer flag more efficient]

Critical review

We have experimented with **COPRA**, **SLPA**, and **LabelRank**, but found **LPA** to be the **most performant**, while yielding communities of equivalent quality.

1. Computational bottleneck.

2. Energy efficiency.

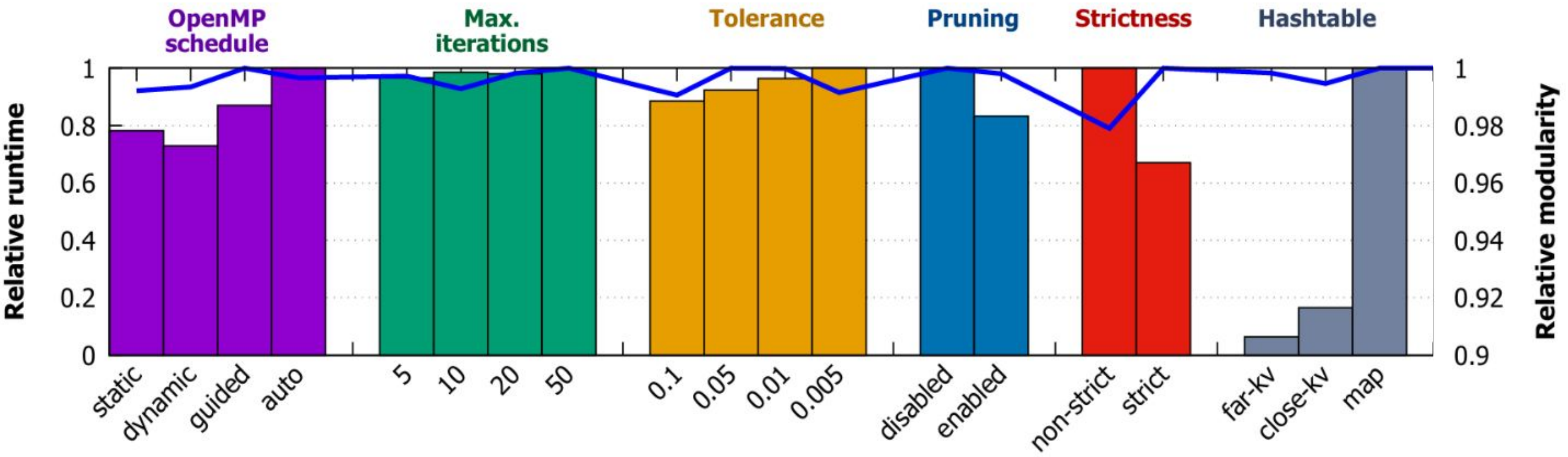
3. Large memory sizes.

4. Existing studies do not study efficient data structures.

5. Optimizations are scattered.

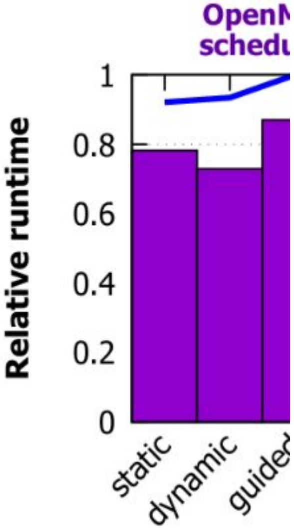
- Introduction
- Related work
- Approach
- Evaluation
- Conclusion

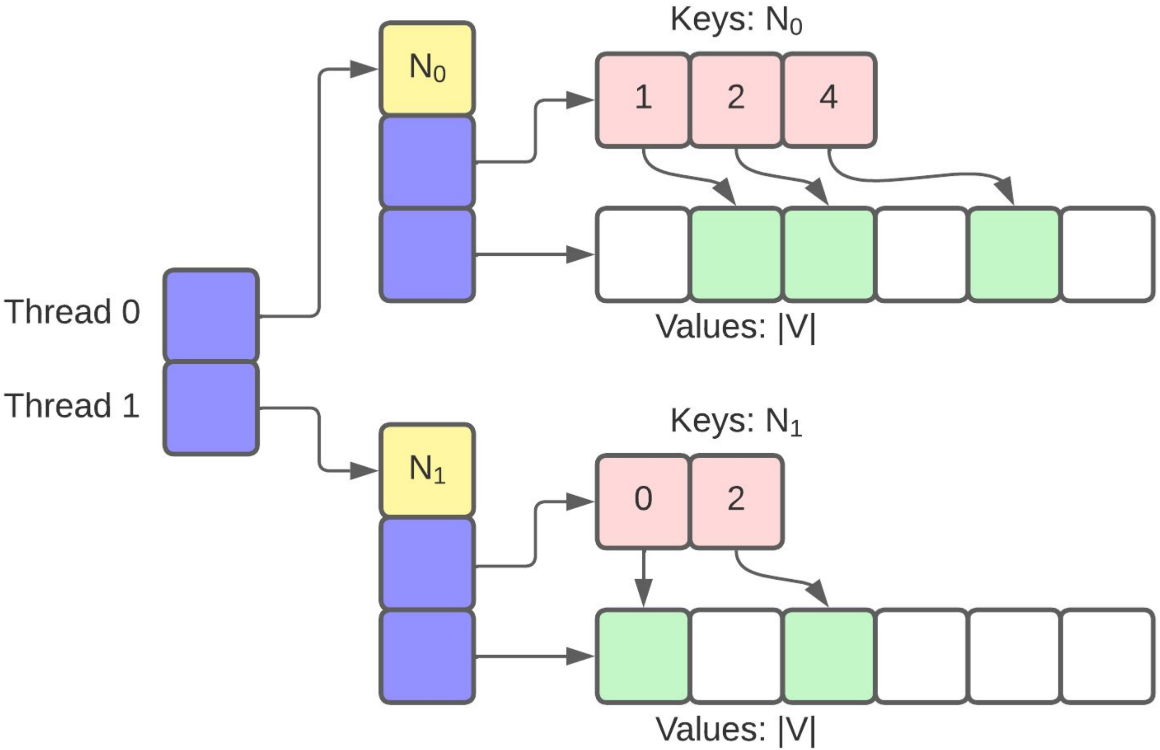
Our Parallel LPA

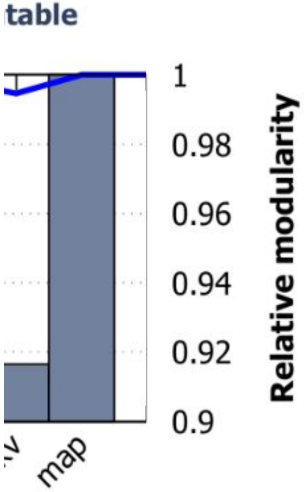


- **Async** version of LPA (distinct sections - faster).
- Dedicated **CF hashtable** per **thread**: **2.6x**.
- OpenMP **dynamic** loop **scheduling** (2048): **1.27X**.
- **Limit** to 20 **iterations**: small.
- **Tolerance** of 0.05: small.
- **Vertex pruning** (**flag** based): **1.17X**.
- **Strict LPA** (vs non-strict): **1.5X**.

Our Parallel LPA







- **Async** vers .
- Dedicated
- OpenMP **d**): **1.17X**.

X.

- Introduction
- Related work
- Approach
- **Evaluation**
- Conclusion

Experimental setup

System:

2 x Intel Xeon Gold 6226R @ 2.9 GHz (16 cores)

376 GB RAM, CentOS 8

GCC 8.5, OpenMP 4.5

32-bit edge weights, 64-bit computation; 64 th.

Dataset: **|V|**: 3.1M to 214M, **|E|**: 25M to 3.8B

Experimental setup

System:

2 x Intel Xeon Gold 6226R @ 2.9 GHz (16 cores)

376 GB RAM, CentOS 8

GCC 8.5, OpenMP 4.5

32-bit edge weights, 64-bit computation; 64 th.

Dataset: |V|: 3.1M to 214M, **|E|**: 25M to 3.8B

Graph	$\|V\|$	$\|E\|$	D_{avg}	$\|\Gamma\|$
Web Graphs (LAW)				
indochina-2004*	7.41M	341M	41.0	147K
uk-2002*	18.5M	567M	16.1	383K
arabic-2005*	22.7M	1.21B	28.2	213K
uk-2005*	39.5M	1.73B	23.7	677K
webbase-2001*	118M	1.89B	8.6	6.48M
it-2004*	41.3M	2.19B	27.9	611K
sk-2005*	50.6M	3.80B	38.5	284K
Social Networks (SNAP)				
com-LiveJournal	4.00M	69.4M	17.4	2.19K
com-Orkut	3.07M	234M	76.2	49
Road Networks (DIMACS10)				
asia_osm	12.0M	25.4M	2.1	278K
europe_osm	50.9M	108M	2.1	1.52M
Protein k-mer Graphs (GenBank)				
kmer_A2a	171M	361M	2.1	40.1M
kmer_V1r	214M	465M	2.2	48.7M

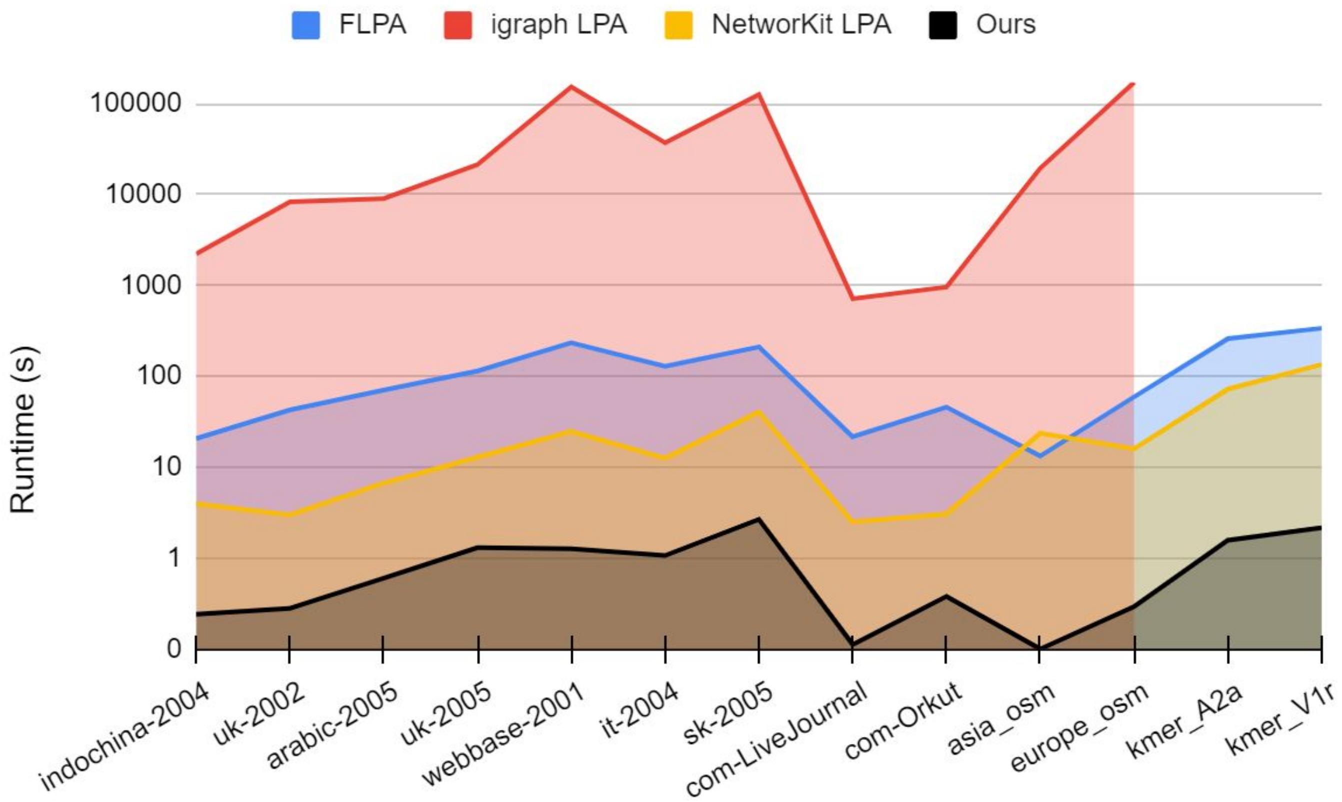
Performance comparison - Runtime

139x FLPA

97000x igraph LPA

40x NetworKit LPA

1.4B edges/s Ours

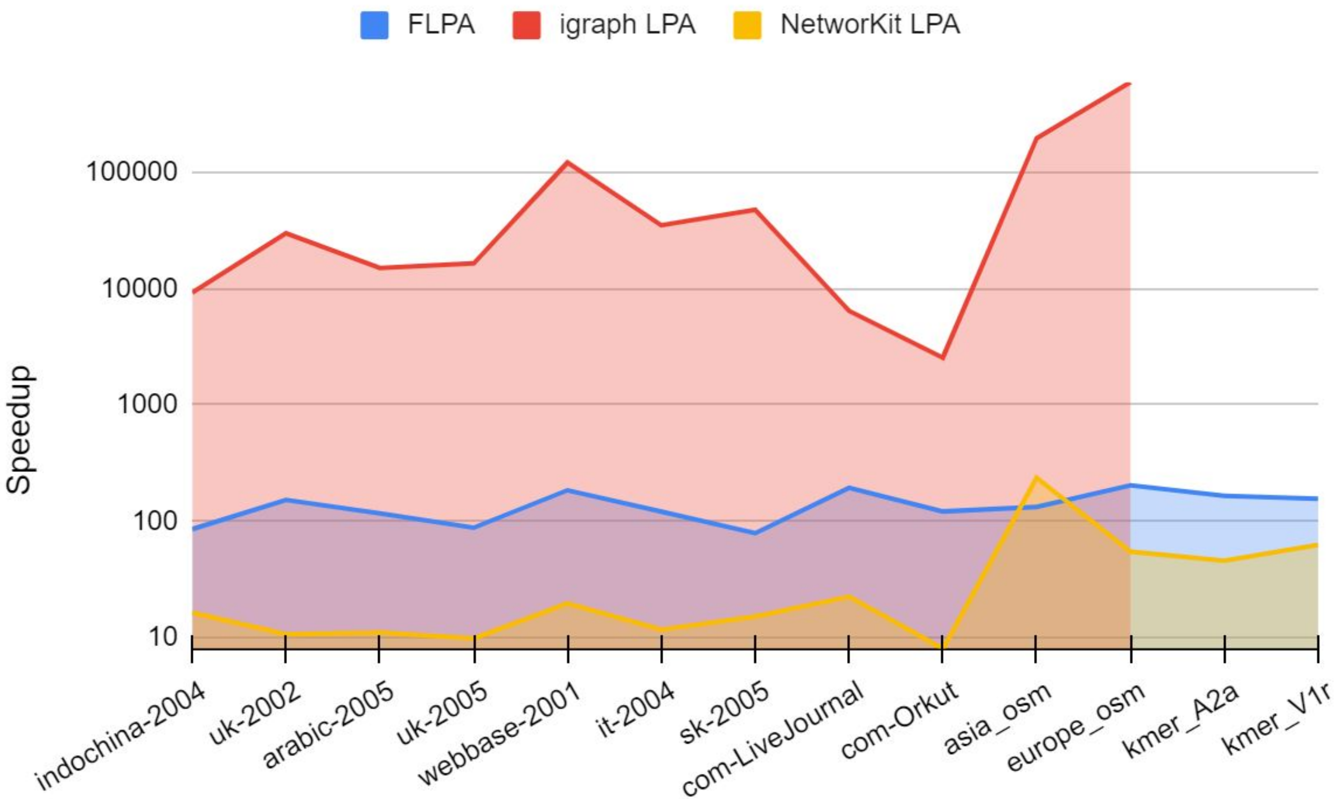


Performance comparison - Speedup

139x FLPA

97000x igraph LPA

40x NetworKit LPA

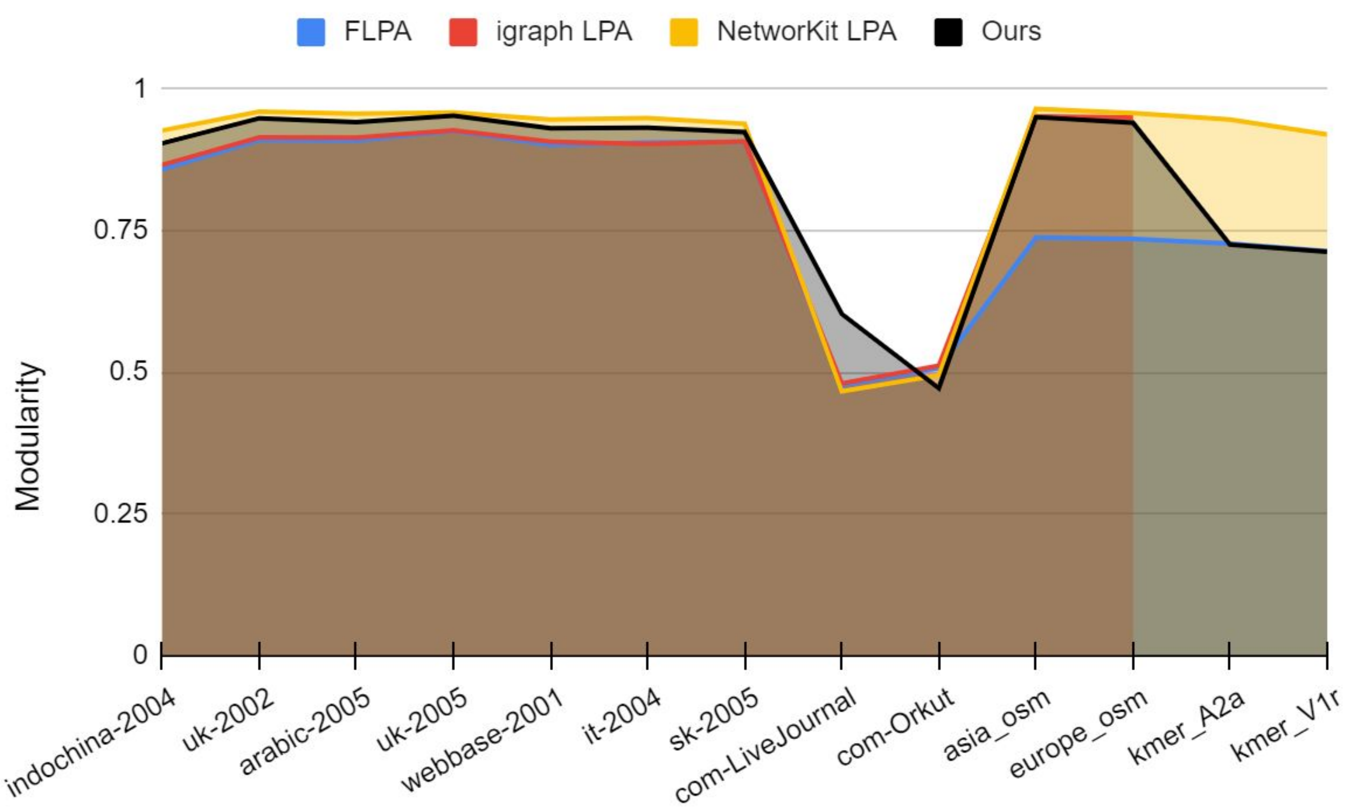


Performance comparison - Modularity

6.6% FLPA

0.2% igraph LPA

-4.1% NetworKit LPA



Strong scaling

With 32 threads:

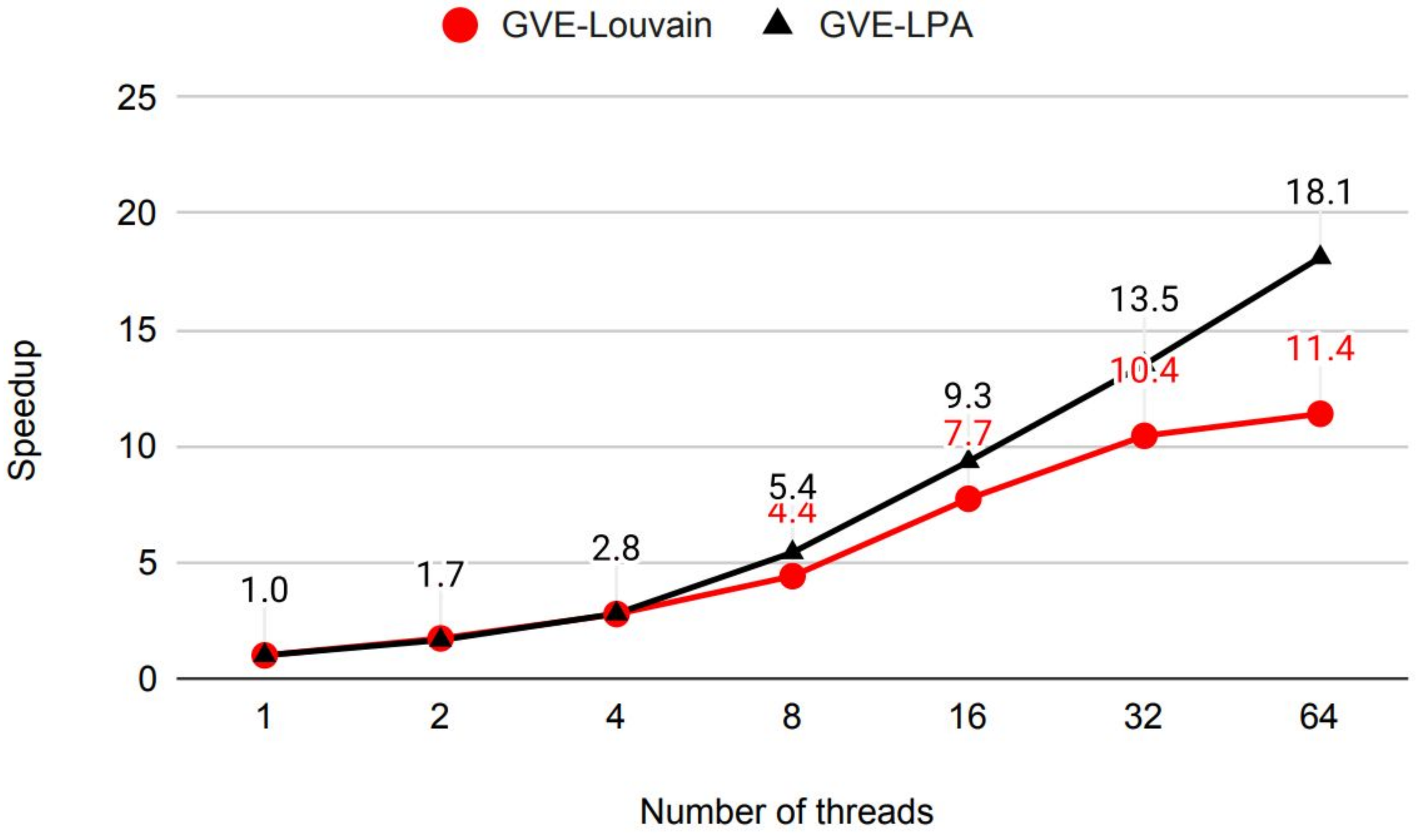
GVE-LPA: 13.5x

GVE-Louvain: 10.4x

With doubling of threads:

GVE-LPA: 1.7x

GVE-Louvain: 1.6x



- Introduction
- Related work
- Approach
- Evaluation
- Conclusion

Conclusion

- Efficient DSA are important for HPC - Hashtable.
    - Utilize parallel primitives where necessary.
    - Minimize repeated memory allocation/deallocation.
    - Consider effects like false sharing.

Acknowledgement and Thanks

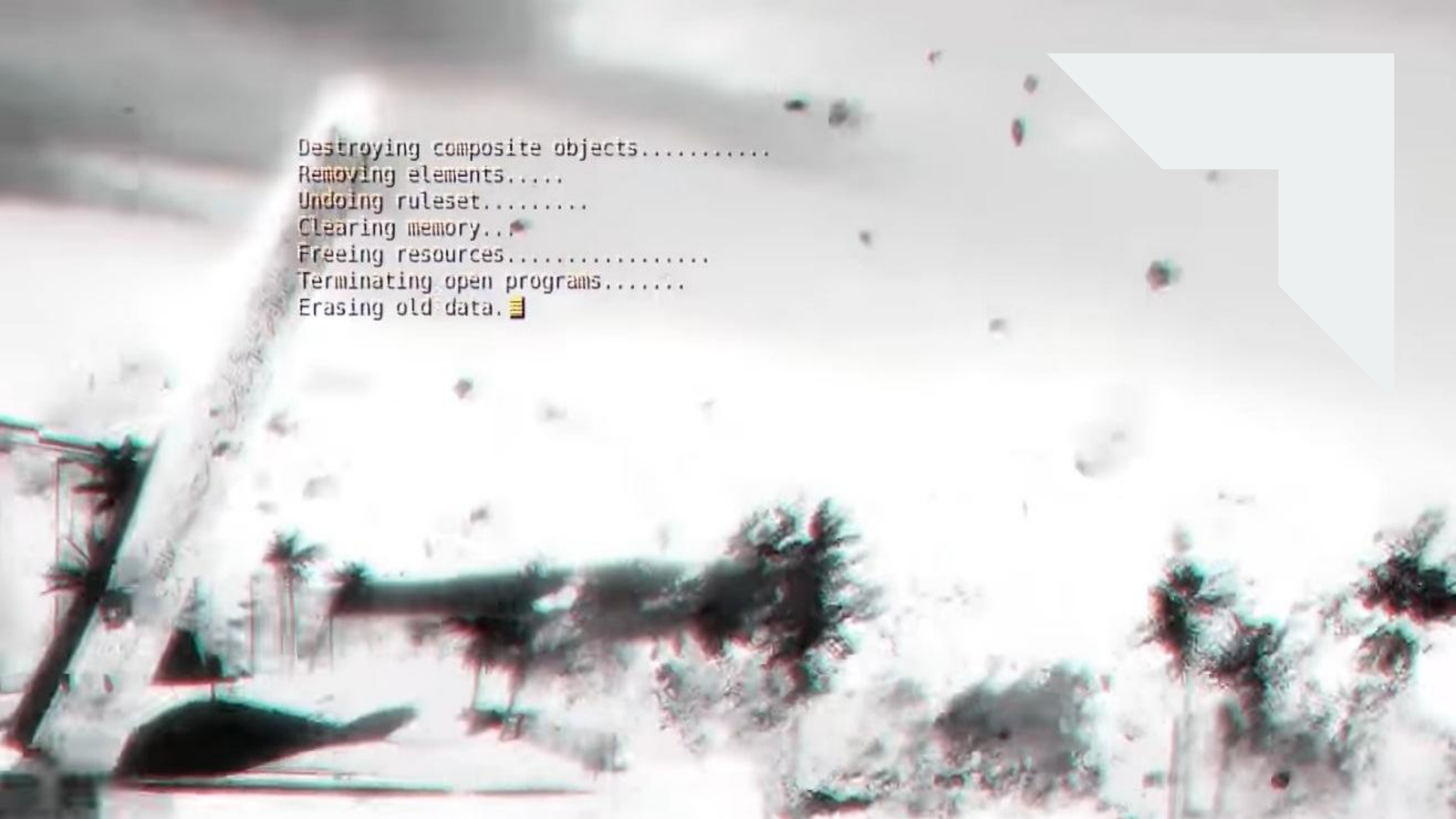
We extend our sincere thanks to HiPES 2024 reviewers, conference chairs, and to you for listening.

Have a memorable day!

Dr. Kishore Kothapalli
IIIT Hyderabad

Dr. Dip Sankar Banerjee
IIT Jodhpur

Source code


Destroying composite objects...........
Removing elements.....
Undoing ruleset.........
Clearing memory...
Freeing resources.................
Terminating open programs.......
Erasing old data.