On the Design of a Framework for Large-Scale Exploratory Graph Analytics

Oliver Andres Alvarado Rodriguez
Doctoral Candidate in Computer Science
New Jersey Institute of Technology, Newark, NJ

PhD Dissertation Advisor
Dr. David A. Bader, Distinguished Professor of Data Science, NJIT

PhD Defense Committee Members
Dr. Ioannis Koutis, Associate Professor of Computer Science, NJIT
Dr. Senjuti Basu Roy, Associate Professor of Computer Science, NJIT
Dr. Shantanu Sharma, Assistant Professor of Computer Science, NJIT
Dr. Zhihui Du, Principal Research Scientist, Department of Data Science, NJIT
Dr. Kamesh Madduri, Associate Professor of Computer Science and Engineering,
The Pennsylvania State University, University Park, PA

March 12, 2025

NJIT
New Jersey Institute of Technology

Table of Contents

Funding Sources

- NSF CCF-2109988
  - “EAGER: High Performance Algorithms for Interactive Data Science at Scale”
    - Principal Investigator: David Bader
- NSF OAC-2402560
  - “Collaborative Research: OAC Core: Cyber-Infrastructure for Community Detection, Extraction, and Search in Large Networks”
    - Collaboration with University of Illinois Urbana-Champaign
      - NJIT Principal Investigator: David Bader
      - UIUC Principal Investigators: Tandy Warnow and George Chacko

NJIT New Jersey Institute of Technology

Main Accomplishments

- This is the first work that properly **democratizes** large-scale exploratory graph analytics for **both** developers and users.
- It is fully **open-source** and **technology transfers** have been successfully completed for data scientists in government and academic researchers at Harvard and University of Illinois Urbana-Champaign.
- All functionality is **competitive** with leading-edge solutions.
- Proof of success has been shown in **various** scientific domains: scientometrics, cybersecurity, and neuroscience.

Problem Statement

- Large-scale graph analytics encounters **three major issues**:
  - It is **fragmented** across various systems and languages where solutions target specific infrastructures—shared-memory ***or*** distributed-memory.
  - There is **no easy way** to run large-scale Python-based graph analytics, current solutions like Spark are **clunky and difficult** to set-up.
  - It is **hard** to parallelize or distribute existing Python graph analytical code and requires added-on salt instead of native language functionality.
- These issues create a gap between HPC and Python-based graph analytics.

Dissertation Statement

Through the design and implementation of **Arachne**, a novel framework for large-scale exploratory graph analytics, the performance gap between HPC and Python-based graph analytics can be bridged.

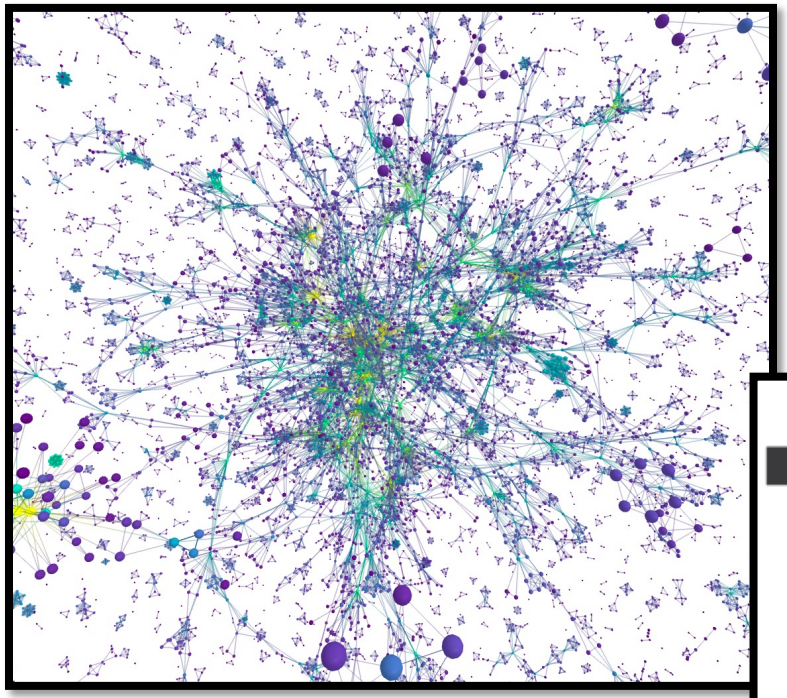
Motivations

Motivation 1: Real-World Data is *Huge*

Area	Name	Type	Size
Scientometrics	OpenAlex	Authorship	50 **GB**
	OpenCitations	Citations	10 **GB**
Cybersecurity	Network Intrusion Data Sets (NIDS)	Network Flows	50 **GB**
Neuroscience	Drosophila FlyWire	Connectome	100 **TB**
	H01	Connectome	1.4 **PB**

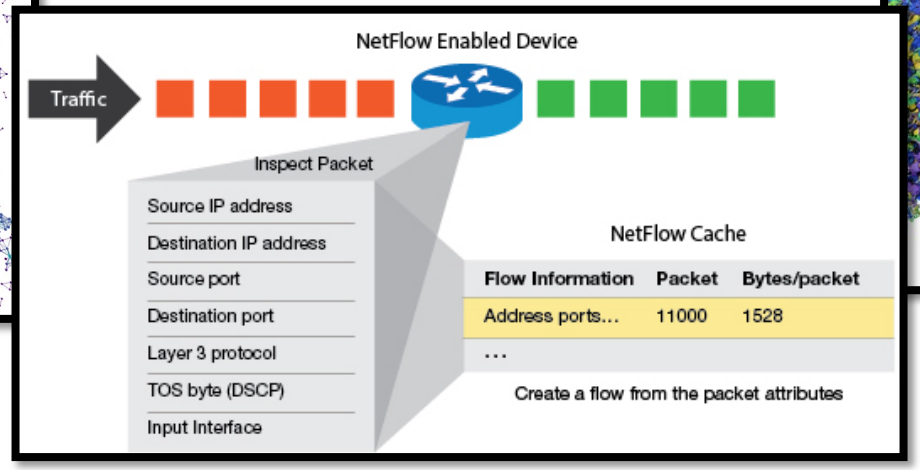
How can we analyze the graphs created from these datasets?

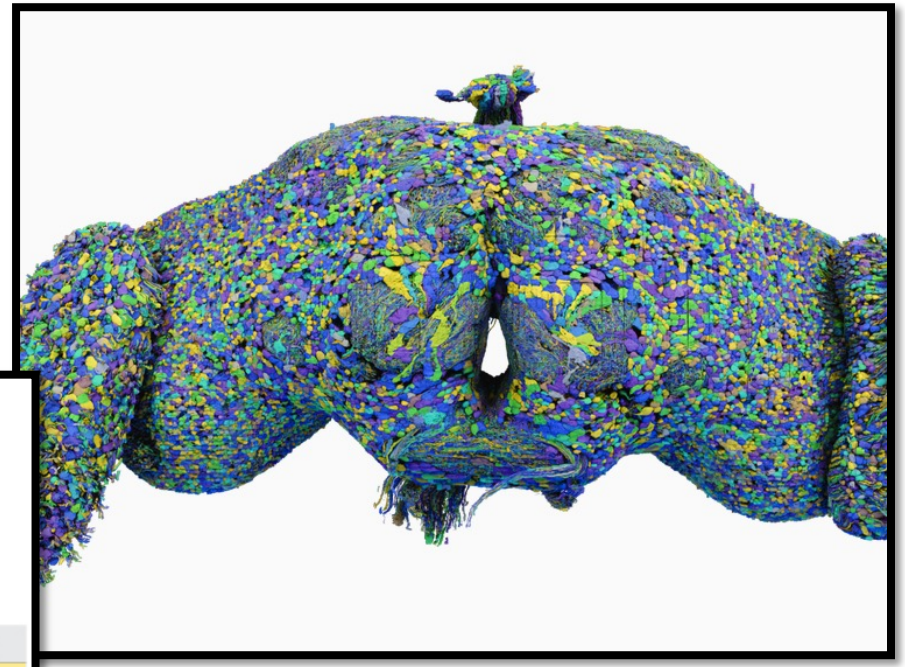
Motivation 2: We ***Need*** Large-Scale Graph Analytics



Finding communities in co-authorship and citation networks (Image credit: [Priem et al. 2022]).

Categorizing packets in NetFlow networks as malicious or benign (Image credit: [Emil 2019]).

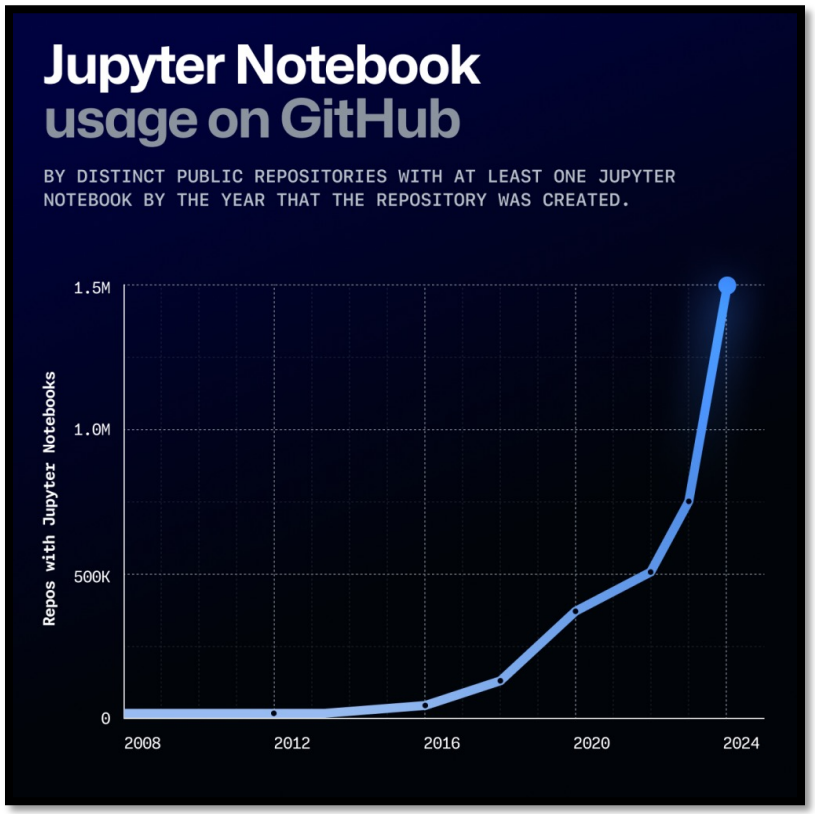


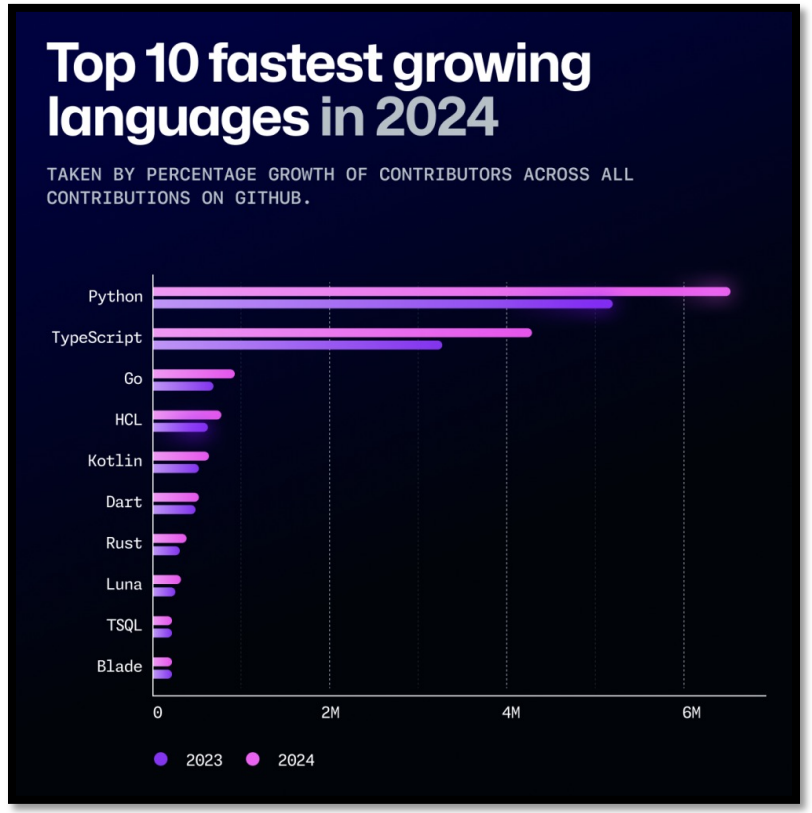


Finding subgraphs within connectomes such as the FlyWire whole-brain Drosophila (fruit fly) connectome (Image credit: [FlyWire 2024]).

New Jersey Institute of Technology

Motivation 3: Python is the *Lingua Franca* of Science





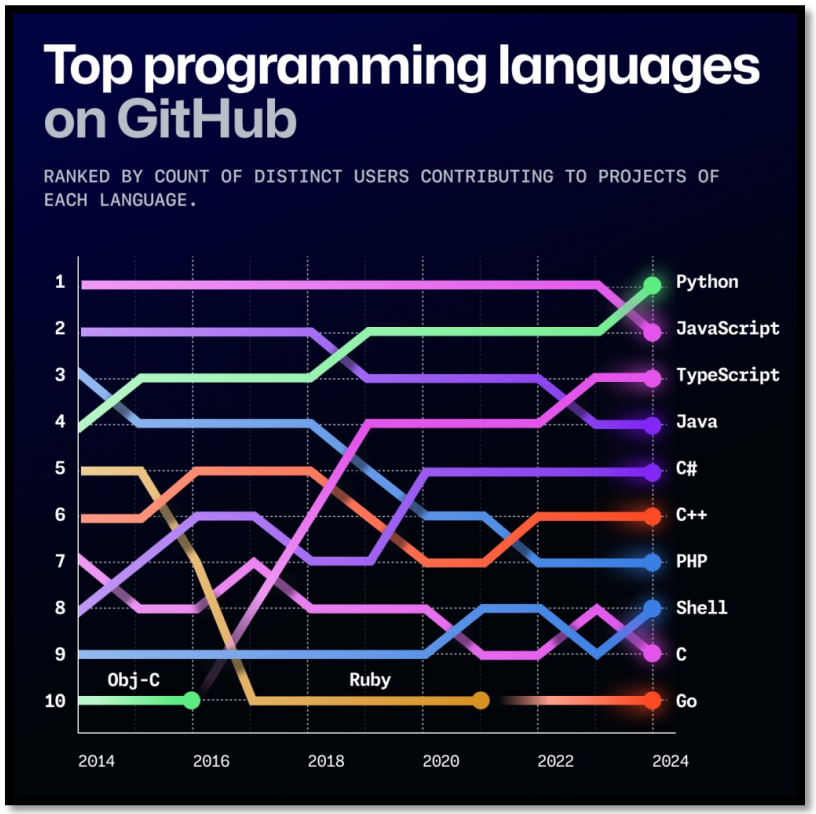
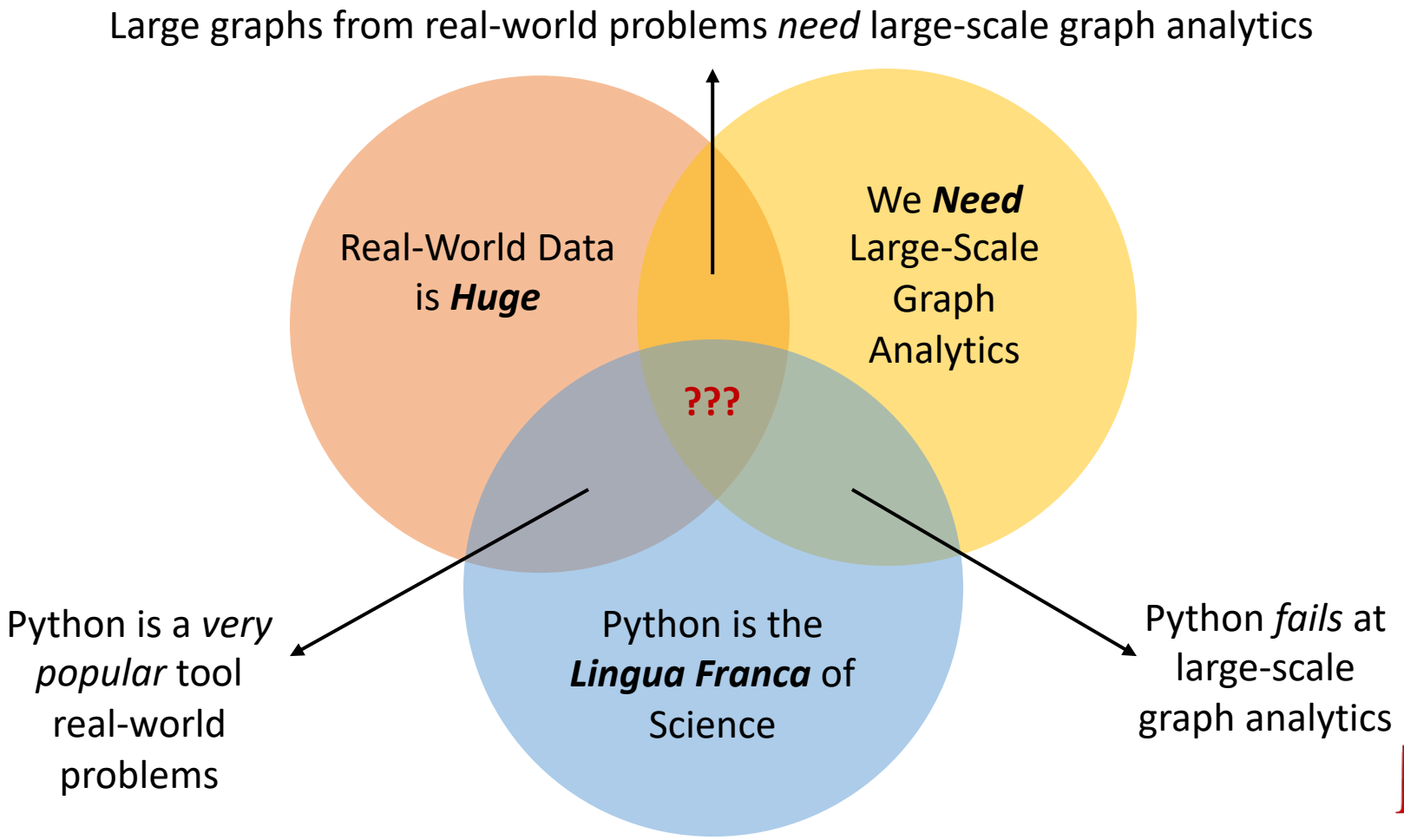


Image Credits: [GitHub Octoverse 2024]

New Jersey Institute of Technology

How Can We Cover This Research Gap?



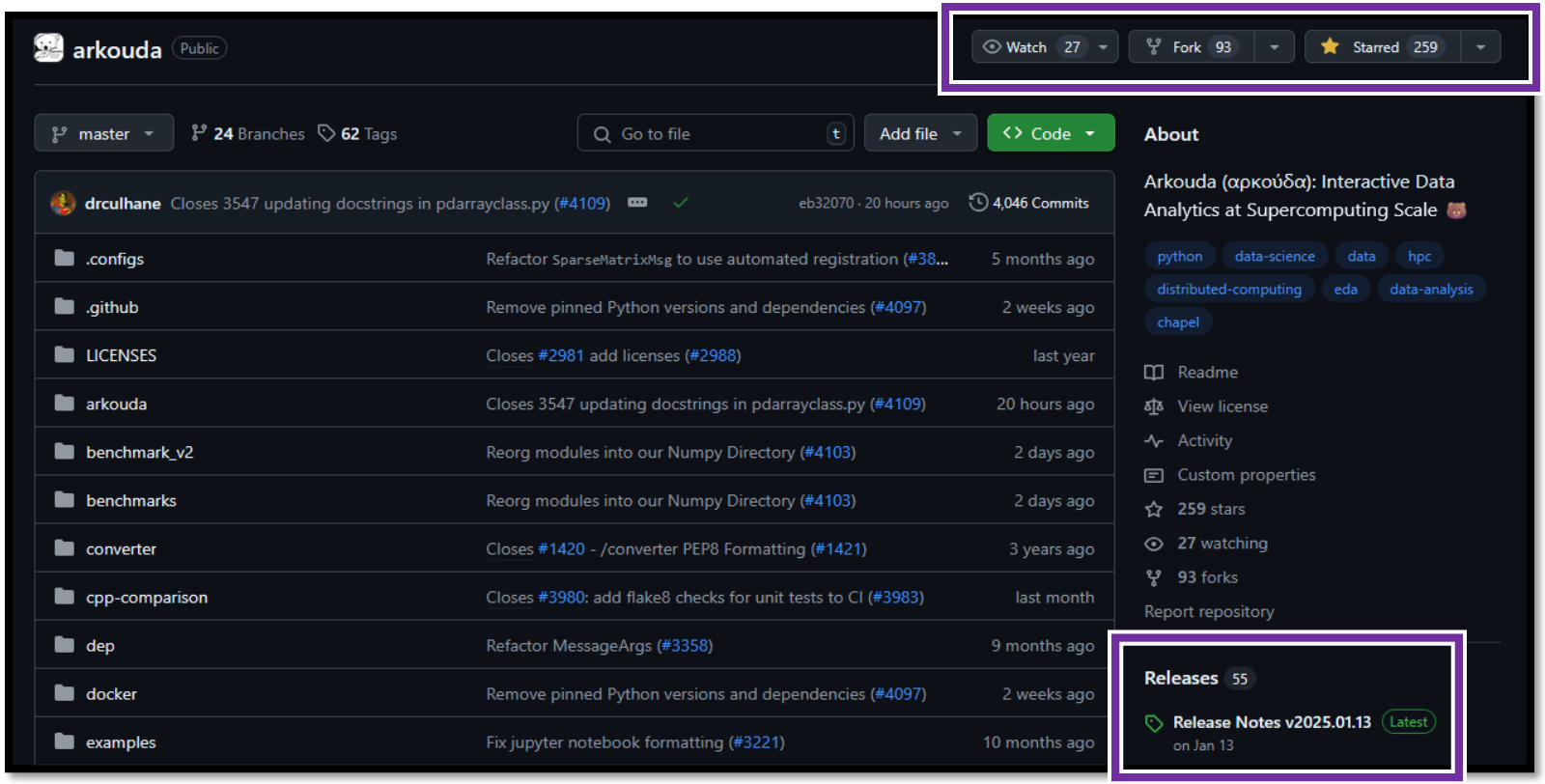
New Jersey Institute of Technology

Arachne to the Rescue!

Arachne: An Open-Source Framework for Large-Scale Exploratory Graph Analytics

- The first work that properly **democratizes** large-scale exploratory graph analytics for **both** developers and users.
- **Efficient** implementations of various graph algorithms, featuring both **traditional** and **novel** techniques for large-scale graph analytics.
  - **Optimized distributed graph querying**, quick input/output across **multiple graph formats**, and **fast** synthetic graph creation.
    - Synthetic graph creation in Arachne takes 20 seconds for an Erdős-Rényi graph with 10 million edges as opposed to NetworkX's **half an hour**.
- Provides **Python-based** exploratory graph analytics at **supercomputing scale**.
  - Built on Chapel, C, and C++. Extends **Arkouda** to support graph analytics.

Arkouda: An Open-Source Framework for Large-Scale Exploratory Data Analytics



[Bears-R-Us 2025]

New Jersey Institute of Technology

A Bird's-Eye View of Arachne and Arkouda

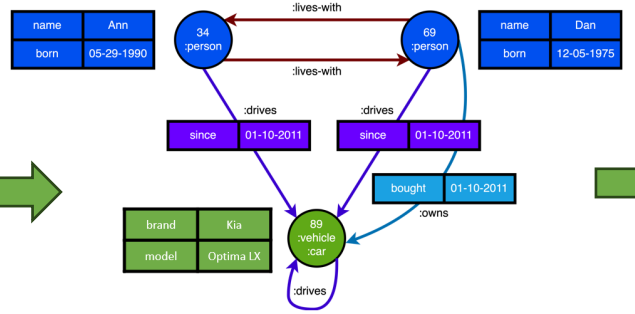
id	label	name	born	brand	model
34	person	Ann	1990	NULL	NULL
69					
89					
89					

src id	dst id	relationship	since	bought
34	69	lives-with	NULL	NULL
69	34	lives-with	NULL	NULL
34	89	drives	2011	NULL
69	89	drives	2011	NULL
69	89	owns	NULL	2011
89	89	drives	NULL	NULL

Load in large CSVs, HDF5s, Parquets, matrix market files, etc.

- read_matrix_market_file()
- add_edges_from()
- rmat()
- gnp()

Convert dataframes to graphs or generate your own synthetic graphs.

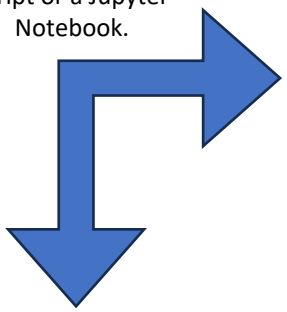


Work with your data as a graph.

- bfs_layers()
- subgraph_isomorphism()
- triangle_counting()
- subgraph_view()

Perform analysis or filter for NetworkX, iGraph, or graph-tool.

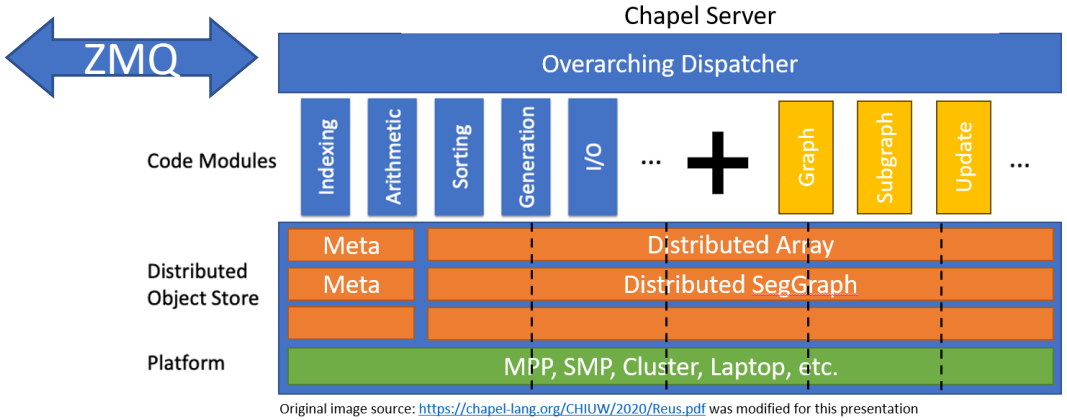
User edits a Python script or a Jupyter Notebook.



User

```python
import arkouda as ak
import arachne as ar

## Get src and dst from input file.

graph = ar.PropGraph()

## Generate label_df and relationships_df from input file.

graph.load_edge_attributes(relationships_df)
graph.load_node_attributes(label_df)

## User generates labels_to_find and relationships_to_find.
returned_nodes = graph.node_attributes["column"] == 1
returned_edges = graph.edge_attributes["column"] == 2

subgraph_src = ak.in1d(returned_edges[0], returned_nodes)
subgraph_dst = ak.in1d(returned_edges[1], returned_nodes)

kept_edges = subgraph_src & subgraph_dst

subgraph_src = subgraph_src[kept_edges]
subgraph_dst = subgraph_dst[kept_edges]

subgraph = ar.Graph()
subgraph.add_edges_from(subgraph_src, subgraph_dst)
## Run some other operations on subgraph!
```

Easily usable through NetworkX-like API.



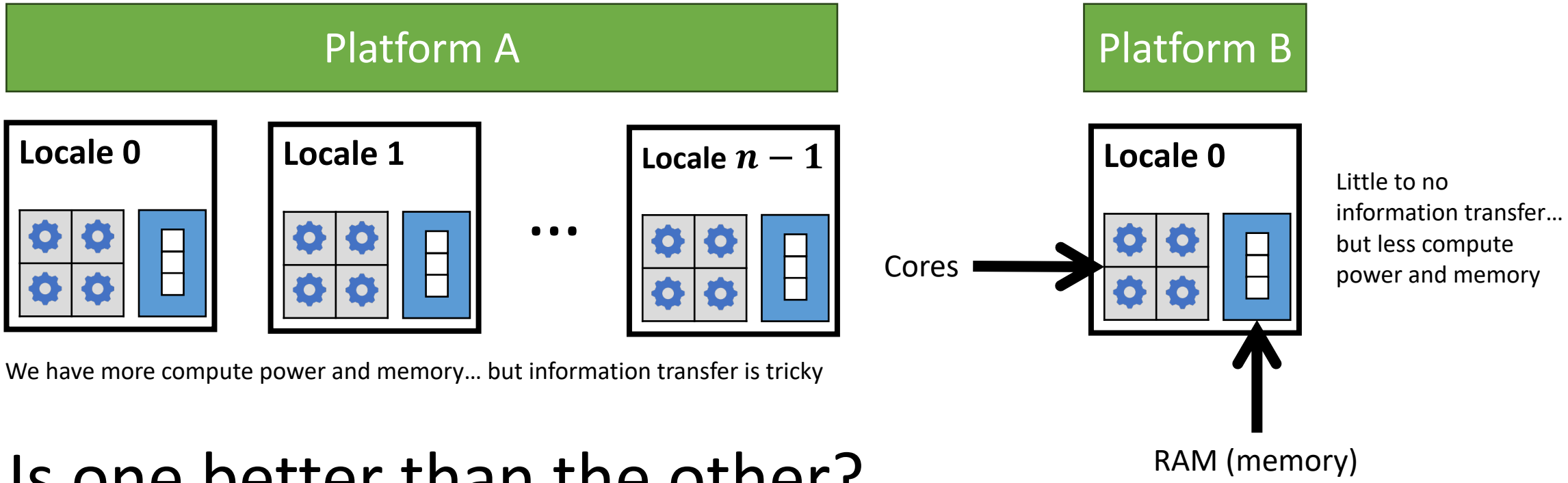
Original image source: https://chapel-lang.org/CHIUW/2020/Reus.pdf was modified for this presentation

Runs on any hardware, data stays in the back-end, user calls API through Python: powerful and productive. (Image credit: [Reus 2020])

OPEN SOURCE: https://github.com/Bears-R-Us/arkouda-njit & https://github.com/Bears-R-Us/arkouda
PAPERS & TALKS AT: IEEE HPEC, IEEE HiPC, IEEE IPDPS, ACM PPoPP, SIAM PP, SC PAW-ATM, ChapelCon & MDPI Algorithms

New Jersey Institute of Technology

Arachne Targeted Infrastructures



We have more compute power and memory… but information transfer is tricky

Is one better than the other?

No? Yes? It depends?

New Jersey Institute of Technology

It Depends!

- Code stemming from **irregular applications** like graph algorithms tend to be easier to code and be more performant when only utilizing **shared-memory**.
- Code stemming from more **regular applications** like array scanning or graph querying is more performant when using **distributed-memory** and increasing the numbers of cores.

How To Guarantee Performance

- We utilize **hybrid** data structures that are **optimally** chosen at **compile-time** based on the executing hardware.
  - ***The user is unaware of this shift***: using multilocale Chapel results in multilocale-efficient code and single locale Chapel results in single locale-efficient code.
- We ensure that **DATA STRUCTURES** and **ALGORITHMS** function effectively on any system.
- ***Every ALGORITHM needs specialized DATA STRUCTURES!***
  - ***There is no "One Ring to Rule Them All."***

Data Structures

Applying Property Graphs to NetFlow Data

IPV4_SRC_ADDR	L4_SRC_PORT	IPV4_DST_ADDR	L4_DST_PORT	PROTOCOL	L7_PROTO	IN_BYTES	OUT_BYTES	IN_PKTS	OUT_PKTS	TCP_FLAGS	FLOW_DURATION_MS	Attack
192.168.100.6	52670	192.168.100.1	53	17	5.21	71	126	1	1	0	4294966	Benign
192.168.100.6	49160	192.168.100.149	444	6	0	217753000	199100	4521	4049	24	4176249	DDoS
192.168.100.46	3456	192.168.100.5	80	17	0	8508021	8918372	9086	9086	0	4175916	Theft
192.168.100.3	80	192.168.100.55	8080	6	7	8442138	9013406	9086	9086	0	4175916	Scout

`ak.stick()`

SOURCE ~ DESTINATION

IPV4 source addresses and ports together make up the source vertex of the edge and respectively the same columns for the destination vertex of the edge.

integer id gen

`ak.GroupBy()`
`ak.groupby.broadcast()`

IPV4_SRC	IPV4_SRC_id	IPV4_DST	IPV4_DST_id
192.168.100.6:52670	3473	192.168.100.1:53	3455
192.168.100.6:49160	4234	192.168.100.149:444	3233

To Arachne!

New Jersey Institute of Technology

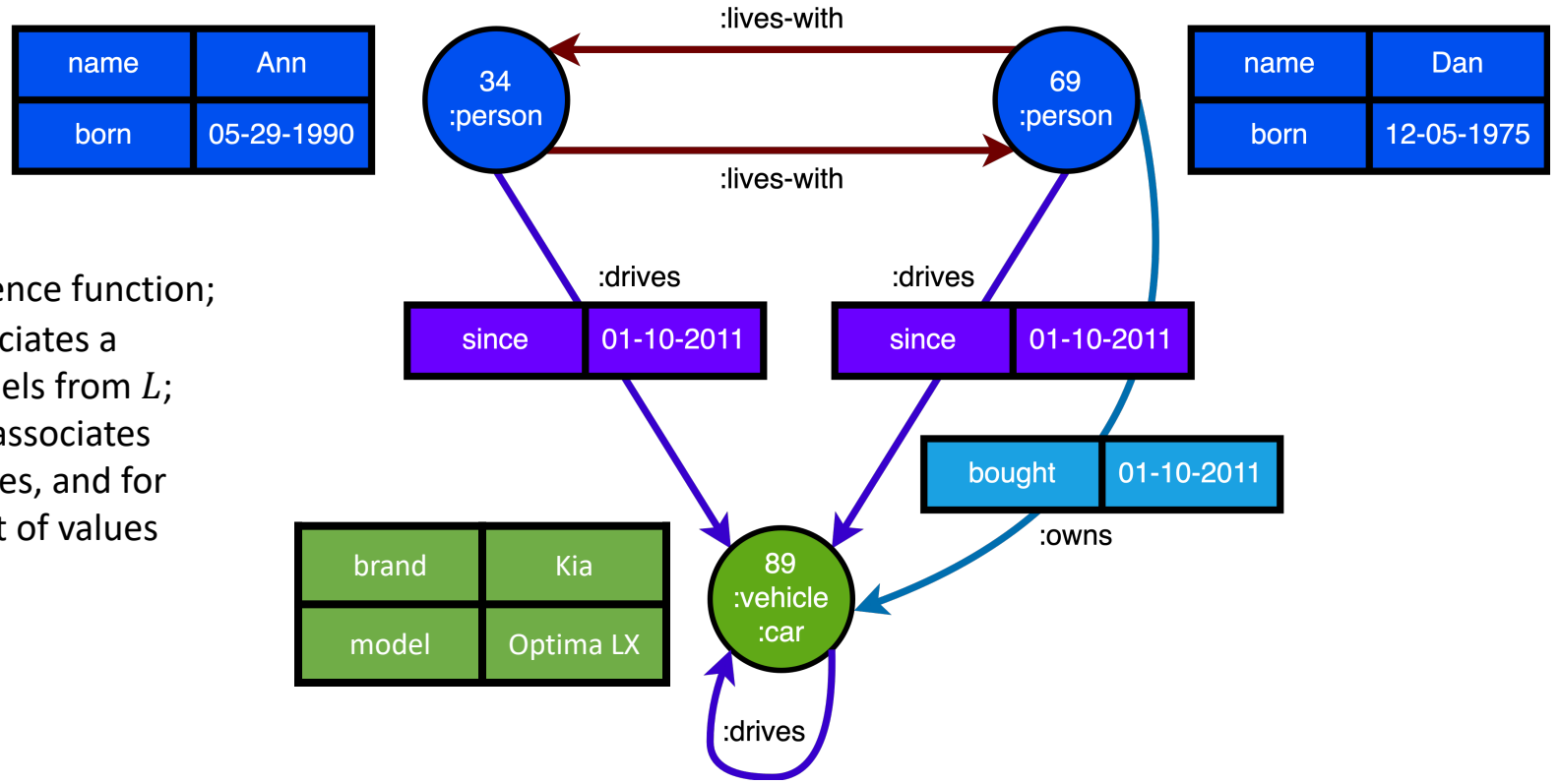
Why Property Graphs?

- A natural way to represent data is through **dataframes** where objects make up rows and their attributes make up columns.
- If two columns can be abstracted out to be **source and destination columns**, then we can convert a dataframe into a graph representation.
- This allows for a richer **non-tabular** way of thinking about the data and lets us **plug in** the property graph into existing graph algorithms.

Property Graph Definitions

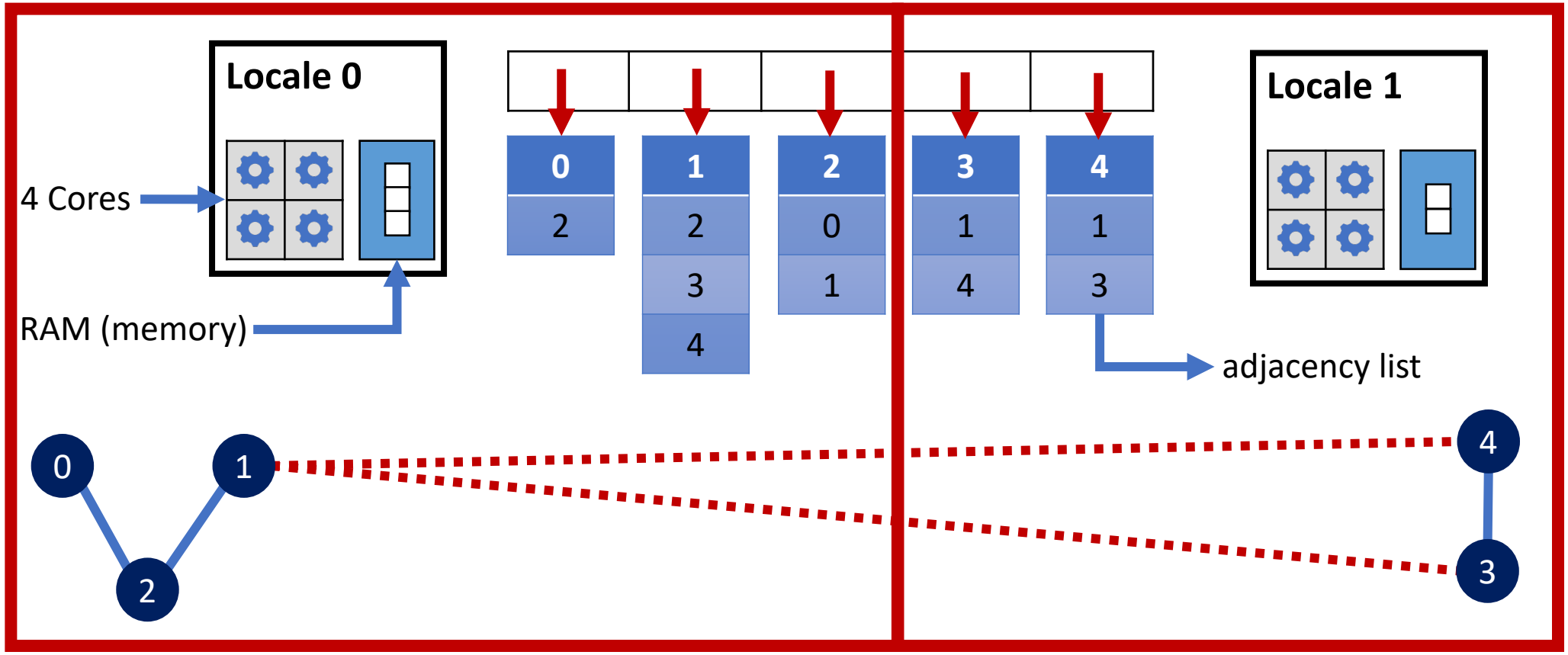
$G_P = (V, E, \rho, \lambda, \sigma)$

- V is a finite set of vertices;
- E is a finite set of edges;
- ρ: $E \rightarrow (V \times V)$ is the incidence function;
- λ: $(V \cup E) \rightarrow SET^{+(L)}$ associates a vertex/edge with a set of labels from L;
- σ: $(V \cup E) \times P \rightarrow SET^{+}(U)$ associates vertices/edges with properties, and for each property it assigns a set of values from U.



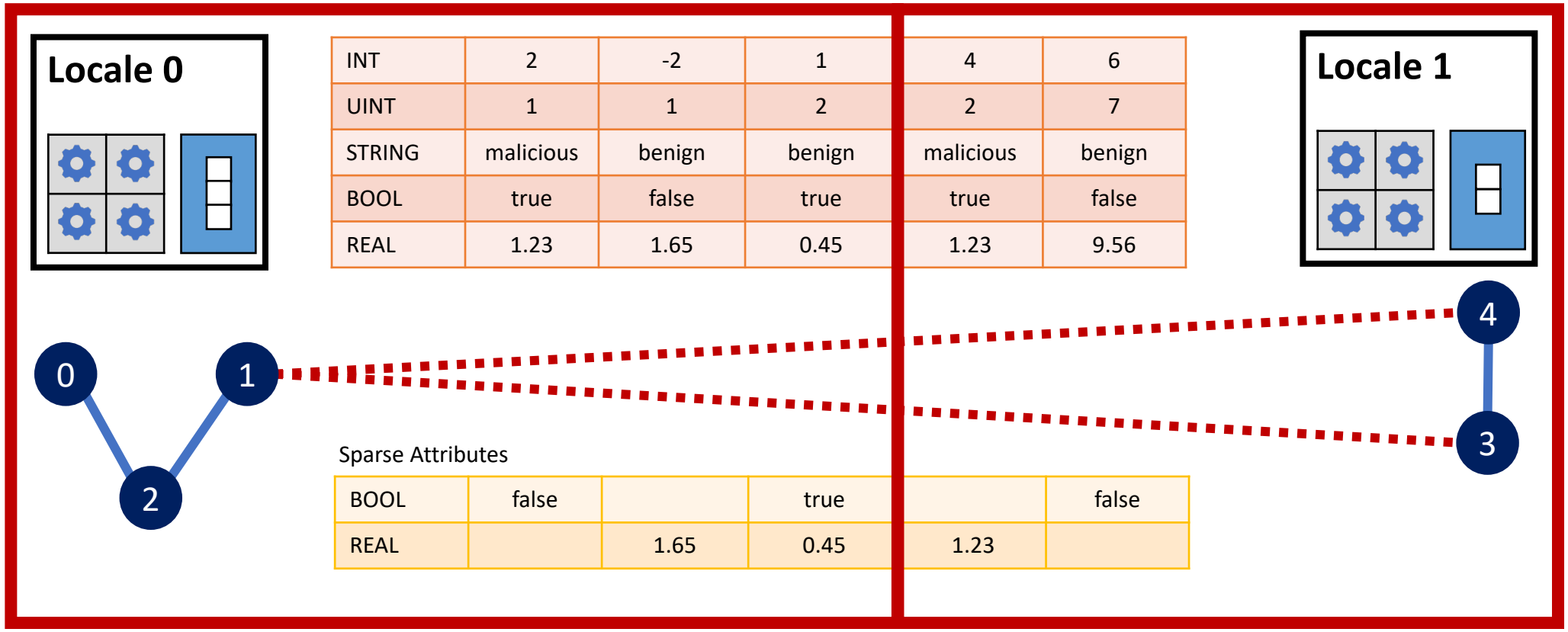
New Jersey Institute of Technology

Vertex-Centric Property Graph Data Structure



New Jersey Institute of Technology

Vertex-Centric Property Graph Data Structure



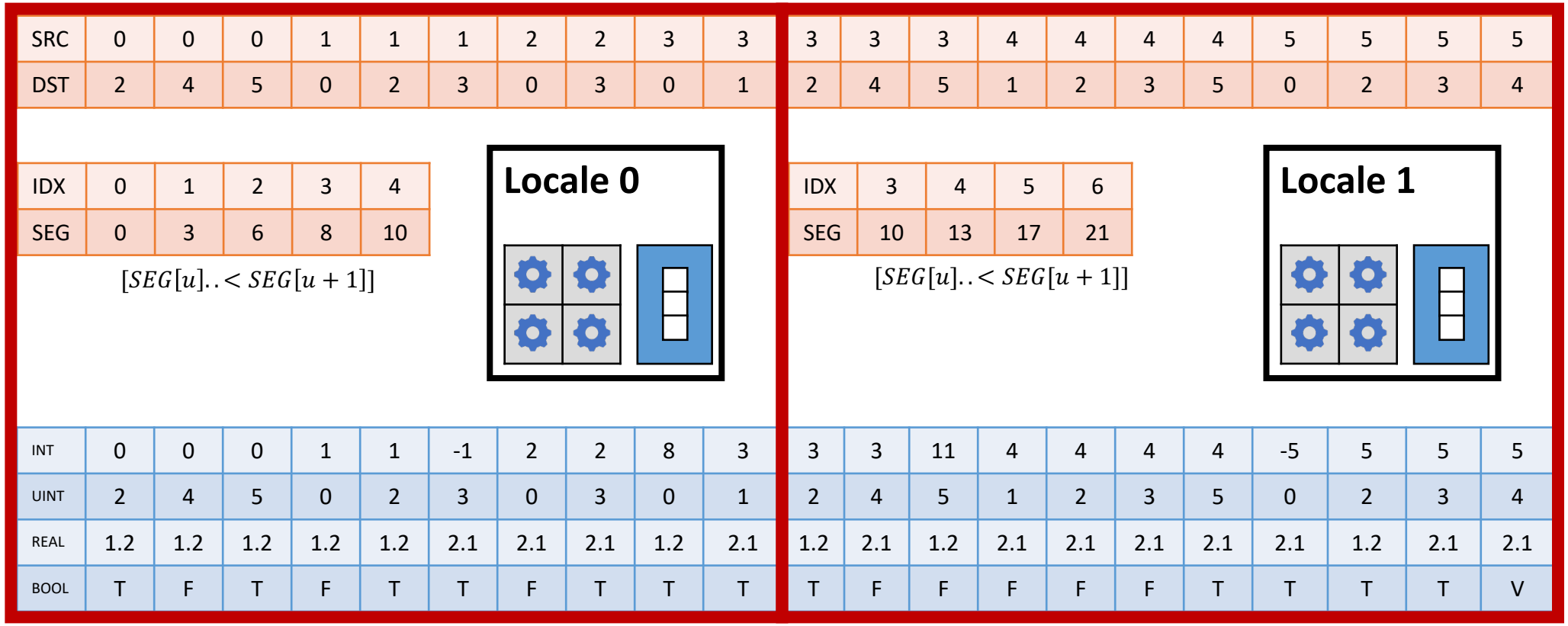
INT	2	-2	1	4	6
UINT	1	1	2	2	7
STRING	malicious	benign	benign	malicious	benign
BOOL	true	false	true	true	false
REAL	1.23	1.65	0.45	1.23	9.56

Sparse Attributes

BOOL	false		true		false
REAL		1.65	0.45	1.23	

New Jersey Institute of Technology

Edge-Centric Property Graph Data Structure



Good for edge-based querying but requires extra overhead for finding neighbors of a given vertex.

Also supports sparse attributes.

Extends and adapts the double-index data structure [Du et al. 2021].

Data Structure Flexibility

- Developers can **use or add** in any representation of a graph to best fit their needs.
- For well-connected components we convert our graph adjacency list representation to a **set representation** to allow for fast union and intersection operations.
- Adding a data structure is easy as it requires only adding a few lines of code to **register** the data structure with the **symbol table** that is used for **object persistence**.

Algorithms

Overview of Graph Algorithms in Arachne

- **Breadth-First Search (BFS)** [Du, Alvarado Rodriguez, Bader 2021]
  - Returns an array of size n with either how many hops away some vertex v is from an initial vertex u or the parent of every vertex in the induced BFS tree.
- **Triangle Counting** [Du, Alvarado Rodriguez, Patchett, Bader 2021]
  - Returns the number of triangles in a graph or the number of triangles that a subset of vertices partakes in.
- **Truss Analytics** [Du, Patchett, Alvarado Rodriguez, Li, Bader 2022]
  - Returns either every edge that partakes in at least $k-2$ triangles, the maximum k value for a graph, or the maximum k for every edge of a graph.
- **Connected Components** [Du, Alvarado Rodriguez, Dindoost, Li, Bader 2023]
  - Returns an array of size n where all vertices that belong to the same component have value x which is the label of the smallest vertex in the component.
- **Subgraph Isomorphism** [Dindoost, Alvarado Rodriguez, Bader 2024]
  - Returns the vertices, edges, or count of the subgraphs in the main graph that match an inputted subgraph.
- **Triangle Centrality** [Patchett, Du, Bader 2022][Patchett 2022]
  - Returns an array of size n with the proportion of triangles centered at a vertex v.
- **Square Counting** [Burkhardt, Harris 2023]
  - Returns the number of squares in a graph or the number of squares that a subset of vertices partake in.
- **Well-Connected Components** [Park, Tabatabaee, Ramavarapu, Liu, Pailodi, Ramachadran, Korobskiy, Ayres, Chacko, Warnow 2024]
  - Determines if the communities of a graph are well-connected or not.

Presentation Format for Each Algorithm

- Why? How? Proof?
  - I will give a real-world example of why an algorithm is useful.
  - I will give a one-slide overview of the key reason(s) behind performance improvements for that algorithm.
  - I will show proof against state-of-the-art or previous versions of the algorithm to show we succeeded at optimizations.
- Three slides total for each algorithm! Please refer to papers for full technical descriptions.

General Experimental Set-Up

Graphs	Type	Usability
Erdős-Rényi (GNP)	Synthetic	SI Benchmarking
Recursive **Mat**rix (RMAT)	Synthetic	Distributed BFS Benchmarking
SNAP	Real	CC, TC, SI Benchmarking
Suite Sparse Matrix Collection	Real	CC, TC, SI Benchmarking
FlyWire	Real	SI Benchmarking
H01	Real	Graph Querying Benchmarking

2x AMD EPYC 7713 CPUs
64 cores per CPU
1TB DDR4 RAM

2x AMD EPYC 7753 CPUs
64 cores per CPU
512GB DDR4 RAM

2x Intel Xeon E5-2650 CPUs
10 cores per CPU
512GB DDR4 RAM

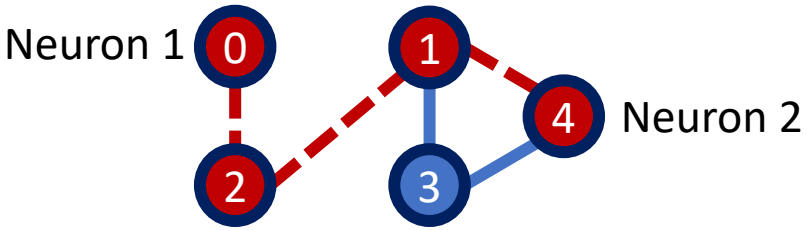
Cray XC System @ HPE
48 cores total
384GB DDR4 RAM

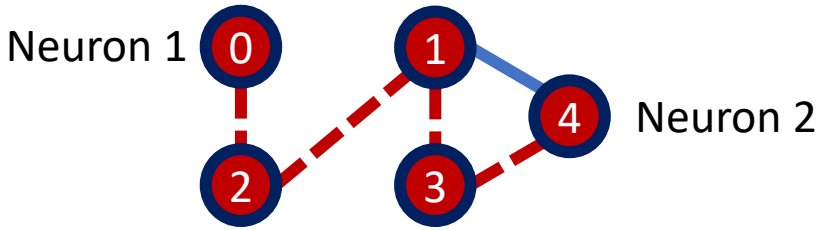
Cray XC system uses a proprietary Aries interconnect.
All others use an Infiniband+GASNet interconnect.

Why Breadth-First Search?

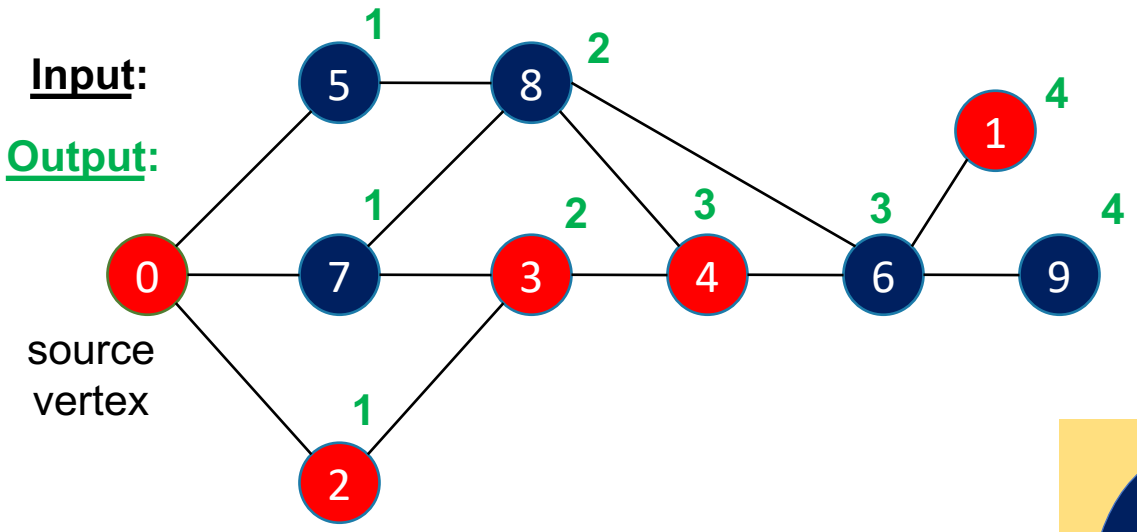
- How many authors away am I from Dijkstra? – **Scientometrics**

- What is the shortest pathway between 2 neurons? – **Neuroscience**

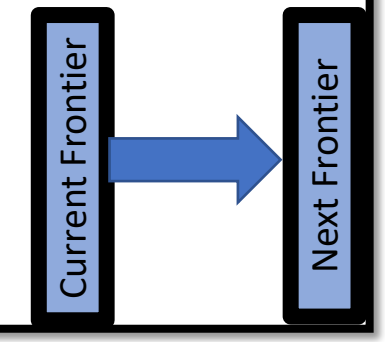




How is BFS Optimized?



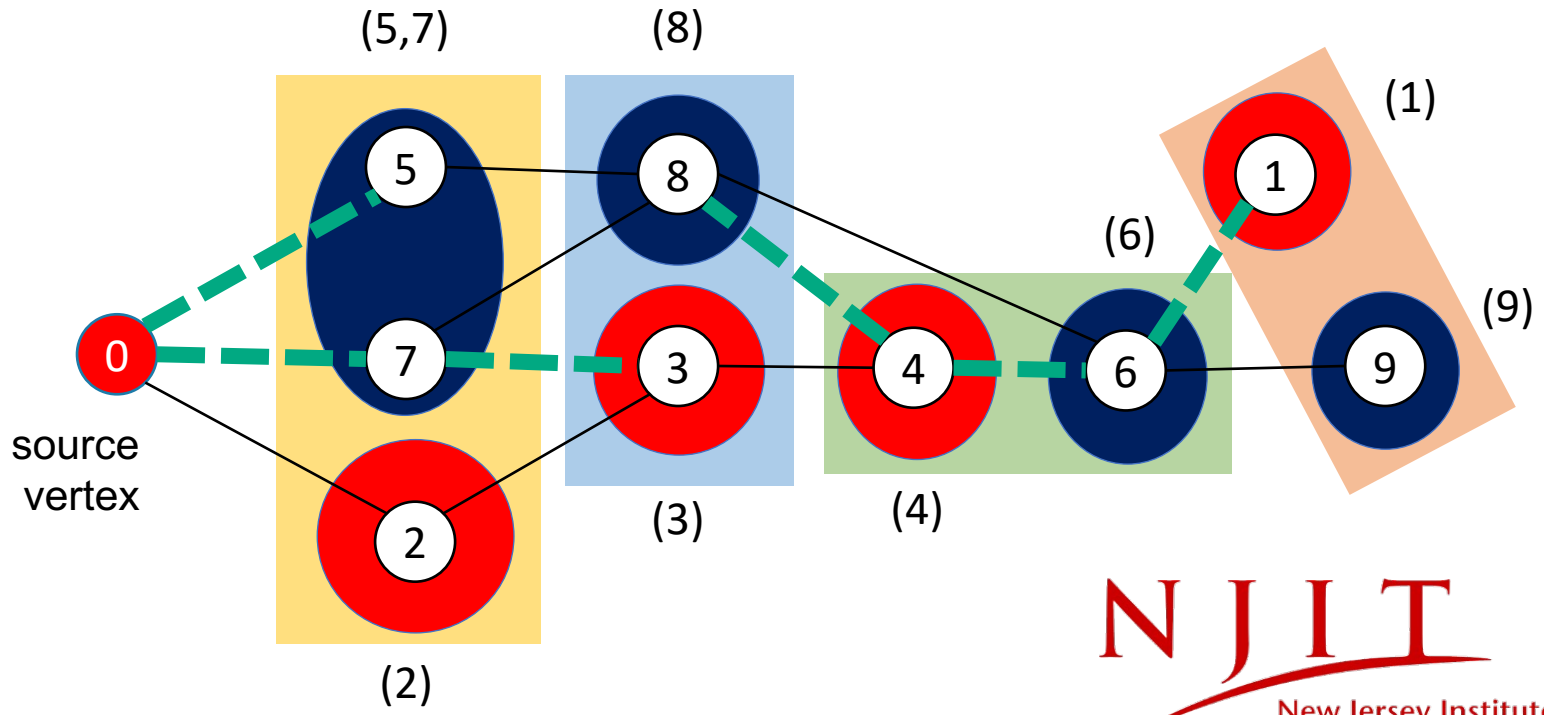

```
9     on Locales[loc] do
10      f ← parentFrontiers[(parentFrontiersIdx + 1) mod 2]
11      for each srcVal in rBuffer.localIter(remBufferPtr, myBufferIdx) do
12        (v, p) ← srcVal
13        if ¬visited.localAccess[v].testAndSet() then
14          parents.localAccess[v] ← p
15          f.pushBack(v)
16        end if
17      end for
18      if freeData then
19        rBuffer.localFree(remBufferPtr)
20      end if
21    end on
```



Outputs/Variants of BFS:

Levels = [0, 4, 1, 2, 3, 1, 3, 1, 2, 4]

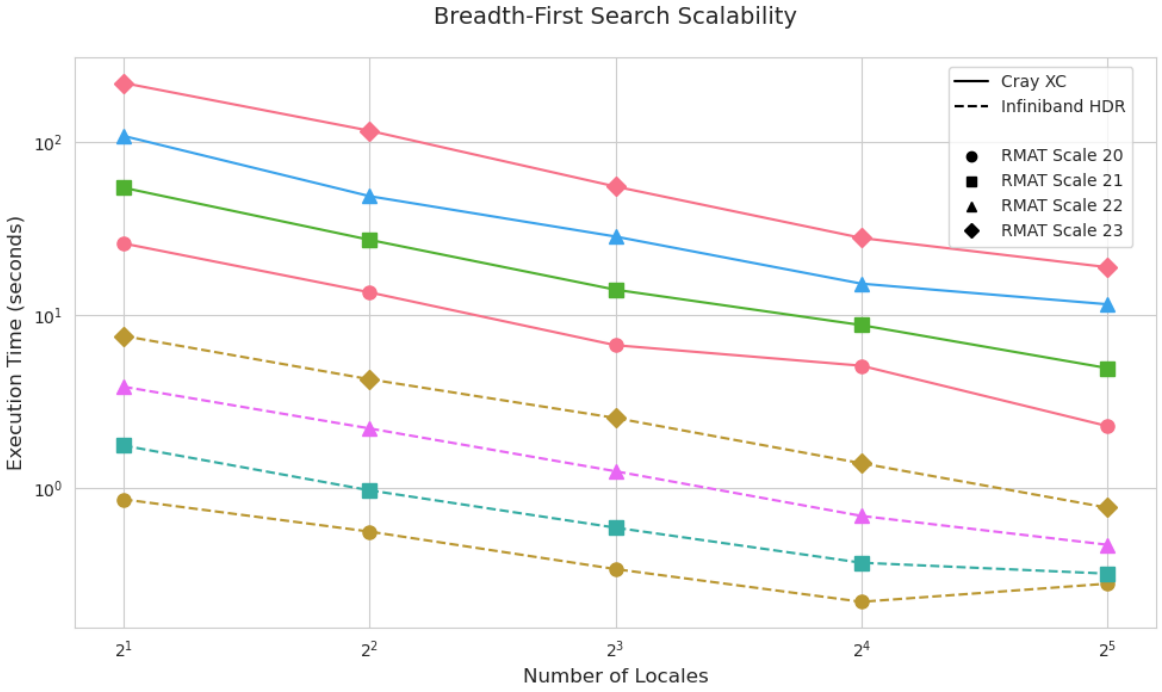
Parents = [0, 6, 0, 7, 8, 0, 4, 0, 5, 6]

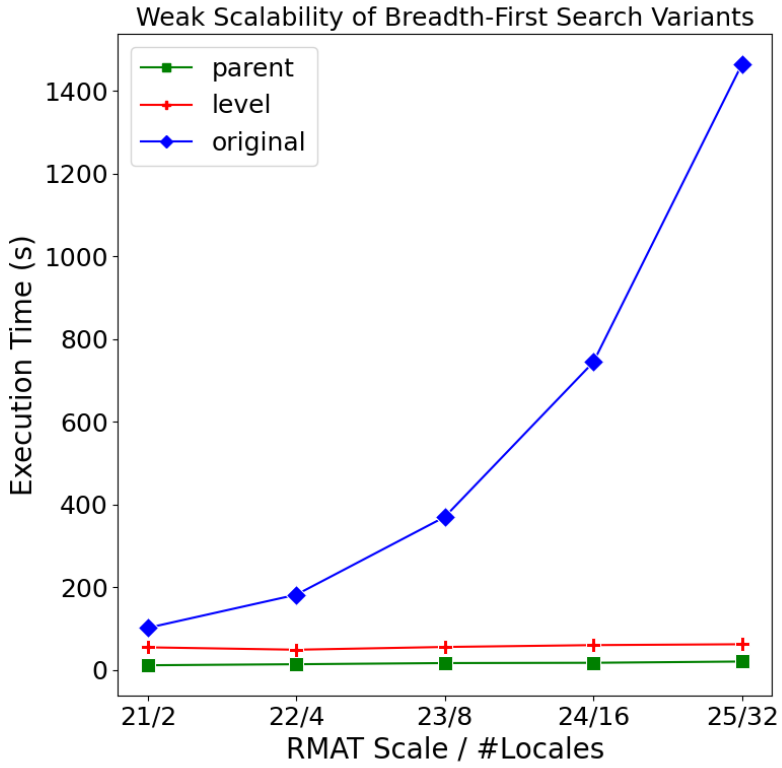


New Jersey Institute of Technology

Optimized BFS Results

Distributed BFS is ***72x*** better than original implementation shown in proposal.





Lower is Better

Distributed BFS shows strong scalability regardless of the type of communication layer.

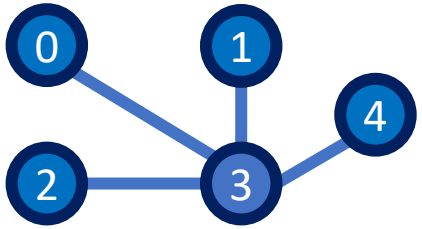
2x AMD EPYC 7713 CPUs
64 processors per CPU
1TB DDR4 RAM

Cray XC System @ HPE
48 cores total
384GB DDR4 RAM

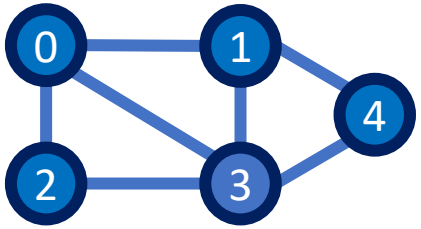
New Jersey Institute of Technology

Why Triangle-Based Analytics?

- Given NetFlow data can we detect botnets? – **Cybersecurity**

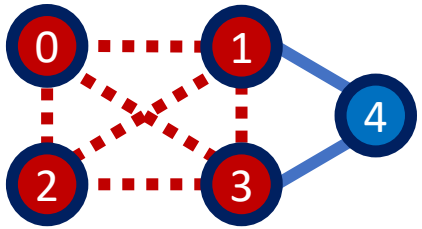


Botnet-like communication



“Regular” communication

- Is an emerging research field noteworthy? – **Scientometrics**



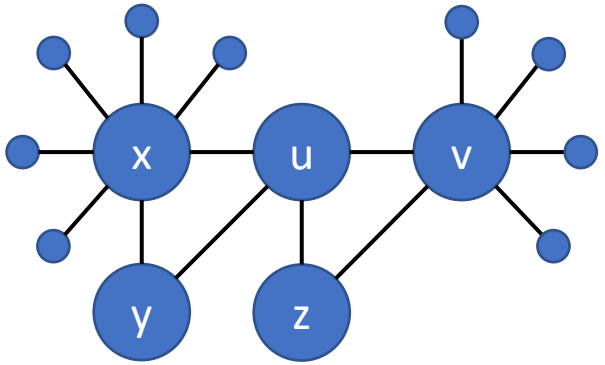
Authors in red form a cohesive k-truss subgraph detailing frequent publishing.

How are Triangle-Based Analytics Optimized?

Given an edge (u, v) we assume that $|Adj(u)| \leq |Adj(v)|$.

Then, for $\{x, y, z\}$ in $Adj(u)$ we check if we can form a complete triangle with (u, v).

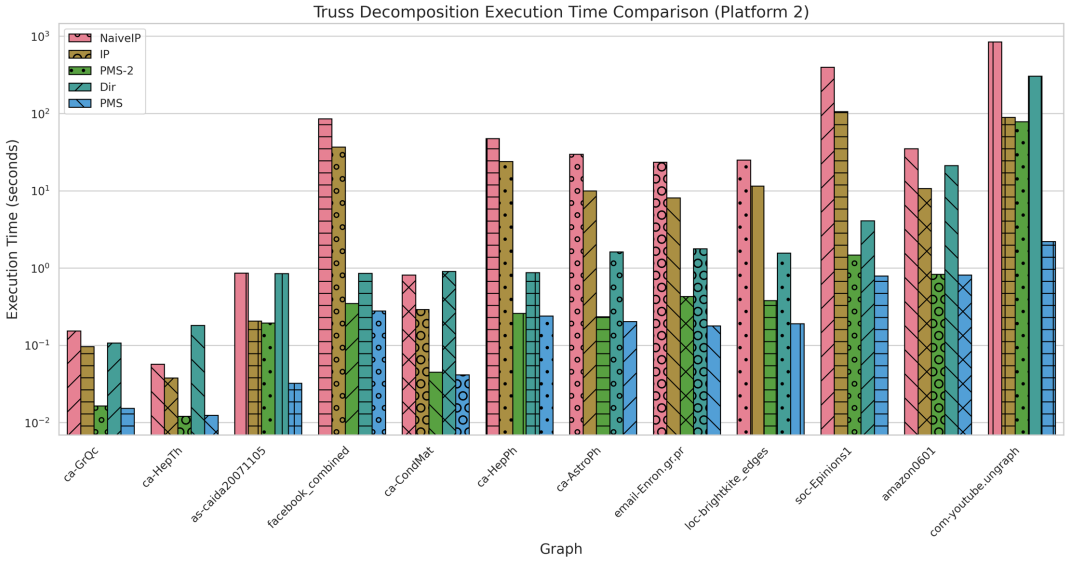
If $|Adj(x, y, z)| < |Adj(v)|$ we will check if $v \in Adj(x, y, z)$, else, we check if $\{x, y, z\} \in Adj(v)$.



$Adj(w)$	Value
$Adj(u)$	4
$Adj(v)$	6
$Adj(x)$	7
$Adj(y)$	2
$Adj(z)$	2

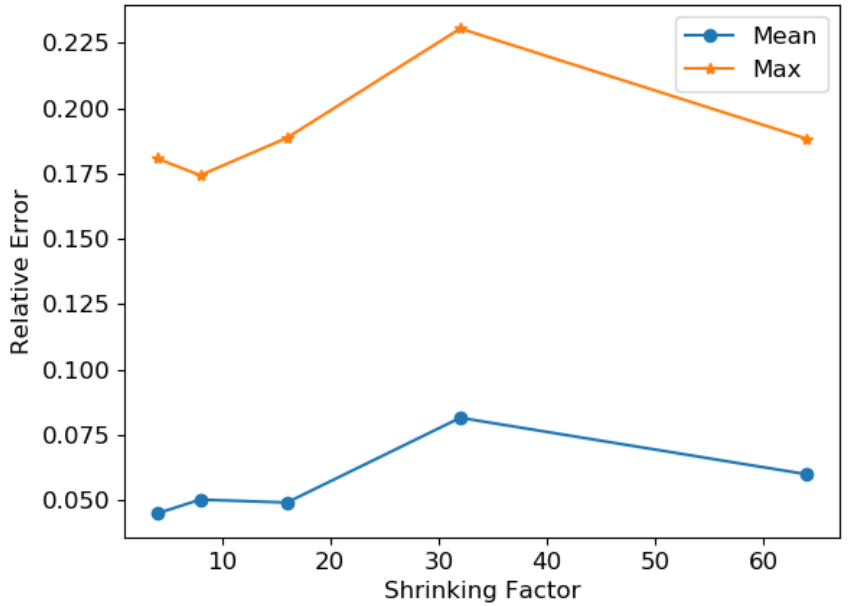
Thread x:	search for x in $Adj(v)$, no match, kill.
Thread y:	search for v in $Adj(y)$, no match, kill.
Thread z:	search for v in $Adj(z)$, match! Increment count.

Optimized Triangle-Based Analytics Results



PMS and PMS-2 for truss decomposition are faster than typical triangle intersection-based methods.

2x AMD EPYC 7713 CPUs
64 processors per CPU
1TB DDR4 RAM



Streaming triangle counting maintains very low relative error.

Lower is Better

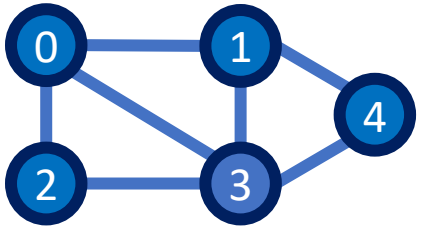
$$TC_N = 0.7264 \times E_H + 1.4157 \times E_M + 1.7529 \times E_T$$
$$TC_P = -0.4464 \times E_{MIN} + 0.1181 \times E_{MEDIAN} + 1.4236 \times E_{MAX}$$

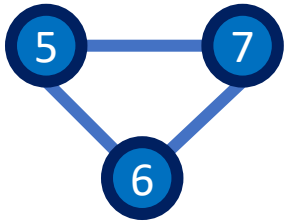
2x Intel Xeon E5-2650 v3 CPUs
10 cores per CPU
512GB DDR4 RAM

New Jersey Institute of Technology

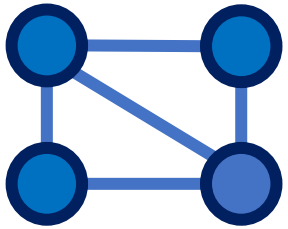
Why Connected Components?

- Do certain authors only publish within their scopes? – **Scientometrics**

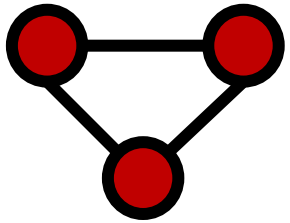




- Can we see disconnected brain regions in EEGs [Vijayalakshmi et al., 2015]? – **Neuroscience**

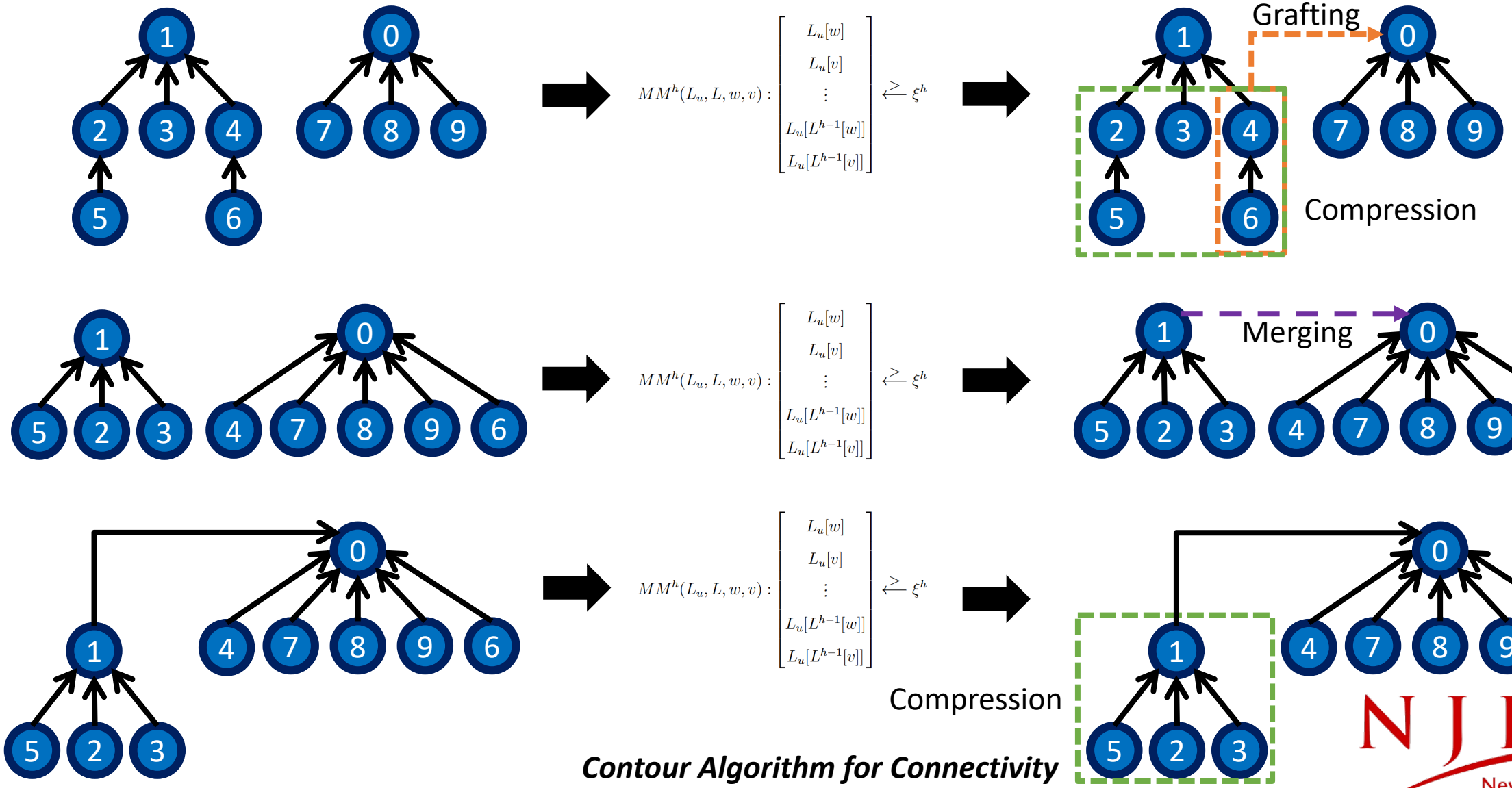


"Happy" neurons



"Sad" neurons

How is Connected Components Optimized?

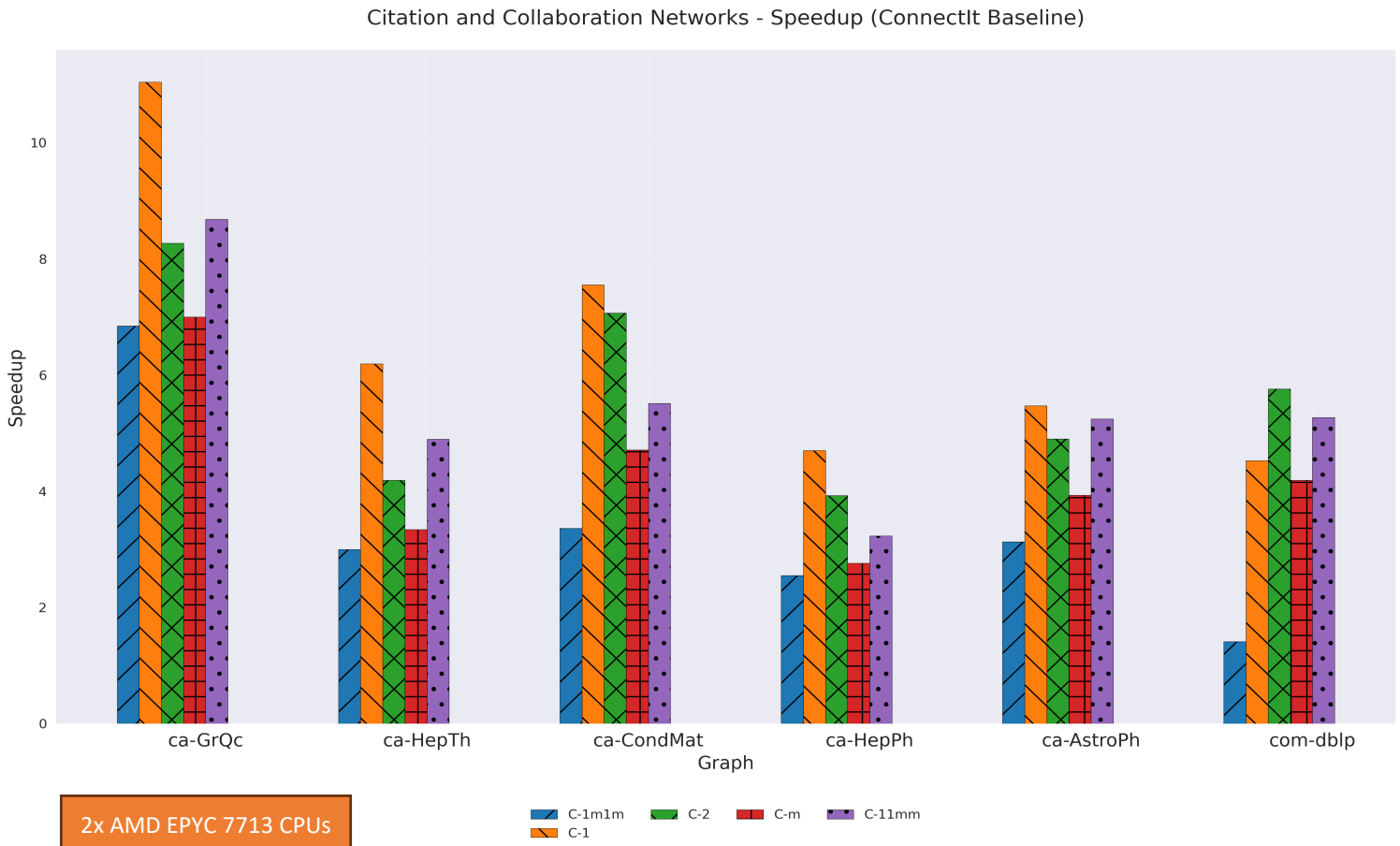


Contour Algorithm for Connectivity

New Jersey Institute of Technology

Optimized Connected Components Results

Contour shows strong speed-up against the state-of-the-art ConnectIt framework.



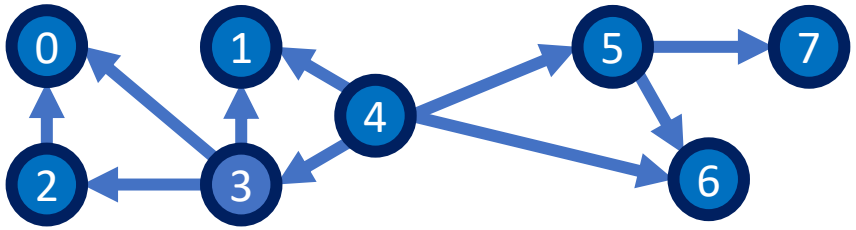
2x AMD EPYC 7713 CPUs
64 processors per CPU
1TB DDR4 RAM

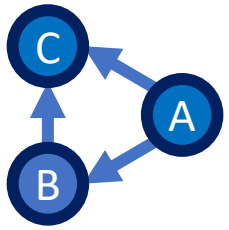
Higher is Better

New Jersey Institute of Technology

Why Subgraph Isomorphism?

- How many occurrences of a structurally significant neural pathway are there? – **Neuroscience**

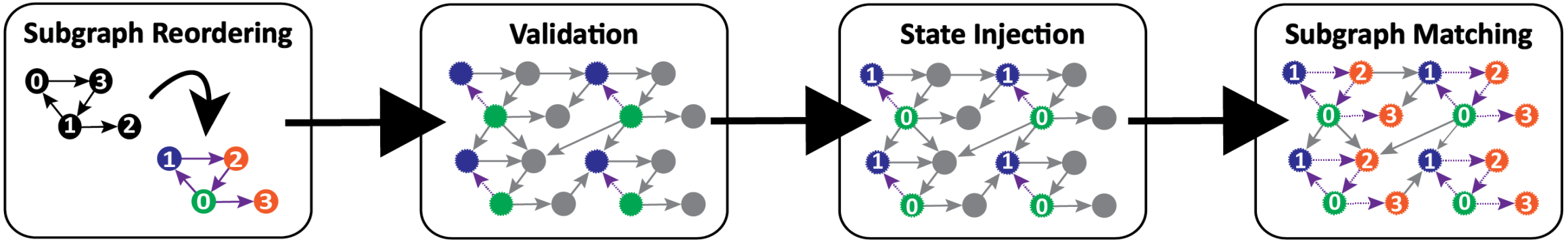




Feed-forward motif

- How many occurrences of a specific shape contain a subset of attributes? – **Neuroscience**, **Scientometrics**, and **Cybersecurity**

How is Subgraph Isomorphism Optimized?



Allows us to reduce the search space by starting off with vertices that are structurally significant by number of out-neighbors.

Allows us to reduce the search space further by allowing the algorithm to dynamically pick initial vertices and edges that are semantically valuable.

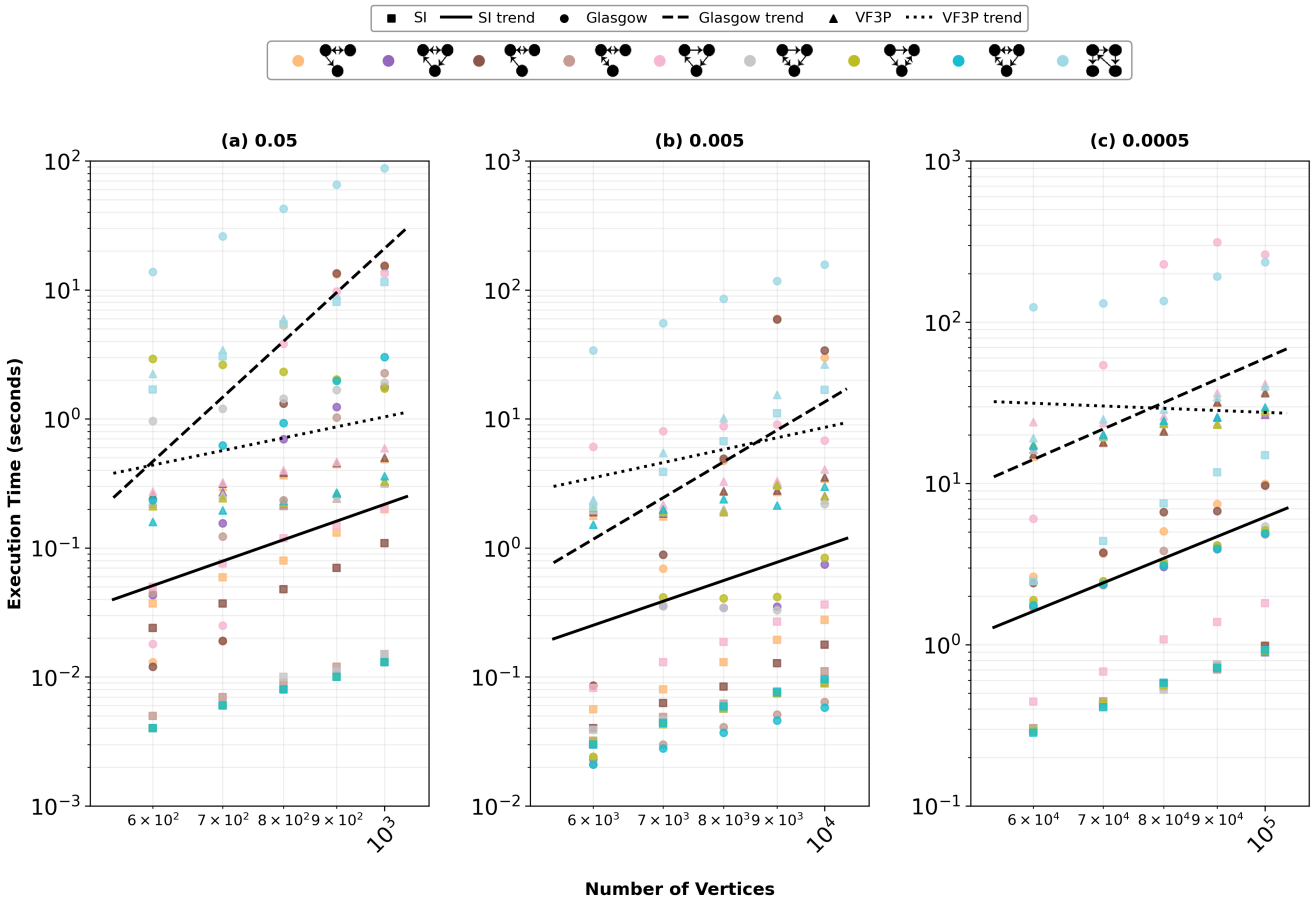
We force subgraph searching to “time travel” and skip unneeded states generated by vertices 0 and 1 of the subgraph and rather start at depth 2.

Match the full subgraphs.

Optimized Subgraph Isomorphism Results

Our SI algorithm is faster than state-of-the-art parallel subgraph searching algorithms on synthetic Erdős-Rényi graphs.

2x AMD EPYC 7713 CPUs
64 processors per CPU
1TB DDR4 RAM



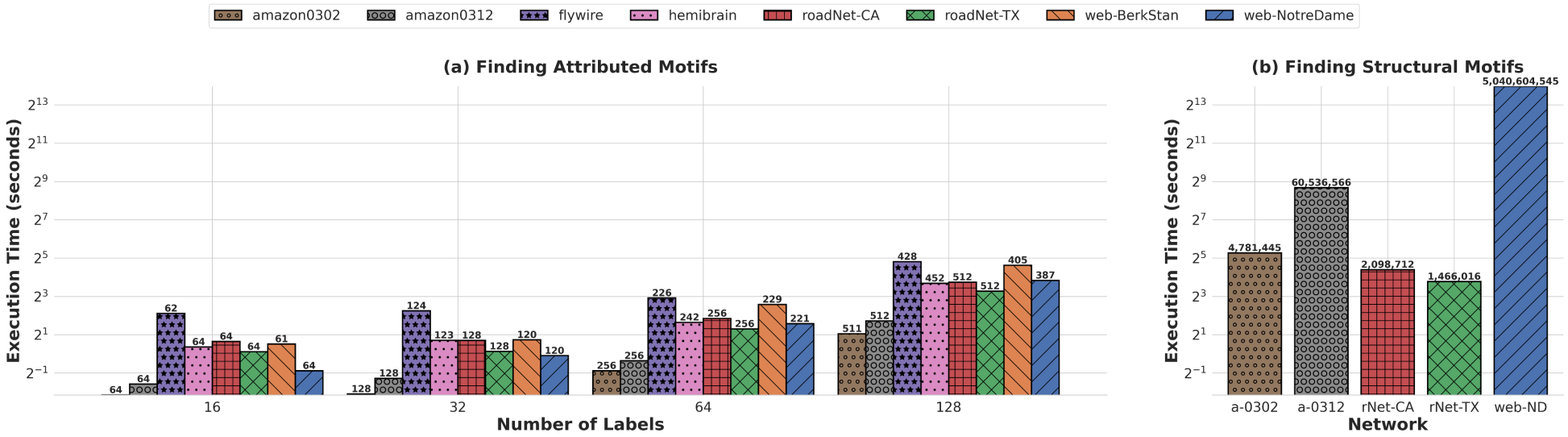
Lower is Better

New Jersey Institute of Technology

Data Injection

Novel data injection benchmark shows significant speed-ups for finding the rarest of subgraphs.

Lower is Better



From **billions** of subgraphs **to a few hundred**.

2x AMD EPYC 7713 CPUs
64 processors per CPU
1TB DDR4 RAM

New Jersey Institute of Technology

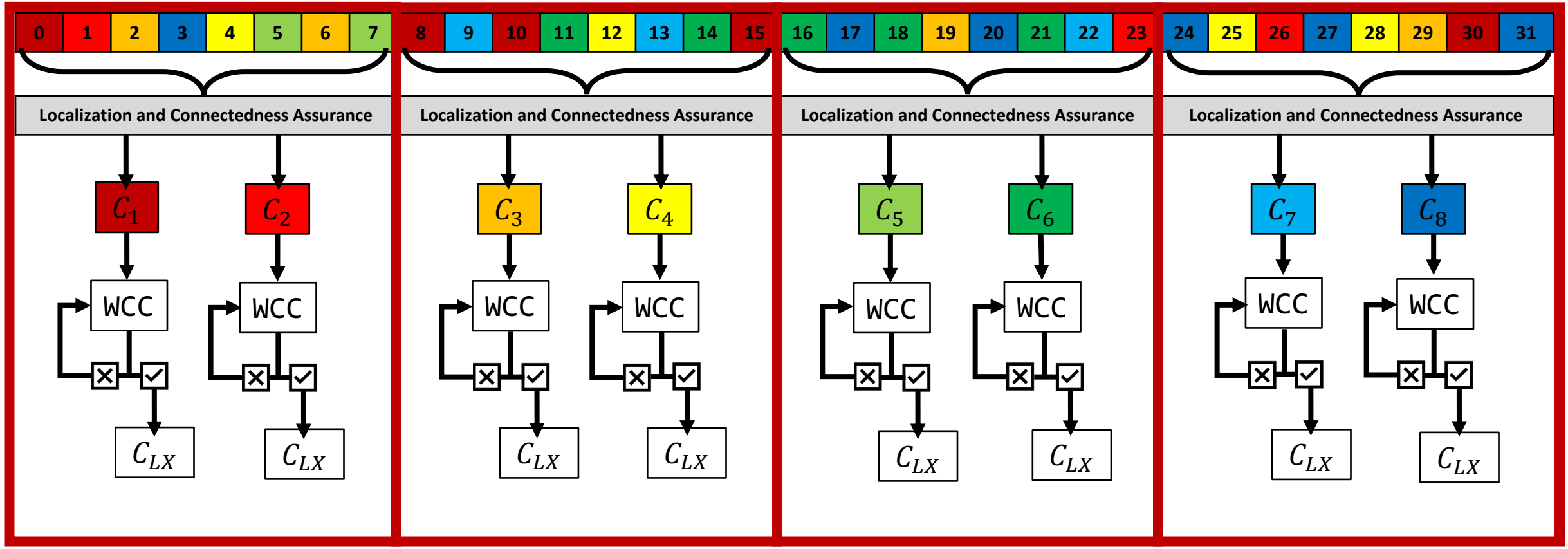
Framework Design

The Well-Connected Components Problem

- This is **very recent work**, started August 2024, that has the objective of creating cyberinfrastructure for community detection, extraction, and search.
  - Collaborative research with University of Illinois Urbana-Champaign.
    - Tandy Warnow, George Chacko, Minhyuk Park, and Vikram Ramavarapu.
- In community detection there are minimal to no guarantees on the **well-connectedness** of each cluster. What does it even mean to have a well-connected community?
  - Our UIUC collaborators designed a metric based on comparing the **minimum cut** of a cluster graph against some given criterion function.
  - If $cut(G_C) > f(n)$ then a cluster is well-connected.
  - WCC can be used **directly** or via a **clustering modification** pipeline.

New Jersey Institute of Technology

The Well-Connected Components Algorithm



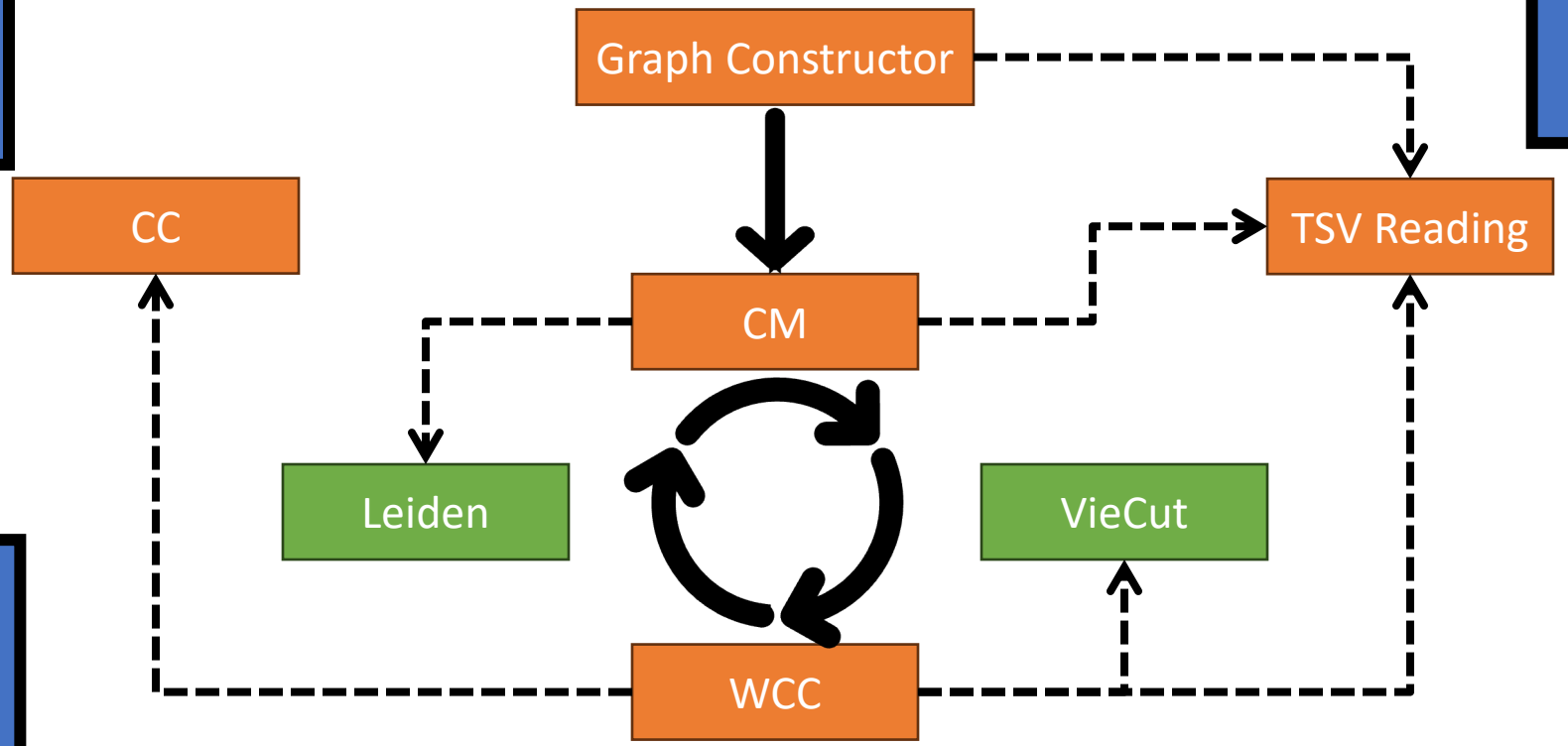
☒ Split cluster into two according to the minimum cut and given criterion function, and recursively check well-connectedness again

☑ Save well-connected cluster with new label consisting of locale identifier (L) and local atomic counter (X)

How Arachne Supports Community Analysis

Fast shared-memory contour algorithm for connected components utilized

Fast parallel and distributed file reading via Chapel serializers



Calling out to Leiden and VieCut optimized for minimal memory allocation

New Jersey Institute of Technology

Summary

Conclusions

- This is the first work that properly **democratizes** large-scale exploratory graph analytics for **both** developers and users.
- It is fully **open-source** and **technology transfers** have been successfully completed for data scientists in government and academic researchers at Harvard and University of Illinois Urbana-Champaign.
- All functionality is **competitive** with leading-edge solutions.
- Proof of success has been shown in **various** scientific domains: scientometrics, cybersecurity, and neuroscience.

Future Work

- Expanding the number of graph analytical kernels within Arachne.
  - **Mohammad Dindoost** is working on algorithms for motif census, enhanced connectivity metrics, and clustering modification.
  - **Fuhuan Li** is working on using Arachne for large-scale fraud detection.
- **Refine** existing algorithms as needed to improve performance in shared-memory or distributed-memory.
- Add new functionality to allow for **Cypher queries** on the property graph data structure.

Publications and Presentations

1. Mohammad Dindoost, Oliver Alvarado Rodriguez, Sounak Bagchi, Palina Pauliuchenka, Zhihui Du, and David A. Bader, “VF2-PS: Parallel and Scalable Subgraph Monomorphism in Arachne,” 2024 IEEE High Performance Extreme Computing Conference (HPEC), 2024.
2. Oliver Alvarado Rodriguez, Zhihui Du, and David A. Bader, “Arachne: A Productive Massive-Scale Graph Analytics Framework,” 2024 SIAM Conference on Parallel Processing for Scientific Computing (SIAM-PP), 2024.
3. Jurgen Kritschgau, Daniel Kaiser, Oliver Alvarado Rodriguez, Ilya Amburg, Jessalyn Bolkema, Thomas Grubb, Fangfei Lan, Sepideh Maleki, Phil Chodrow, and Bill Kay, “Community Detection in Hypergraphs via Mutual Information Maximization,” Nature Scientific Reports, 2024.
4. Zhihui Du, Oliver Alvarado Rodriguez, Fuhuan Li, Mohammad Dindoost, and David A. Bader, “Contour Algorithm for Connectivity,” 2023 IEEE High Performance Computing, Data, and Analytics (HiPC), 2023.
5. Oliver Alvarado Rodriguez, Zhihui Du, and David A. Bader, “Arachne: High-Performance Algorithms and Software for Large-Scale Graph Analytics,” 2023 SIAM New York-New Jersey-Pennsylvania (SIAM-NNP) Section Annual Meeting, 2023.
6. Oliver Alvarado Rodriguez, Fernando Vera Buschmann, Zhihui Du, and David A. Bader, “Property Graphs in Arachne,” 2023 IEEE High Performance Extreme Computing Conference (HPEC), 2023.
7. David A. Bader, Fuhuan Li, Anya Ganeshan, Ahmet Gundogdu, Jason Lew, Oliver Alvarado Rodriguez, and Zhihui Du, “Triangle Counting Through Cover-Edges,” 2023 IEEE High Performance Extreme Computing Conference (HPEC), 2023.
8. Oliver Alvarado Rodriguez, Naren Khatwani, Zhihui Du, and David A. Bader, “Interactive Large-Scale Data and Graph Analytics,” 2023 ACM Principles and Practice of Parallel Programming Conference (PPoPP), 2023. ***Tutorial.***
9. Zhihui Du, Oliver Alvarado Rodriguez, Fuhuan Li, and David A, Bader, “High-Performance Truss Analytics in Arkouda,” 2022 IEEE High Performance Computing, Data, and Analytics (HiPC), 2022.
10. Oliver Alvarado Rodriguez, Zhihui Du, Joseph T. Patchett, Fuhuan Li, and David A. Bader, “Arachne: An Arkouda package for Large-Scale Graph Analytics,” 2022 IEEE High Performance Extreme Computing Conference (HPEC), 2022.
11. Zhihui Du, Oliver Alvarado Rodriguez, Joseph T. Patchett, Fuhuan Li, and David A. Bader, “Enabling Exploratory Large Scale Graph Analytics through Arkouda,” 2021 IEEE High Performance Extreme Computing Conference (HPEC), 2021.
12. Zhihui Du, Oliver Alvarado Rodriguez, and David A. Bader, “Large Scale String Analytics In Arkouda,” 2021 IEEE High Performance Extreme Computing Conference (HPEC), 2021.
13. Zhihui Du, Oliver Alvarado Rodriguez, Joseph T. Patchett, Fuhuan Li, and David A. Bader, “Interactive Graph Stream Analytics in Arkouda,” Journal of Algorithms, 2021.

Works-In-Progress

1. "State Injection: Enhancing Parallel Subgraph Isomorphism for Graphs with Multiple Attributes and Billions of Motifs"
   - Under review at the 23rd Symposium on Experimental Algorithms (SEA 2025)
   - Mohammad Dindoost, Oliver Alvarado Rodriguez, and David A. Bader
2. "Interactive Neuron Topology-Aware Motif Analysis in Large Connectome Graphs"
   - Finalizing submission for IEEE VIS 2025
   - Michael Shewarega, Jakob Troidl, Oliver Alvarado Rodriguez, Mohammad Dindoost, Phillip Harth, Hannah Haberkern, Johannes Stegmaier, David Bader, and Hanspeter Pfister
     - Harvard University, RWTH Aachen, NJIT, Zuse Institute Berling, and University of Würzburg.
3. "On the Parallelization and Distribution of the Well-Connected Components Algorithm"
   - Oliver Alvarado Rodriguez, Mohammad Dindoost, David A. Bader, Tandy Warnow, and George Chacko
     - NJIT and University of Illinois Urbana-Champaign
4. "HyperXplorer: Parallel and Scalable Motif Census Exploration"
   - Mohammad Dindoost, Oliver Alvarado Rodriguez, and David A. Bader
5. "Cluster Metrics to Further Classify Well-Connected Components"
   - Mohammad Dindoost, Bartosz Bryg, Oliver Alvarado Rodriguez, and David A. Bader
6. "Applications of Fast State Injection Subgraph Isomorphism on Connectomes"
   - Oliver Alvarado Rodriguez, Mohammad Dindoost, Michael Shewarega, Jakob Troidl, Hanspeter Pfister, and David A. Bader
7. "Arachne: A Framework for Large-Scale Exploratory Graph Analytics" *culminative journal paper*
   - Oliver Alvarado Rodriguez, Mohammad Dindoost, Zhihui Du, and David A. Bader

Status of Chapter Topic(s) during Proposal Defense	Status Then	Status Now
Introduction – description of problem statement, overview of related graph software, Arkouda overview	🟩	🟩
Arachne 1.0 – data st ture overvie structure, supported graph formats	🟩	🟩
Breadth-First Searc igori c i ov results	🟩	🟩
Truss Analytics – ithm prove re	🟩	🟩
Connected Com nts – algorithmic		🟩
Arachne 1.5 – modify current algo hybrid for both na and dis ted y		🟩
Community Detection – massively parallel well-connected-com iden clustering, c.		🟩
Arachne 2.0 – major overhauls made from community detection suite addition		🟩
Property Graphs – storing property graphs, modify our algorithms for property graph analysis, results		🟩
Arachne 3.0 – highlight significant changes made by introducing property graphs	🟥	🟩

COMPLETED

Key: **not started:** 🟥 **in progress:** 🟧 **completed:** 🟩

References

J. Priem, H. Piwowar, and R. Orr, "OpenAlex: A fully-open index of scholarly works, authors, venues, institutions, and concepts," arXiv preprint arXiv:2205.01833, 2022.

K. Emil, "What is NetFlow? Cisco NetFlow? sFlow? IPFIX? Flow Monitoring Explained," eG Innovations Blog, Sep. 19, 2019. [Online]. Available: https://www.eginnovations.com/blog/what-is-netflow/. [Accessed: Feb. 20, 2025].

FlyWire: A Human-AI Collaboration for Reconstructing the Full Brain Connectome of Drosophila. [Online]. Available: https://flywire.ai/about. [Accessed: Feb. 20, 2025].

GitHub, "Octoverse: AI leads Python to top language as the number of global developers surges," The GitHub Blog, Oct. 2024. [Online]. Available: https://github.blog/news-insights/octoverse/octoverse-2024/. [Accessed: Feb. 20, 2025].

Bears-R-Us, "Arkouda: Interactive Data Analytics at Supercomputing Scale," GitHub repository, [Online]. Available: https://github.com/Bears-R-Us/arkouda. [Accessed: Feb. 20, 2025].

W. Reus, “CHIUW 2020 Keynote: Arkouda: Chapel-Powered, Interactive Supercomputing for Data Science,” in Chapel Implementers and Users Workshop, 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2020, pp. 650–650.

R. Vijayalakshmi, D. Nandagopal, N. Dasari, B. Cocks, N. Dahal, and M. Thilaga, "Minimum connected component – A novel approach to detection of cognitive load induced changes in functional brain networks," *Neurocomputing*, vol. 170, pp. 15–31, 2015, doi: 10.1016/j.neucom.2015.03.092.

Thank You ☺
Questions?