VF2-PS: Parallel and Scalable Subgraph Monomorphism in Arachne

Mohammad Dindoost, Oliver Alvarado Rodriguez, Sounak Bagchi+, Palina Pauliuchenka, Zhihui Du, David A. Bader

Department of Data Science

New Jersey Institute of Technology

Newark, NJ, USA

+Edison Academy Magnet School, Edison, NJ,USA

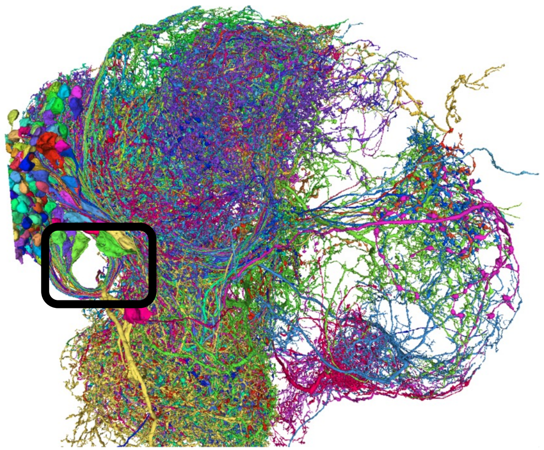
Supported by NSF grant: CCF-2109988 & NSF OAC-2402560

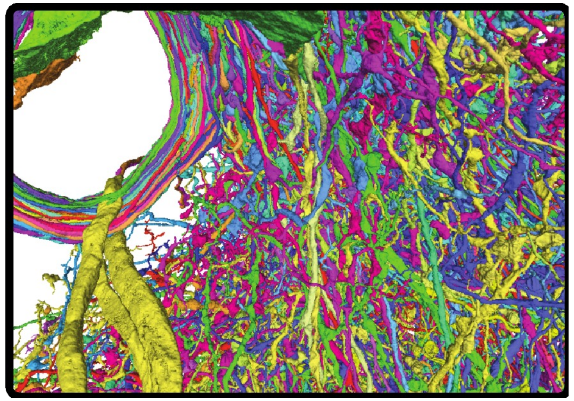
NJIT New Jersey Institute of Technology

Motivations

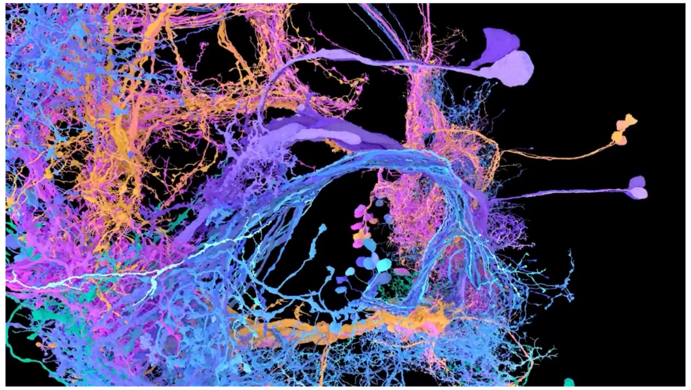
1. Existing tools(mostly based on Python), such as NetworkX and DotMotif, can only handle small graphs/datasets.
2. Highly productive tools are necessary to improve data science performance.
3. An algorithm like subgraph isomorphism requires a way to quickly access the properties of a graph during its semantic check step.

Real use-case: Connectome H01 Analysis

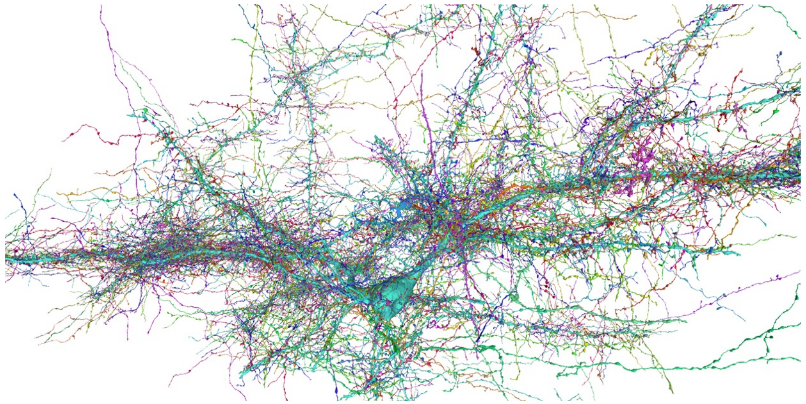




Drosophila Hemibrain Dataset, [Scheffer et al. 2020]



Drosophila Auditory Circuit [Baker et al. 2022]
Video: Amy Sterling, FlyWire

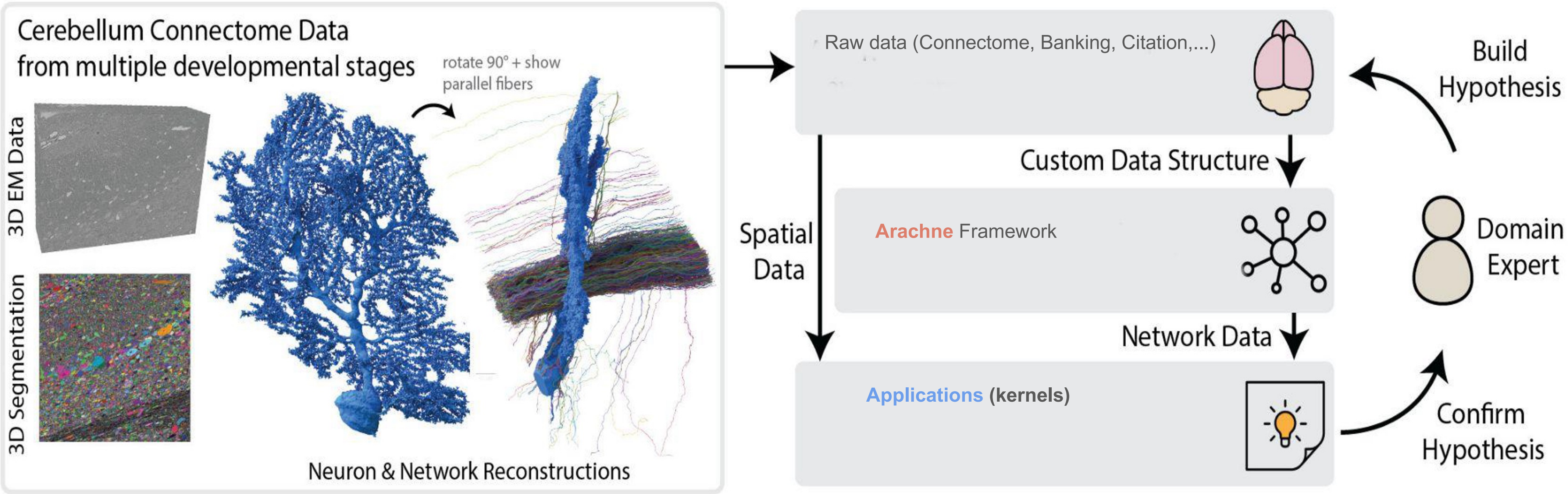


- Using Arachne, we can convert connectome datasets with one hundred million rows of JSON objects to distributed HDF5 files.
- With Arachne, graphs of this size can be queried in seconds.
- Modestly sized ~ 250GB of raw data

Slide credit: Jakob Troidl, Hanspeter Pfister, Jeff Lichtman (Harvard University)

NJIT New Jersey Institute of Technology

Needs



New Jersey Institute of Technology

Arachne addresses these needs

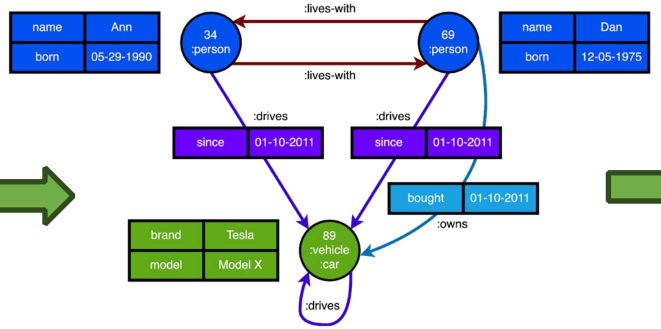
id	label	name	born	brand	model
34	person	Ann	1990	NULL	NULL
69					
89					
89					

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 terabytes-sized CSVs, HDF5s, Parquets, etc.

- read_matrix_market_file()
- read_hdf()
- read_parquet()
- rmat()

Generate or load graphs in from various sources.

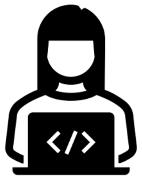


Convert tabular data to a property graph with all data closely maintained with vertex and edges.

- bfs_layers()
- subgraph_isomorphism()
- square_counting()
- subgraph_view()

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

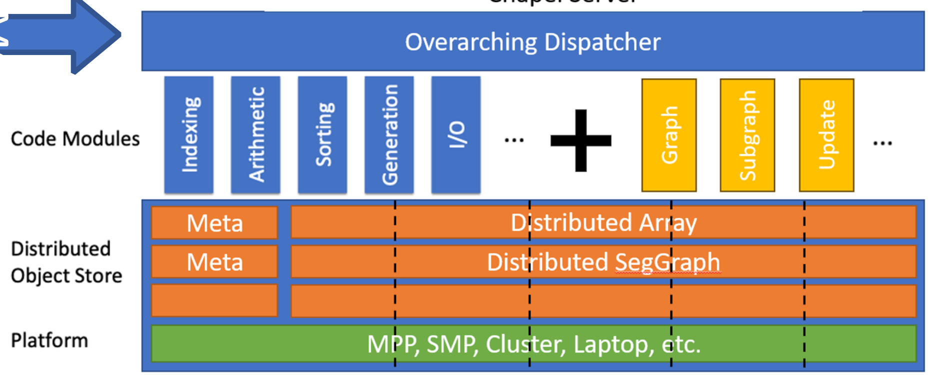
User edits a Python script or a Jupyter Notebook.



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! Reference our HPEC22 paper ☺
```

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. Server can run on supercomputers; Python API usable locally.

OPEN SOURCE: https://github.com/Bears-R-Us/arkouda-njit
PUBLICATIONS & PRESENTATIONS AT: HPEC, HiPC, IPDPS, &

New Jersey Institute of Technology

Problem Definition

The subgraph isomorphism is about finding Subgraph (pattern) inside a larger Graph (host/target). Two version (induced and non-induced). Challenge: NP-complete problem

Algorithm 1 *VF2 [2]*

```
procedure VF2(subgraph, host graph, state)
    if mapping is full then
        return mapping
    end if
    candidates ← GETCANDIDATEPAIRS(subgraph, host graph, state)
    for each candidate c in candidates do
        if c satisfies isFeasible rules then
            NewState ← add new candidate to state
            results ← VF2(subgraph, host, NewState)
        end if
    end for
end procedure
```

Core 1 and Core 2 Keep track of mapping.
The most time consuming part!

Recursion

NJIT
New Jersey Institute of Technology

VF2-PS Algorithm

Algorithm 2 Parallel *VF2-PS* algorithm that generates the mappings of vertices u from the host graph that are mapped to vertices v from the subgraph.

Input: A state $S_{current}$ with the current mapping information for a given recursive depth d.

Output: Mappings M of all host graph and subgraph pairs that induce a monomorphism.

$M = list(int)$ ▷ Parallel-safe list.
if $d == n_2$ **then** ▷ n_2 is the size of the subgraph.
  for $v \in S_{current}.core$ **do**
    $M.pushBack(v)$
  end for
  return M
end if
$candidates = getCandidatePairs(S_{current})$
for all $(u, v) \in candidates$ **do**
  if $isFeasible(u, v, S_{current})$ **then**
    $S_{clone} = S_{current}.clone()$
    $addToTinTout(u, v, S_{clone})$
    $M_{new} = VF2(S_{clone}, d + 1)$
    for $m \in M_{new}$ **do**
      $M.pushBack(m)$
    end for
  end if
end for all
return M

Core Keeps track of mapping.

where each task will execute on a given candidate, based on current state

New Jersey Institute of Technology

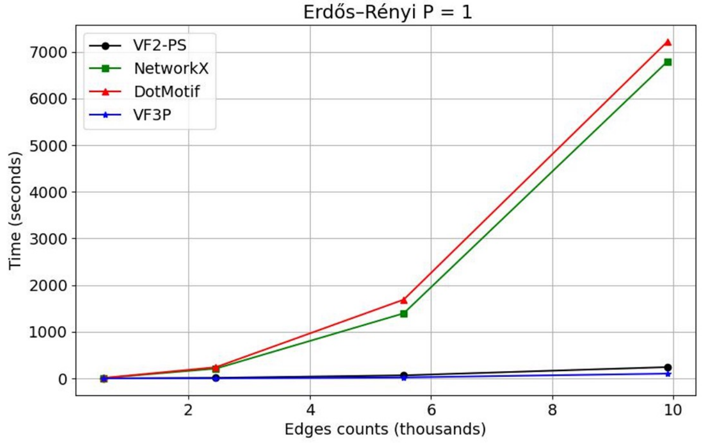
Experiments

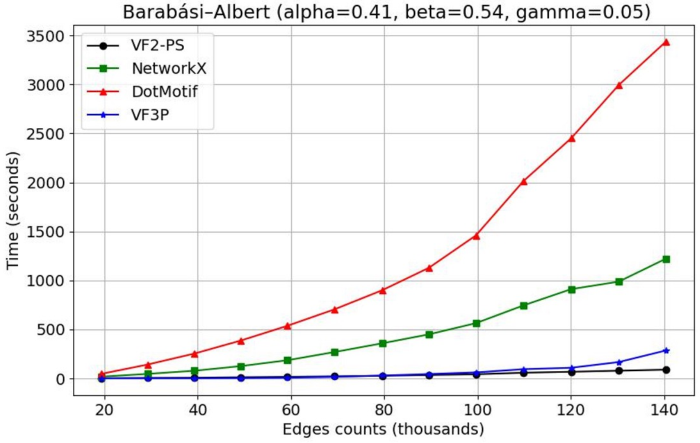
- We conducted systematic comparisons between our implementation and those from well-established and widely used Python libraries such as NetworkX and DotMotif. Additionally, we experimented with VF3P.
- Synthetic graphs
  - The synthetic directed graphs were derived from standard random graph models, including Erdős–Rényi, Watts–Strogatz, and Barabási–Albert, which are frequently used in network analysis studies.
- Real-world datasets
  - the Hemibrain v.1.2 dataset, the Enron email network, and the Math Overflow temporal network

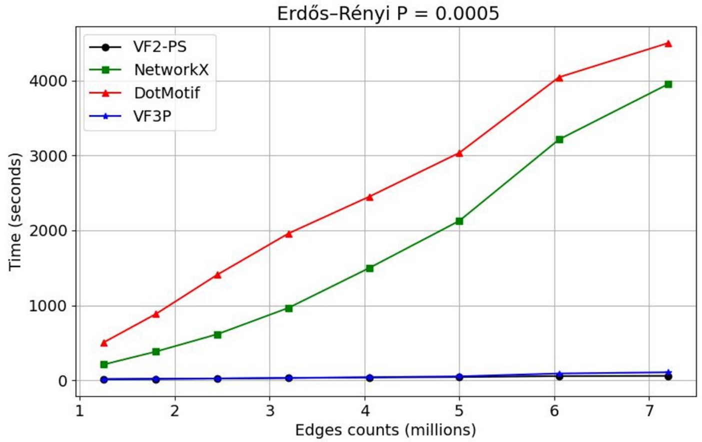
REAL-WORLD DATASETS USED FOR EXPERIMENTATION, SORTED BY THE NUMBER OF EDGES.

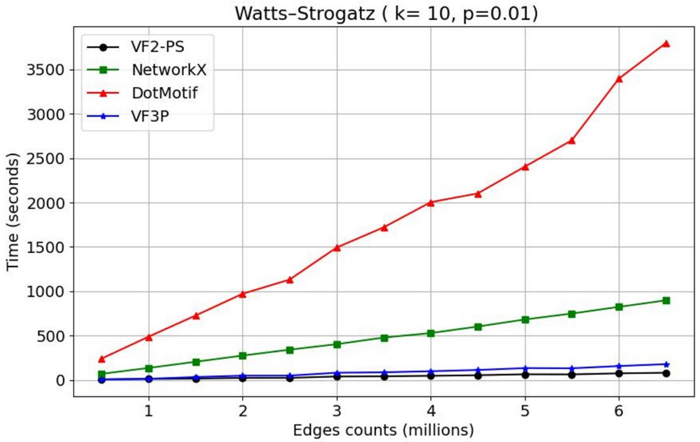
Dataset	Number of vertices	Number of edges	Density	Field
Enron Emails	36,692	183,831	0.0001	Communication network
Math Overflow	24,818	506,550	0.0008	Social network
Hemibrain v1.2	21,739	3,550,403	0.0075	Neuroscience

Performance on Synthetic graphs



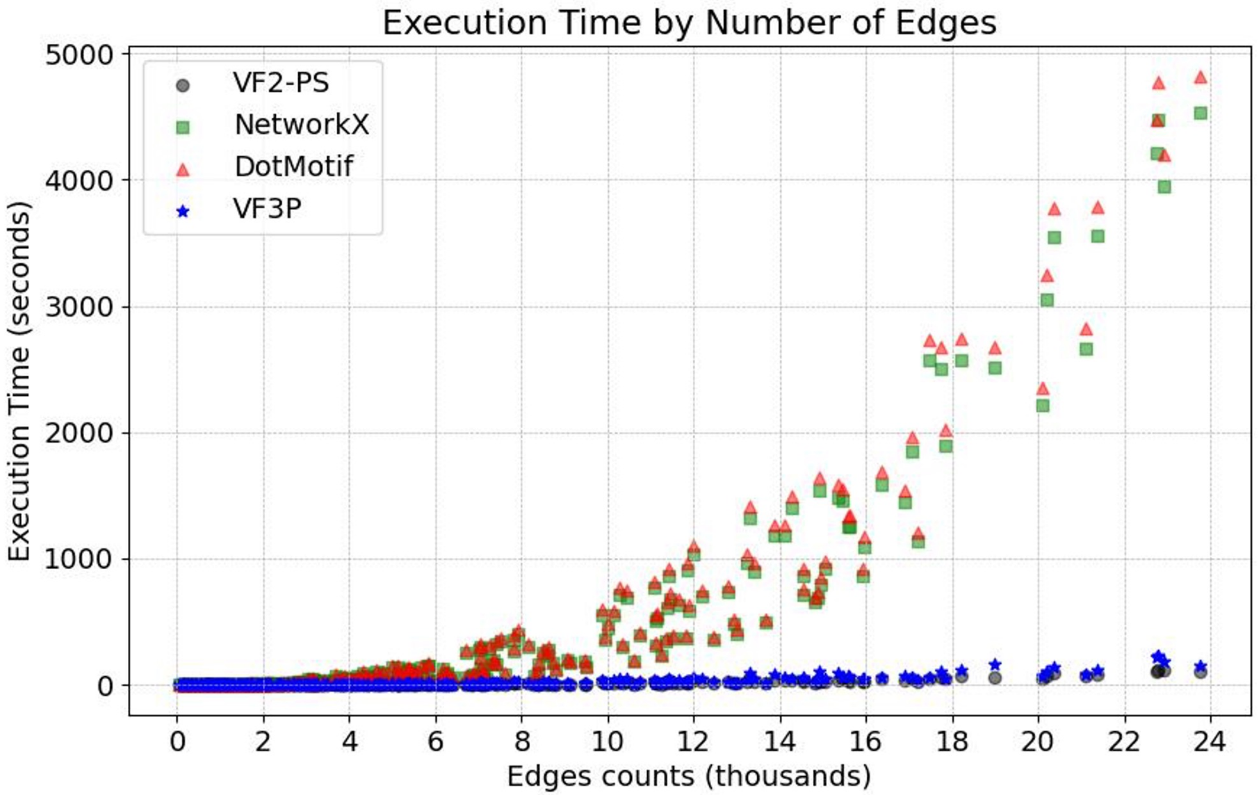






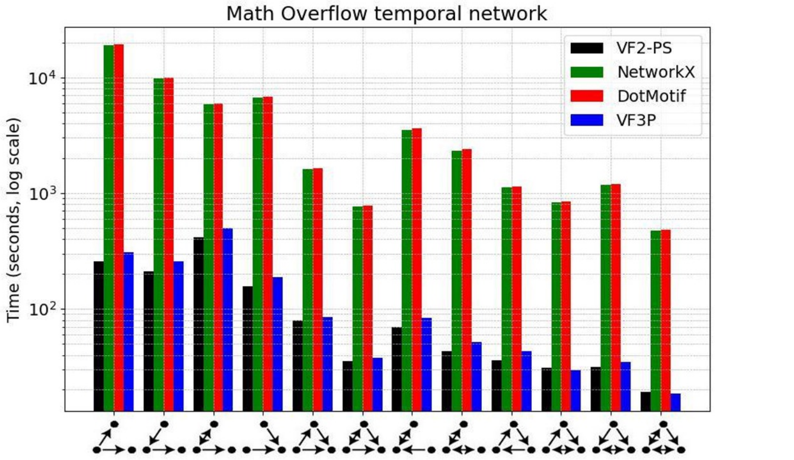
Performance with Various Structures

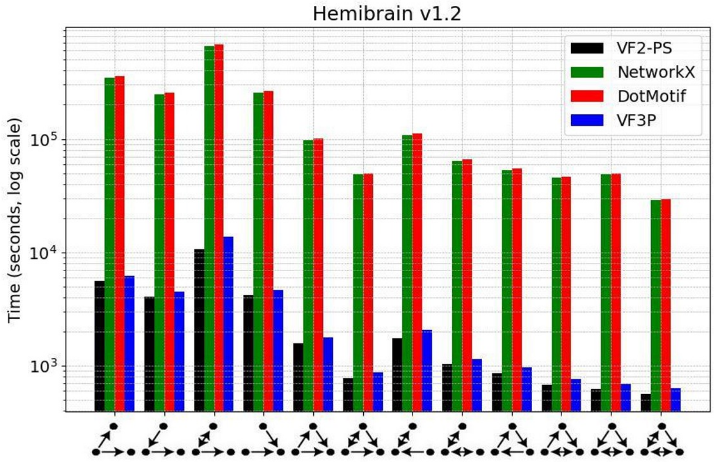
- These tests were carefully designed to examine the effect of both network size and subgraph size and structure
- we produced 300 distinct directed Erdős–Rényi graphs.The vertex counts for these graphs were uniformly distributed within a range of 100 to 300, and their edge densities were uniformly sampled from a continuum spanning 0.05 to 0.1.

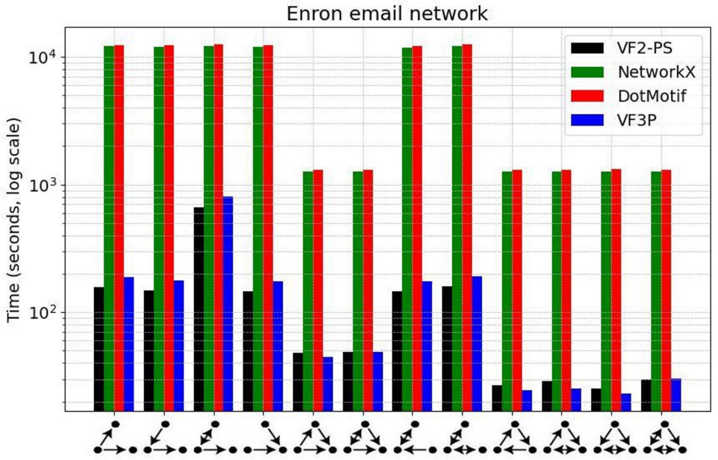


Performance on Real-World Graphs

- For the **Enron email network**, VF2-PS achieved an impressive **speedup of 81.5** times compared to the widely used NetworkX. Similarly, in the **Stack Overflow dataset**, VF2-PS facilitated a **speedup of 72.8** times, and for the **Hemibrain dataset**, the **speedup reached 97.0** times. These metrics highlight Arachne's robust performance and precision in motif finding tasks.



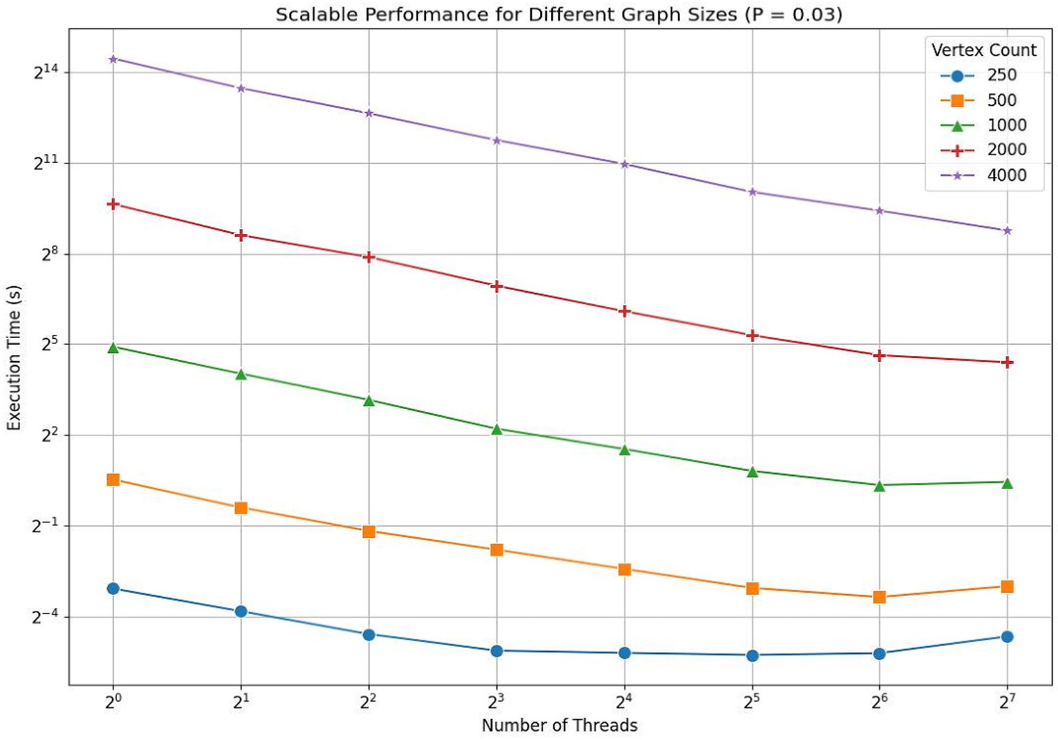




New Jersey Institute of Technology

Scalable Performance

- we use different numbers of parallel threads to run the same task on the same graph. In Chapel, we can update the value of the environmental variable CHPL_RT_NUM_THREADS_PER_LOCALE to vary the number of threads.



Conclusion

- A parallel and optimized implementation of the VF2 algorithm for subgraph monomorphism implemented into Arachne.
- Comprehensive experimental results on synthetic and real-world graphs showing that our subgraph monomorphism method is significantly faster than the ubiquitous, Python-based, graph packages, DotMotif and NetworkX. Additionally, it shows better performance compared to VF3 Parallel.
- Our solution easily can handle massive graphs.

Thank You ☺
Questions?

Problem Definition

- The subgraph isomorphism is about finding Subgraph (pattern) inside a larger Graph (host/target)
- Two version (induced and non-induced).
- Challenge: NP-complete problem

Ciaran McCreech et al. 2018

New Jersey Institute of Technology

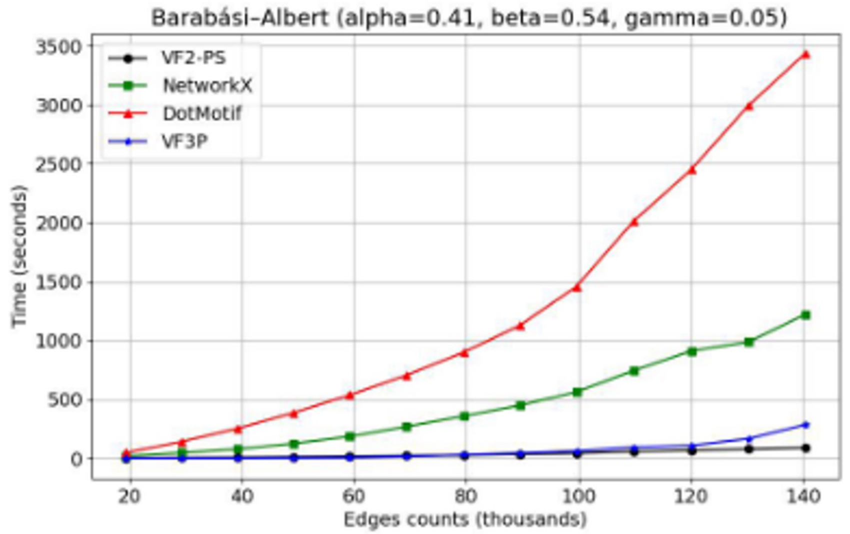
Optimization Methods

- Reduce the amount of space utilized by the VF2 implementation by restructuring the state data structure.
  - In the original VF2 data structure, two vectors, core_1 and core_2 are used to keep the current mapping. However, we use core_2[n_2]=n_1 to keep the current mapping (n_1,n_2). This can save the space used by state variables and make the search for unmapped vertices easy. Based on the simplified state data structure, it suffices to check the value of core_2[i], 0<= I< |V_2|-1, to know if a vertex has been mapped.
- Invoke very productive and fast parallelization in Chapel that automatically creates parallel tasks and assigns them to available threads dynamically dependent on the amount of threads at a given time.
- the parallel for loop will split up into many blocks where each task will execute on a given block.
  - Any freed threads can be utilized by subsequent recursive calls, provided they become available when the execution reaches one of the nested 'for all' loops.

New Jersey Institute of Technology

Performance on Scale-free Networks

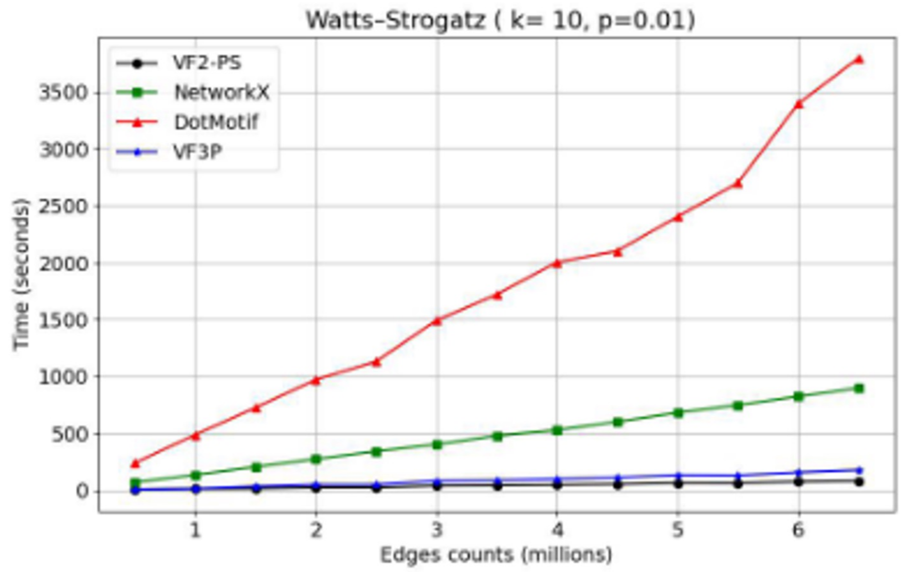
- This Figure shows one of the many generated configurations, characterized by parameters $\alpha = 0.41$, $\beta = 0.54$, and $\gamma = 0.05$.
- a network with moderate preferential attachment and a higher propensity for internal connections, resulting in moderate clustering and balanced degree distribution.



New Jersey Institute of Technology

Performance on Small-world Networks

- In theses setups we tried to demonstrate the impact of the rewiring probability in transitioning from highly structured networks to those exhibiting more random properties.
- It can be seen that VF2-PS can efficiently handle the intricate structures of small-world graphs with millions of edges, which are common in many real-world scenarios.



NJIT New Jersey Institute of Technology