Truss Analytics Algorithms and Integration in Arkouda

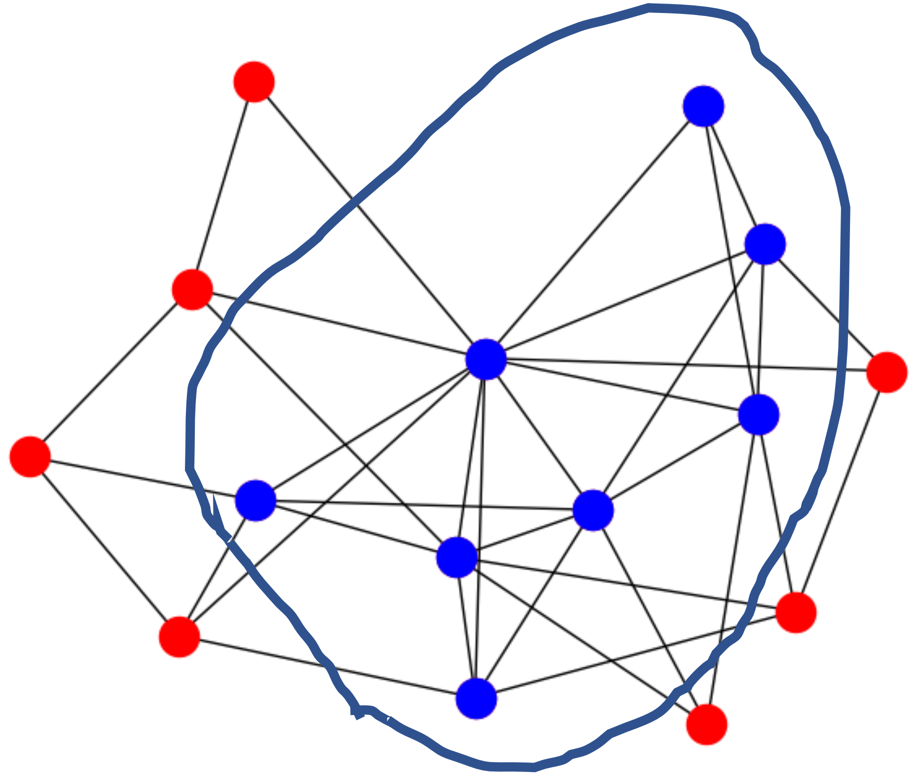
Zhihui Du (Presenter)
Joseph Patchett
Oliver Alvarado Rodriguez
David A. Bader

CHIUW 2022

This research is supported by NSF grant CCF-2109988

What is a Truss

- K-Truss
  - a cohesive subgraph to explore **close relationships** in a graph
    - every edge must be part of k−2 triangles in the subgraph
  - value K
    - the degree of closeness and the size of the subgraph
    - Larger k means a larger group with a closer connection
  - Widely used
    - polynomial time, a tractable problem (no NP-complete problem) in theory



K=4 truss
Connected by the blue vertices

Why Truss Analytics

- Answer three questions about a group of elements
  - Who are in the **group** that meets k requirement?
    - K-Truss problem
  - Who are in the **maximal group** that all members have the closest connection?
    - Max K-Truss problem
  - What are all the **different groups** that meet the different degrees of relationships?
    - Truss Decomposition problem
- A typical community detection method
  - Explore insights from a large graph

NJIT New Jersey Institute of Technology

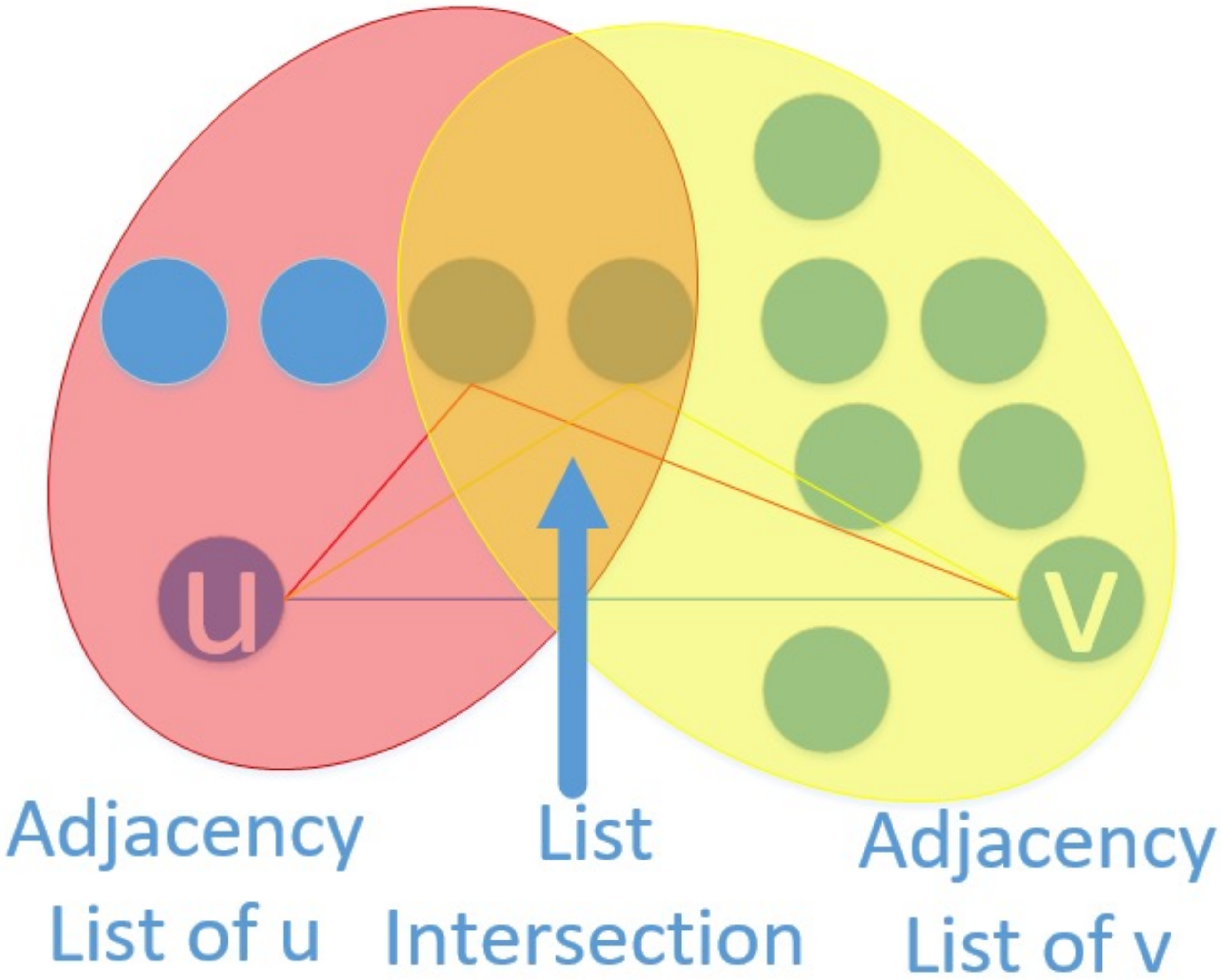
Algorithm Core and Contributions

- Algorithm Core
  - Triangle Counting
- Contributions
  - A parallel triangle counting core with O(logn) instead of O(n) time
  - Productive and High-Performance Tool for truss analytics
    - Use Chaple to implement the algorithm and Use Python as the Interface (Arkouda)

New Jersey Institute of Technology

Core Algorithm Description

- Existing Methods
  - List Intersection
    - Binary Search (O(nlog(n)), two adjacency lists
    - Path Merge (O(n)), two adjacency lists
- Proposed Method
  - Minimum Search (three adjacency lists)
    - Time complexity O(logn)
      - degree(u)<degree(v), w in adjacency of u
      - If degree(w)<degree(v)
        - Search v in the adjacency of w
      - If degree(w)>=degree(v)
        - Search w in the adjacency of v
    - degree(u) parallel threads
    - Significant time savings for real-world graphs



Building Parallel k-Truss Algorithms

Algorithm 1: Naive K-Truss Parallel Algorithm

$NaiveKTruss(G, k)$
/* $G = \langle E, V \rangle$ is the input graph with edge set E and vertex set V. k is the given K-Truss value.
$EdgeDel[\,] = -1$ // initialize all edges as not deleted
while *there is any edge can be deleted* **do**
    $sup[\,] = 0$ // initialize the triangle counting array
    forall (*undeleted edge* $e = \langle u, v \rangle \in E$) && ($e$ *is local*) **do**
        calculate $sup(e, G)$ using list intersection or minimum search method
        $sup[e] = sup(e, G)$
    end
    forall ($e = \langle u, v \rangle \in E$) && ($e$ *is local*) **do**
        if ($EdgeDel[e] == -1$) && ($sup[e] < k - 2$) **then**
            $EdgeDel[e] = k - 1$
        end
    end
end
return $EdgeDel$

Calculate # Triangles in parallel

Remove Edges that cannot Support k-2 Triangles

NJIT New Jersey Institute of Technology

Building Optimized Parallel K-Truss Algorithms

- Calculate Triangles in Parallel
- Remove edges that cannot support k-2 triangles using minimum search method
- LOOP (search affected edges)
  - Use minimum search method to identify affected edges and update triangle supports
  - Remove edges that cannot support k-2 triangles

Algorithm 2: Optimized K-Truss Parallel Algorithm

```
OptKTruss(G,k)
/* G = ⟨E,V⟩ is the input graph with edges set E and vertices set V. k is the given K-Truss value.
EdgeDel[] = −1 // initialize all edges as undeleted
sup[] = 0 // initialize the support array of each edge
SetDel = ϕ;SetAff = ϕ
forall (edge e ∈ E) && (e is local) do
    sup[e] = sup(e, G) using minimum search method
end
forall (e ∈ E) && (e is local) do
    if (EdgeDel[e] == −1) && (sup[e] < k − 2) then
        EdgeDel[e] = 1 − k
        Add e into SetDel
    end
end
while (SetDel is not empty) do
    forall (e1 ∈ SetDel) && (e1 is local) do
        using minimum search method to find e2 and e3 that can form a triangle with e1
        reduce the support of e2 and e3 if they are undeleted edges
        add the affected edges into SetAff if their supports are less than k − 2
    end
    forall (e ∈ SetDel) && (e is local) do
        if (EdgeDel[e] == 1 − k) then
            EdgeDel[e] = k − 1
        end
    end
    SetDel.clear()
    SetDel <=> SetAff // switch the values of the two sets.
end
return EdgeDel
```

New Jersey Institute of Technology

Building Parallel Max K-Truss Algorithms

- Estimate the upper bound value of kUp
- Modified Binary Search in [3, kUp]
  - The k-truss is not empty but (k+1) truss is
    - If kLow is not empty then check kUp
    - If kUp is empty then check kMid
      - While kMid is empty then update kMid and check again
      - If kMid is not empty then update kLow and search recursively

Algorithm 3: Max K-Truss Parallel Algorithm

$MaxKTruss(G)$
/* $G = \langle E, V \rangle$ is the input graph with edges set E and vertices set V.
Let $k_{low} = 3$ and set k_{up} based on the proposed analysis method
return $DownWardSearch(G, k_{low}, k_{up})$
$function\ DownWardSearch(G, k_{low}, k_{up})$
$EdgeDel = kTruss(G, k_{low})$
if *(All edges have been deleted)* **then**
    return $(k_{low} - 1, EdgeDel)$
end
else
    $EdgeDel = kTruss(G, k_{up})$
    if *(there are undeleted edges in $EdgeDel$)* **then**
        return $(k_{up}, EdgeDel)$
    end
    else
        $k_{mid} = (k_{low} + k_{up})/2$
        $EdgeDel = kTruss(G, k_{mid})$
        while *(All edges have been deleted in $EdgeDel$)* **do**
            $k_{up} = k_{mid} - 1$
            $k_{mid} = (k_{low} + k_{up})/2$
            $EdgeDel = kTruss(G, k_{mid})$
        end
        if $(k_{mid} == k_{up} - 1)$ **then**
            return $(k_{mid}, EdgeDel)$
        end
        else
            $k_{low} = k_{mid} + 1$
            return $DownwardSearch(G, k_{low}, k_{up})$
        end
    end
end

Data Structure Design and Selection

- High-level Set/DistBag and Low-level Arrays
  - Easy<->Performance
  - List intersection using set.contains operation is expensive
- Atomic Variables
  - The number of triangles updated by multiple removed edges
  - The total number of removed edges updated by multiple removed edges
- ForAll/CoForAll
  - Implicit synchronization among different parallel threads

NJIT
New Jersey Institute of Technology

Experimental Setup

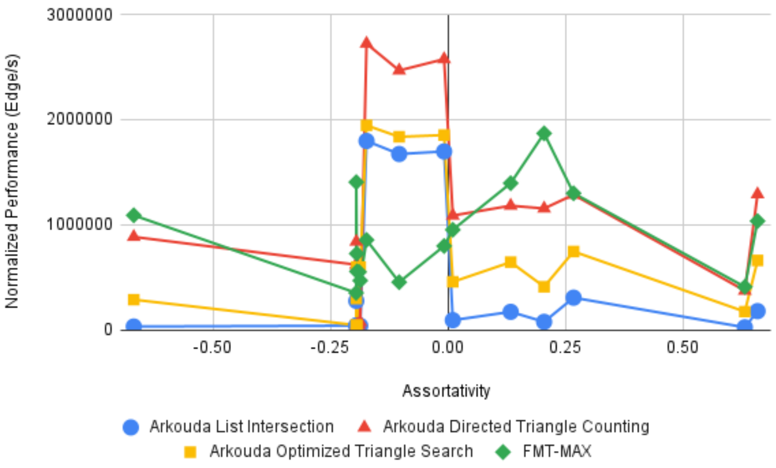
- Data sets
  - 9 real-world graphs
  - 7 synthetic graphs
- Comparison algorithms
  - Naïve List Intersection (set)
  - Naïve Min Search (set)
  - Optimized Min Search (array)
  - Path Merge (not given in the paper)

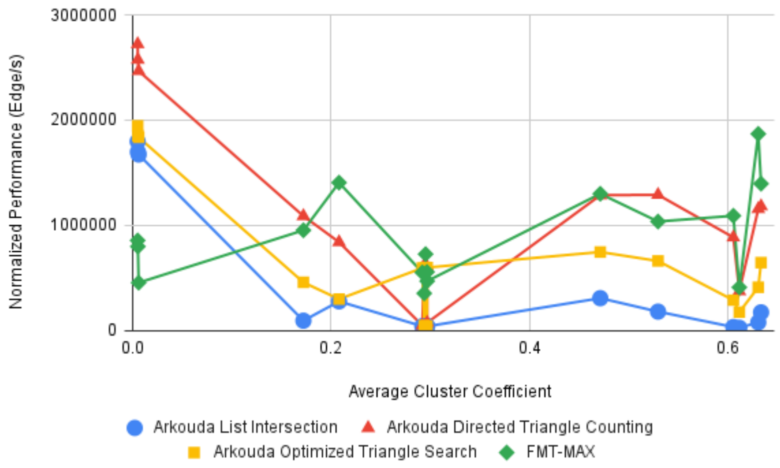
New Jersey Institute of Technology

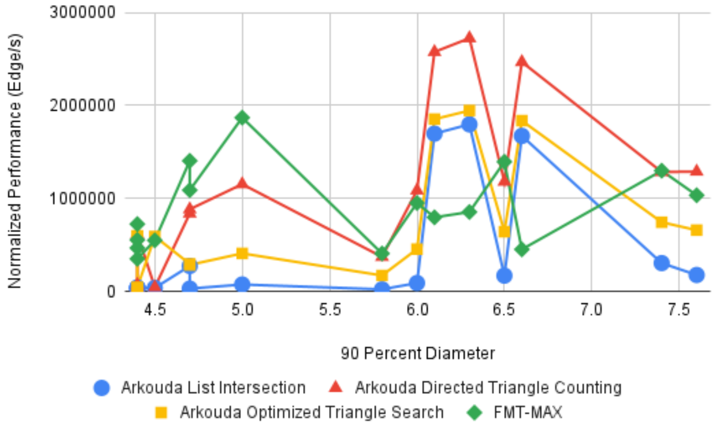
Performance Comparison

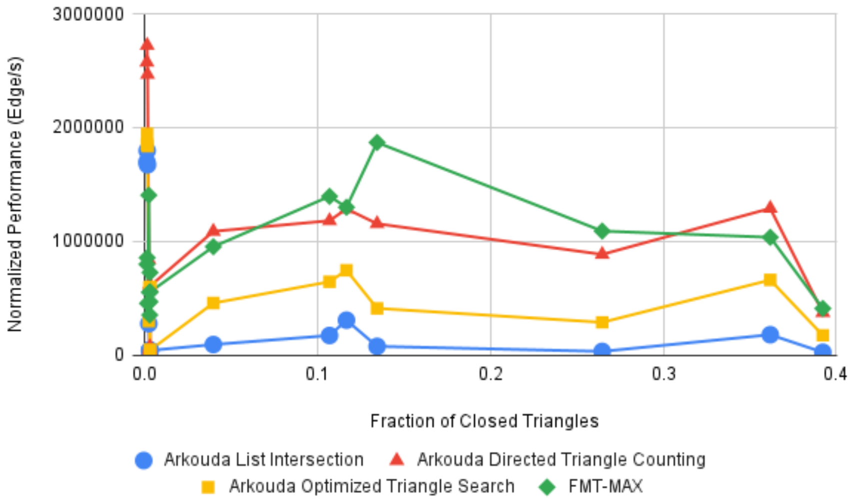
Graph	LI Naive K-Truss	MS Naive K-Truss	MS Opt K-Truss	MS Max K Truss	MS Truss Decomposition	Speedup 1	Speedup 2
amazon0601	1008.58	509.29	60.61	93.22	66.22	2.0	16.6
as-caida20071105	16.70	2.98	1.00	1.73	0.88	5.6	16.7
ca-AstroPh	113.28	56.11	9.64	11.16	5.17	2.0	11.7
ca-CondMat	23.52	11.58	2.11	2.58	2.21	2.0	11.2
ca-GrQc	2.49	1.24	0.29	0.35	0.36	2.0	8.6
ca-HepPh	29.33	14.69	3.07	3.22	3.45	2.0	9.6
ca-HepTh	3.88	1.93	0.50	0.61	0.61	2.0	7.7
com-Youtube	4885.27	302.37	55.72	71.89	61.94	16.2	87.7
delaunay_n10	2.04	1.05	0.08	0.09	0.08	1.9	25.5
delaunay_n11	5.50	2.81	0.16	0.18	0.16	2.0	34.2
delaunay_n12	14.00	7.15	0.32	0.36	0.31	2.0	44.3
delaunay_n13	36.69	18.74	0.62	0.70	0.61	2.0	58.9
delaunay_n14	98.61	50.46	1.23	1.46	1.22	2.0	79.9
delaunay_n15	266.96	136.52	2.49	2.93	2.45	2.0	107.3
delaunay_n16	735.75	378.16	4.91	5.83	4.87	1.9	149.8

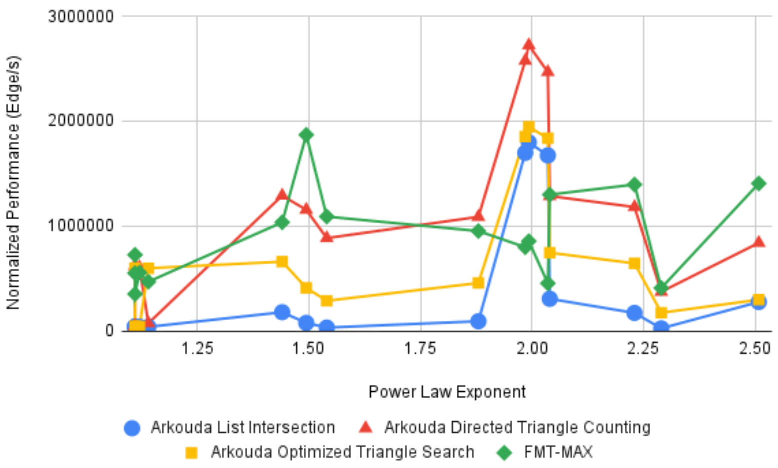
Preliminary Graph topology and Normalized Performance Analysis











New Jersey Institute of Technology

Conclusion

- Algorithm
  - The novel parallel **minimum search**-based method can really improve the performance Truss Analytics Algorithms compared with the widely used existing list intersection methods.
- Language
  - Chapel's high-level data structure (set/distbag…), atomic variables and parallel structure forall/coforall are very helpful to implement the parallel truss algorithms **productively**
- Data Science Environment
  - Arkouda can help **Python users** to employ the provided large-scale truss analytics with high performance

NJIT
New Jersey Institute of Technology

Acknowledgement

We appreciate the help from Michael Merrill, William Reus, Brad Chamberlain, Elliot Joseph Ronaghan, Engin Kayraklioglu, David Longnecker and the Chapel and Arkouda community when we integrated the algorithms into Arkouda. This research was funded in part by NSF grant number CCF-2109988.

Thank You!

Q&A