Truss Analytics Algorithms and Integration in Arkouda

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

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
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
Algorithm Core and Contributions

• Algorithm Core
  • Triangle Counting

• Contributions
  • A parallel triangle counting core with $O(\log n)$ 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)
Core Algorithm Description

• Existing Methods
  • List Intersection
    • Binary Search \(O(n\log(n))\), two adjacency lists
    • Path Merge \(O(n)\), two adjacency lists

• Proposed Method
  • Minimum Search (three adjacency lists)
    • Time complexity \(O(\log n)\)
      • \(\text{degree}(u) < \text{degree}(v)\), \(w\) in adjacency of \(u\)
      • If \(\text{degree}(w) < \text{degree}(v)\)
        • Search \(v\) in the adjacency of \(w\)
      • If \(\text{degree}(w) \geq \text{degree}(v)\)
        • Search \(w\) in the adjacency of \(v\)
    • \(\text{degree}(u)\) parallel threads
    • Significant time savings for real-world graphs
Building Parallel k-Truss Algorithms

**Algorithm 1: Naive K-Truss Parallel Algorithm**

```plaintext
Algorithm NaiveKTruss(G, k)
/* G = {E, V} is the input graph with edge set E and vertex set V. k is the given K-Truss value. */

1. EdgeDel[] = -1 // initialize all edges as not deleted
2. while there is any edge can be deleted do
3.     sup[] = 0 // initialize the triangle counting array
4.     forall (undeleted edge e = (u, v) ∈ E) && (e is local) do
5.         calculate sup(e, G) using list intersection or minimum search method
6.         sup[e] = sup(e, G)
7.     end
8.     forall (e = (u, v) ∈ E) && (e is local) do
9.         if (EdgeDel[e] == -1) && (sup[e] < k - 2) then
10.        EdgeDel[e] = k - 1
11.       end
12.     end
13. end
14. return EdgeDel
```

**Calculate # Triangles in parallel**

**Remove Edges that cannot Support k-2 Triangles**
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
Building Parallel Max K-Truss Algorithms

- Estimate the upper bound value of $k_{Up}$
- Modified Binary Search in $[3, k_{Up}]$
  - The $k$-truss is not empty but $(k+1)$ truss is
    - If $k_{Low}$ is not empty then check $k_{Up}$
    - If $k_{Up}$ is empty then check $k_{Mid}$
      - While $k_{Mid}$ is empty then update $k_{Mid}$ and check again
      - If $k_{Mid}$ is not empty then update $k_{Low}$ and search recursively
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
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)
## Performance Comparison

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

• Algorithm
  • The novel parallel **minimum search**-based method can really improve the performance of 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 for all/co for all 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
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