On Efficient External-Memory Triangle Listing

Yi Cui, Di Xiao, and Dmitri Loguinov

Internet Research Lab (IRL)
Department of Computer Science and Engineering
Texas A&M University, College Station, TX, USA 77843
December 13, 2016
Agenda

• Introduction
• Background
• Analysis
• Pruned Companion Files
• Implementation
• Experiments
Introduction

• Given a simple undirected graph $G = (V, E)$, list all triangles $\Delta_{ijk}$ such that $i, j, k \in V$ and $(i, j), (j, k), (i, k) \in E$

• Triangle listing has many important applications
  – Network analysis: clustering coefficient, transitivity
  – Web/social networks: spam/community detection
  – Graphics, databases, bioinformatics, theory of computing

• It may seem like a simple problem at first glance; however, there are many open issues
  – Modeling CPU cost under different acyclic orientations, choosing the best search order, understanding I/O complexity, and designing faster algorithms
  – Our goal here is to address some of these questions
Agenda

• Introduction
• Background
• Analysis
• Pruned Companion Files
• Implementation
• Experiments
Background

• There are $3! = 6$ ways to list each triangle $\Delta_{ijk}$
  – Doing so involves redundant computation and requires additional effort for duplicate elimination
  – Worse yet, complexity is a function of the second moment of undirected degree

• Significantly better results are possible by converting the graph into a directed version and checking each possible triangle exactly once
  – Second moments of directed degree are much smaller
  – CPU cost improves not just by 6x, but often by orders of magnitude (e.g., 1000x on Twitter)

• Suppose $G$ has $n$ nodes and $m$ edges
Background

- All prior work on creation of directed graphs can be unified by a two-step process
  - **Relabeling**: Shuffle nodes with some permutation $\theta$, then sequentially label nodes from 1 to $n$
  - **Acyclic orientation**: Direct edges from nodes with larger labels to those with smaller

- There are a total of $n!$ possible permutations of nodes

- Well-known orientations
  - Ascending (A) / Descending (D) degree
  - Round-Robin (RR) / Complementary Round-Robin (CRR)
  - See the paper for details
Agenda

• Introduction
• Background
• Analysis
• Pruned Companion Files
• Implementation
• Experiments
Search Order Analysis

- Suppose the search starts with $i$, continues to $j$, and finishes with $k$
  - But how to choose the relationship between these nodes?

- There are six search orders in oriented graphs
  - For example: $i > j > k$ starts from the largest node, continues to the middle node, and finishes with the smallest
  - Some search orders visit only in-neighbors, some only out-neighbors, and others do both

- Interestingly, the search order coupled with permutation $\theta$ greatly affects CPU and I/O complexity!
  - Not formally observed or studied before
Generalized Iterators (GI)

- To study this further, we propose a framework of 18 triangle-search techniques that subsumes all previous methods
- Generalized Vertex Iterator (GVI)
  - Methods T₁-T₆
- Generalized Lookup Edge Iterator (GLEI)
  - Methods L₁-L₆
- Generalized Scanning Edge Iterator (GSEI)
  - Methods E₁-E₆
- The first two rely on hash tables, the last one on sequential intersection of neighbor lists
Comparison Objectives

- Triangle listing has four performance metrics
  - CPU cost (\# of hash table lookups for GVI, GLEI and intersection length for GSEI)
  - Amount of sequential I/O (our focus today)
  - Auxiliary hash table lookups (see the paper)
  - Minimum RAM that the method supports (see the paper)

- The CPU cost is modeled in our PODS 2017 paper
  - Among the 18 methods, only 4 have non-equivalent CPU cost

- But what about I/O?
  - Can all 18 methods be implemented in a single algorithm? How many I/O-equivalence classes are there? Which method is best? Under what permutation?
Does Orientation Affect I/O?

- MGT [Hu SIGMOD13]
  - Load the graph in chunks of memory size (one edge), scan the entire $G$ to pick up the remaining two edges
  - Assuming RAM size $M$, MGT reads $m^2/M$ edges from disk

- Pagh [Pagh PODS14]
  - Randomly color nodes with $c = \sqrt{m/M}$ colors and partition edges into $c^2$ subgraphs; run MGT over $c^3$ triples of subgraphs for a total I/O of $9m^{1.5}/M$

- Neither method depends on acyclic orientation and thus search order; however, can we do better?
  - We know orientation reduces CPU cost, can it help with I/O?
  - We consider this novel idea below
Agenda

- Introduction
- Background
- Analysis
- Pruned Companion Files
- Implementation
- Experiments
Pruned Companion Files (PCF)

- Our framework for external-memory triangle listing
  - Two steps: graph partitioning and creation of companion files
  - Due to random lookups, edge \((j, k)\) must be loaded in RAM; however, the other two edges of each triangle can be scanned from the corresponding companion file

- Partitioning
  - Split \(V\) into \(p\) exhaustive, pair-wise non-overlapping sets \(V_1, V_2, \ldots, V_p\)
  - Partition \(G\) into subgraphs \(G_1, G_2, \ldots, G_p\), where \(G_l\) has all edges with either \(k\) (PCF-A) or \(j\) (PCF-B) in \(V_l\)

- The paper shows that PCF-A produces different I/O from PCF-B, provides algorithms for deterministically load-balancing partitions (omitted here)
Pruned Companion Files (PCF)

- For each $G_l$, we create a companion file $C_l$ that contains the missing edges
  - The paper covers all 18 methods in one simple algorithm
  - Extra care is taken to minimize the size of $C_l$

- **Theorem 1**: For all $p \geq 1$, PCF finds each triangle exactly once and its CPU cost remains constant
Pruned Companion Files (PCF)

- When combining CPU cost and I/O, we find 16 algorithms (PCF-A/B for each of the 8 CPU classes)
  - Each cell is different from every other

- Findings
  - As it turns out, $E_1$ has better I/O than $E_2$!
  - Only two methods ($T_1$ and $E_1$) require the same $\theta$ to achieve optimal CPU cost and I/O
  - $T_1$ and $E_1$ are winners in their categories
  - PCF-B outperforms PCF-A, achieves minimal number of auxiliary lookups, and lowest RAM usage
Scaling Rate of I/O

- **Theorem 2**: Under PCF-B and mild constraints on degree, both $T_1$ and $E_1$ have linear I/O for all $M$
- In contrast, prior work requires $M$ to scale at least as fast as $m$ for this to happen
  - Consider Twitter as an illustration (9.3 GB, 1.2B edges)
  - For $M = 1$ MB, PCF shows a 75x improvement over MGT and 10x over Pagh

<table>
<thead>
<tr>
<th></th>
<th>RAM (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1024</td>
</tr>
<tr>
<td>MGT</td>
<td>5.39</td>
</tr>
<tr>
<td>Pagh</td>
<td>22.91</td>
</tr>
<tr>
<td>PCF</td>
<td>1.48</td>
</tr>
</tbody>
</table>

I/O (billion edges) vs. RAM in Twitter (1.2B edges)
Agenda

• Introduction
• Background
• Analysis
• Pruned Companion Files
• Implementation
• Experiments
Implementation

• Besides cost, we consider the speed of operations
  – Hash table lookups for GVI/GLEI and intersection for GSEI
  – We dismiss GLEI as it is always inferior to GVI

• The optimal choice boils down to T₁ vs E₁
  – They have the same I/O, but CPU cost differs
  – T₁ has fewer operations, but they are inherently slower
  – Google hash table: 19M/sec
  – Naive scalar intersection: 264M/sec (14x faster)

• In real-world graphs, E₁ has only 2-3x more CPU cost
  – However, our PODS 2017 paper shows existence of graphs
    where the cost ratio goes unbounded as \( n \to \infty \), i.e., T₁ is
    always faster in the limit
Implementation

- **PaCiFier**: Our implementation of $E_1$ under PCF-B
  - Efficient preprocessing (i.e., relabeling and orientation)
  - Intersection with SIMD (Single Instruction Multiple Data)
  - Compressed labels to 16 bits for faster intersection

<table>
<thead>
<tr>
<th></th>
<th>Speed (M/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branchless intersection</td>
<td>416</td>
</tr>
<tr>
<td>SIMD 32-bit intersection</td>
<td>1,119</td>
</tr>
<tr>
<td>SIMD 16-bit intersection</td>
<td>1,801</td>
</tr>
</tbody>
</table>

- Multi-core parallelization
- CPU and I/O parallelization
Agenda

• Introduction
• Background
• Analysis
• Pruned Companion Files
• Implementation
• Experiments
Experiments

- Setup: six-core Intel i7-3930K 4.4 GHz, 8 GB RAM
- PaCiFier’s preprocessing is over 2x faster than the closest competitor (see the paper)
- Compare to the fastest vertex iterator (MGT) and the fastest edge iterator (PDTL from [Giechaskiel ICPP15])
  - PaCiFier is 14-79x faster than MGT and 5-10x than PDTL

<table>
<thead>
<tr>
<th>Graph</th>
<th>Nodes</th>
<th>Edges</th>
<th>Triangle</th>
<th>Size (GB)</th>
<th>MGT</th>
<th>PDTL</th>
<th>PaCiFier</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebUK</td>
<td>62.3M</td>
<td>1.9B</td>
<td>179.1B</td>
<td>7.5</td>
<td>599</td>
<td>94</td>
<td>17</td>
</tr>
<tr>
<td>Twitter</td>
<td>41.7M</td>
<td>2.4B</td>
<td>34.8B</td>
<td>9.3</td>
<td>2,238</td>
<td>327</td>
<td>63</td>
</tr>
<tr>
<td>Yahoo</td>
<td>720.2M</td>
<td>12.9B</td>
<td>85.8B</td>
<td>53.3</td>
<td>1,080</td>
<td>619</td>
<td>79</td>
</tr>
<tr>
<td>IRL-domain</td>
<td>86.5M</td>
<td>3.4B</td>
<td>112.8B</td>
<td>13.3</td>
<td>5,946</td>
<td>849</td>
<td>148</td>
</tr>
<tr>
<td>IRL-host</td>
<td>642.0M</td>
<td>12.9B</td>
<td>437.4B</td>
<td>52.7</td>
<td>11,099</td>
<td>1,773</td>
<td>367</td>
</tr>
<tr>
<td>IRL-IP</td>
<td>1.6M</td>
<td>1.6B</td>
<td>1.0T</td>
<td>6.1</td>
<td>18,617</td>
<td>2,358</td>
<td>237</td>
</tr>
<tr>
<td>ClueWeb</td>
<td>8.2B</td>
<td>102.4B</td>
<td>879.3B</td>
<td>358</td>
<td>failed</td>
<td>13,782</td>
<td>1,737</td>
</tr>
</tbody>
</table>
Experiments

- PaCiFier requires 195x less I/O than MapReduce methods, 35-65x less than MGT ($M = 256$ MB)

<table>
<thead>
<tr>
<th>Graph</th>
<th>RAM (MB)</th>
<th>GP</th>
<th>TTP</th>
<th>MGT</th>
<th>PaCiFier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yahoo (in GB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4,096</td>
<td>3,271</td>
<td>1,599</td>
<td>178</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>1,024</td>
<td>7,632</td>
<td>3,198</td>
<td>710</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>16,408</td>
<td>6,663</td>
<td>2,841</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>ClueWeb (in TB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4,096</td>
<td>68</td>
<td>28</td>
<td>8</td>
<td>0.9</td>
<td></td>
</tr>
<tr>
<td>1,024</td>
<td>142</td>
<td>56</td>
<td>31</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>291</td>
<td>114</td>
<td>125</td>
<td>1.9</td>
<td></td>
</tr>
</tbody>
</table>

- In ClueWeb with $M = 256$ MB, estimated time to finish I/O

<table>
<thead>
<tr>
<th>I/O Device</th>
<th>MGT</th>
<th>PaCiFier</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 GB/sec RAID</td>
<td>35 hrs</td>
<td>32 min</td>
</tr>
<tr>
<td>100 MB/sec HDD</td>
<td>&gt; 2 weeks</td>
<td>5.3 hrs</td>
</tr>
</tbody>
</table>
Thank you!
Any questions?

Contact: yicui@cse.tamu.edu