Computer Science, Texas A&M University

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 Δ_{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 Δ_{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

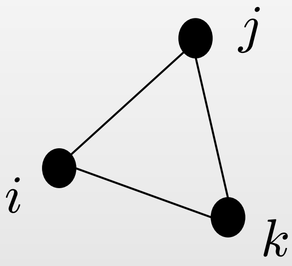
Background

- All prior work on creation of directed graphs can be unified by a two-step process
  - Relabeling: Shuffle nodes with some permutation θ, 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

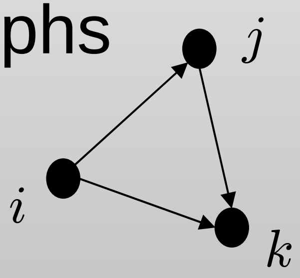
- Introduction
- Background
- Analysis
- Pruned Companion Files
- Implementation
- Experiments

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?

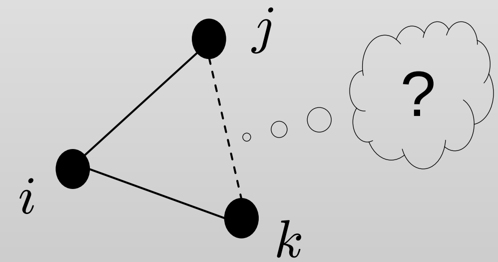
- There are six search orders in oriented graphs
  - For example: $i>j>k$ starts from the largest node, continues to the middle node, and finishes with the smallest
  - Some search orders visit only in-neighbors, some only out-neighbors, and others do both



- Interestingly, the search order coupled with permutation θ greatly affects CPU and I/O complexity!
  - Not formally observed or studied before

Generalized Iterators (GI)

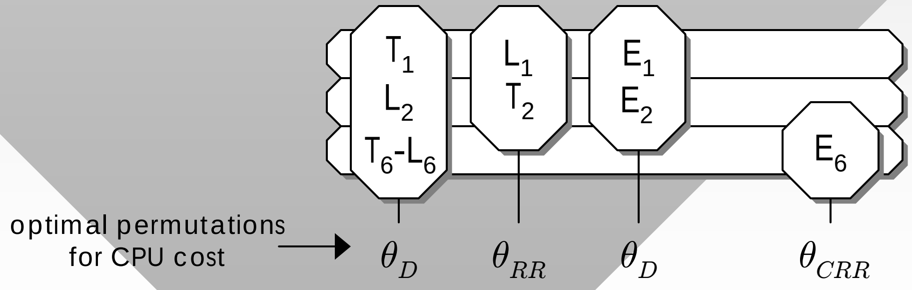
- To study this further, we propose a framework of 18 triangle-search techniques that subsumes all previous methods



- Generalized Vertex Iterator (GVI)
  - Methods T_1-T_6
- Generalized Lookup Edge Iterator (GLEI)
  - Methods L_1-L_6
- Generalized Scanning Edge Iterator (GSEI)
  - Methods E_1-E_6
- 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

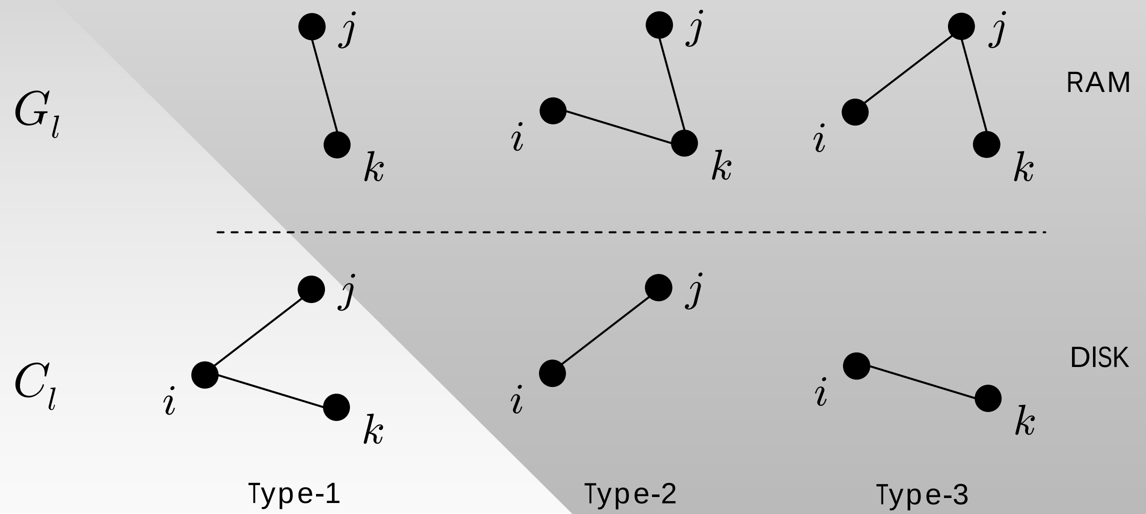
- Introduction
- Background
- Analysis
- Pruned Companion Files
- Implementation
- Experiments

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, …, V_p
  - Partition G into subgraphs G_1, G_2, …, 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)

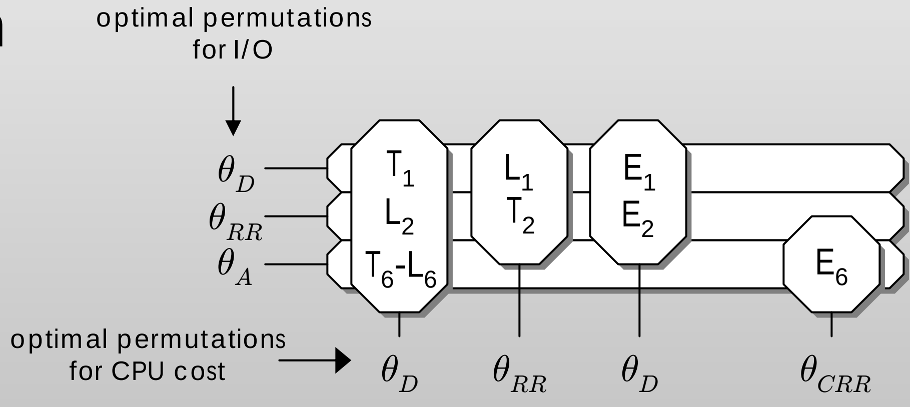
- For each G_l, we create a companion file C_l that contains the missing edges
  - The paper covers all 18 methods in one simple algorithm
  - Extra care is taken to minimize the size of C_l



- 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 θ 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

	RAM (MB)										
	1024	**512**	**256**	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
MGT	5.39	10.77	21.55	43.10	86.19	172.4	344.8	689.5	1379	2758	5516
Pagh	22.91	32.39	45.81	64.79	91.63	129.6	183.3	259.2	366.5	518.3	733.0
PCF	1.48	2.75	4.76	7.64	11.67	17.17	24.52	33.97	45.56	58.90	73.11

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 T_1 vs E_1
  - They have the same I/O, but CPU cost differs
  - T_1 has fewer operations, but they are inherently slower
  - Google hash table: 19M/sec
  - Naive scalar intersection: 264M/sec (14x faster)
- In real-world graphs, E_1 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., T_1 is always faster in the limit

Computer Science, Texas A&M University

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

	Speed (M/sec)
Branchless intersection	416
SIMD 32-bit intersection	1,119
SIMD 16-bit intersection	1,801

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

Agenda

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

Computer Science, Texas A&M University

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

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

Experiments

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

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

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

I/O Device	MGT	PaCiFier
1 GB/sec RAID	35 hrs	32 min
100 MB/sec HDD	> 2 weeks	5.3 hrs

Thank you!
Any questions?

Contact: yicui@cse.tamu.edu