The Input-Output Complexity of Triangle Enumeration

Rasmus Pagh
IT University of Copenhagen
pagh@itu.dk

Francesco Silvestri
University of Padova
silvest1@dei.unipd.it

Abstract
Our focus is on determining the input/output (I/O) complexity of triangle enumeration. Let E be the number of edges, $M < E$ the size of internal memory, and B the block size. We improve the I/O complexity to $O\left(E^{1.5}/(B\sqrt{M})\right)$ expected I/Os, which improves the previous bounds by a factor $\min(\sqrt{E/M}, \sqrt{M})$. We give cache-aware and cache-oblivious algorithms. We also show that they are I/O optimal by proving that any algorithm enumerating T distinct triangles must always use $\Omega\left(T/(B\sqrt{M})\right)$ I/Os.

Introduction

The problem

- Enumerate all triangles in a graph.

Some problems with triangles

- **Counting triangles**: compute the (approximate) number of triangles in a graph.
- **Listing triangles**: generate and store all triangles (i.e. , store on disk).
- **Enumerating triangles**: generate all triangles in a graph, but do not store them.

Enumerating vs listing

- Almost no difference in RAM model:
  - cost of generating a triangle ≈ cost of storing a triangle.
- Huge difference in the I/O model:
  - Memory may not contain all triangles and $\Omega(T/B)$ I/Os are required for storing T triangles.
  - Storing all triangles in a $\sqrt{E}$-clique requires $\Omega\left(E\sqrt{E}/B\right)$ I/Os.
  - The cost of storing triangles may be larger than the cost of generating triangles. Our work shows that all triangles can be generated in $\Theta\left(E\sqrt{E}/(B\sqrt{M})\right)$ I/Os.
- In many cases, we **don't need** to store triangles:
  - In database systems, pipelining of operations may not require to store intermediate results.
  - Other applications in which enumerating all triangles is a preprocessing step may not require to store all triangles.

Motivation

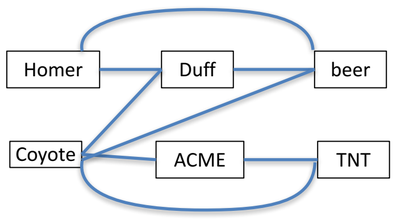
- Many algorithms for processing graphs often need to consider small subgraphs such as triangles.
- Examples: studying social processes in networks, community detection, solving systems of geometric constraints.
- Another example comes from database theory: A database is created to store information on sales people, and on the products types and brands they sell. We use three tables for guaranteeing the 5th normal form.

Sales person	Product
H. Simpson	Beer
W. Coyote	Beer
W. Coyote	TNT

Sales person	Brand
H. Simpson	DUFF
W. Coyote	ACME
W. Coyote	DUFF

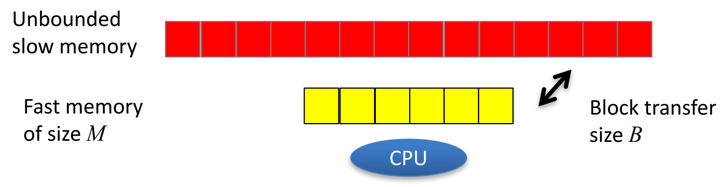
Product	Brand
Beer	DUFF
TNT	ACME

The joint of the three tables reduces to find all triangles in a graph: just view each table as a bipartite graph with vertices corresponding to attribute values.



The I/O model

- Input graphs are usually big and do not fit in internal memory.
- Use the **I/O model** [Vitter 2008] (also known as external memory model):
  - Consists of two memory levels.
  - A (potentially) unbounded slow memory, named external memory.
  - A fast memory of size M (in words), named internal memory.
  - Data are moved between the two levels in blocks of B words.
  - The CPU can only read and write data in the internal memory.
  - Complexity of an algorithm: number of I/O operations.



Cache-oblivious algorithm

- A cache-oblivious algorithm is an algorithm that does not use memory parameters M and B [Frigo et al., 1999]. Blocks are moved between the two memory levels by an automatic replacement policy (e.g., OPT, LRU).

Previous work

- All papers target the listing problem
- [Dementiev, PhD 2007] $O\left(\frac{E\sqrt{E}\log_{M/B}(E/B)}{B}\right)$ I/Os
- [Menegola, TR 2010] $O\left(E+\frac{E\sqrt{E}}{B}\right)$ I/Os
- [Hu et al., SIGMOD 2013] $O\left(\frac{E^2}{BM}+\frac{T}{B}\right)$ I/Os
  - Provide worst-case lower bound $\Omega\left(\frac{E\sqrt{E}}{B\sqrt{M}}+\frac{T}{B}\right)$ I/Os

Our results

- Optimal cache-aware algorithm requiring $O\left(\frac{E\sqrt{E}}{B\sqrt{M}}\right)$ I/Os in expectation.
- Optimal cache-oblivious algorithm requiring $O\left(\frac{E\sqrt{E}}{B\sqrt{M}}\right)$ I/Os in expectation.
- Derandomization of the cache-aware algorithm without asymptotically increasing the I/O complexity.
- Best-case lower bound: the enumeration of T triangles requires $\Omega\left(\frac{T}{B\sqrt{M}}\right)$ I/Os.

Cache-aware algorithm

Preliminaries

- Vertices are ordered (e.g. by ID)
- A triangle (v_1, v_2, v_3) is **(c_1, c_2, c_3)-colored** if:
  - $v_1 < v_2 < v_3$
  - v_1 has color c_1, v_2 has color c_2, v_3 has color c_3
- $E_{c1, c2}$: edges with colors c_1, c_2 ; $E_{c1, c3}$: edges with colors c_1, c_3; $E_{c2, c3}$: edges with colors c_2, c_3,

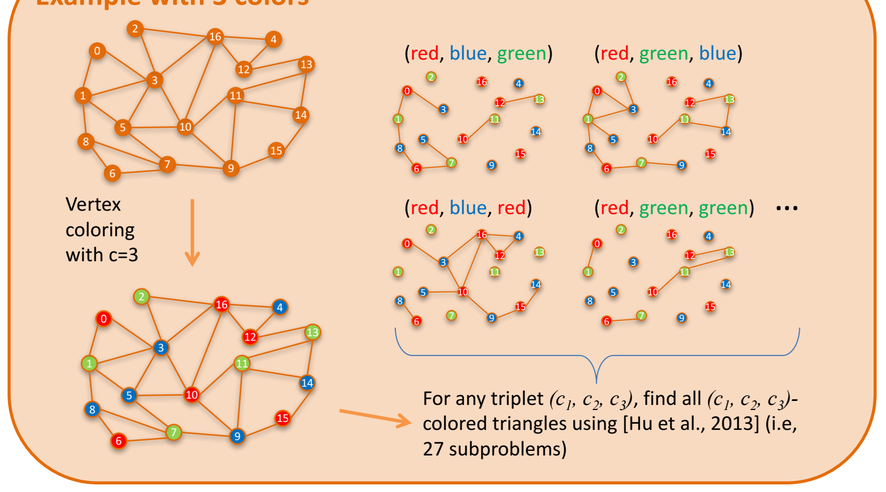
The cache-aware algorithm

1. Randomly color each vertex independently and uniformly with $c = \sqrt{E/M}$ colors
   - A triangle can be colored in c^3 ways
   - Vertexes can be colored using a 4-wise independent hash function
2. For each color triplet (c_1, c_2, c_3), with $0 \le c_1, c_2, c_3 < c$
   I. Consider edge sets compatible with the triplet (c_1, c_2, c_3), that is the sets $E_{c1, c2}$, $E_{c1, c3}$, $E_{c2, c3}$.
   II. Using [Hu et al., 2013], enumerate all (c_1, c_2, c_3)-colored triangles in the edge set $E_{c1, c2} \cup E_{c1, c3} \cup E_{c2, c3}$.

I/O complexity

- For each (c_1, c_2, c_3), we need: $E_{c1, c2}$, $E_{c1, c3}$, $E_{c2, c3}$ Each set has expected size M
- Expected subproblem size: $\mathbf{E}\left[E_{c_1,c_2}+E_{c_1,c_3}+E_{c_2,c_3}\right]=3M$
- Expected I/O of a subproblem $O\left(\frac{\left(E_{c_1,c_2}+E_{c_1,c_3}+E_{c_2,c_3}\right)^2}{BM}\right)=O(M/B)$
- Number of subproblems $c^3=\frac{E}{M}\sqrt{\frac{E}{M}}$
- **Total expected I/O:** $O\left(\frac{E}{B}\sqrt{\frac{E}{M}}\right)$

Example with 3 colors



Cache-oblivious Algorithm

1. If there are $O(1)$ edges, enumerate all triangles using a naïve approach.
2. Otherwise:
   I. Use the cache-aware algorithm using $c=2$ colors.
   II. Solve recursively the 8 coloring problems (each of about 1/4 size).

The algorithm has the same I/O complexity of the cache-aware algorithm: $O\left(\frac{E}{B}\sqrt{\frac{E}{M}}\right)$ I/Os in expectation.

Derandomization

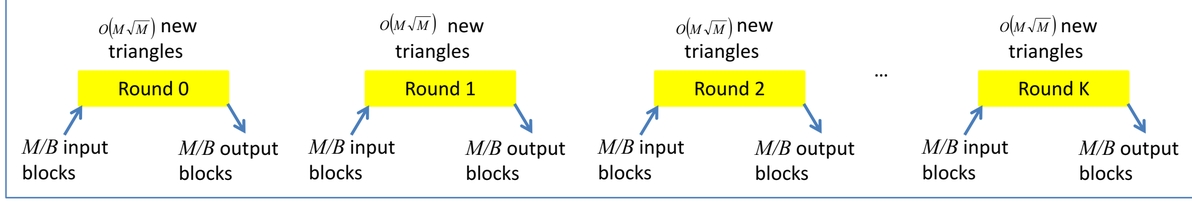
- The cache-aware algorithm can be derandomized, without increasing the I/O complexity, as follows.
- We use a small family of 4-wise independent functions [Alon et al., 1992]
- We fix the color of each vertex in $\log(E/M)$ iterations:
  - In each iteration, one bit of the coloring is fixed.
- In each iteration, we compute how well each function balances subproblems:
  - According to some "cost" function;
  - It can be proved that a "good" function exists in the family.

Lower bound

- Assumption: information on edges/vertices are indivisible: that is, for enumerating a triangle we need all its edges in memory at the same time.
- **Best-case lower bound**: applies to each input with T triangles and every possible algorithm execution.
- **Main lower bound**: the enumeration of T distinct triangles requires $\Omega\left(\frac{T}{B\sqrt{M}}\right)$ I/Os.
- The hardest graph is the $\sqrt{E}$-clique: $\Omega\left(\frac{E}{B}\sqrt{\frac{E}{M}}\right)$ I/Os are required.

Sketch of the proof

- Let A be the execution of an algorithm enumerating T triangles with M memory
- Let A' be the simulation of A on a memory of size $2M$ so that
  - A' can be decomposed in rounds
  - Each round starts with M/B inputs and ends with M/B outputs
  - Same asymptotic I/O complexity as A
- The simulation relies on the following ideas:
  - Half of the memory simulates the memory used by A
  - Half of the memory is used as buffer for I/Os
- In each round, there are M edges in memory:
  - $O(M\sqrt{M})$ triangles can be enumerated with M edges [Aftrati et al. 2013];
  - In each round, $O(M\sqrt{M})$ new triangles can be enumerated;
  - $\Omega(T/(M\sqrt{M}))$ rounds are required.
- Since each round requires M/B I/Os the lower bound follows



Acknowledgments

This work was supported by the Danish National Research Foundation under the Sapere Aude program, by MIUR of Italy under project AMANDA, and by the University of Padova under project CPDA121378.

References

[Aftrati et al. 2013] F. N. Afrati, A. D. Sarma, S. Salihoglu, J. D. Ullman. *Upper and lower bounds on the cost of a Map-Reduce computation*. Proc. VLDB Endow., 6(4):277–288, 2013.

[Alon et al., 1992] N. Alon, O. Goldreich, J. Hastad, R. Peralta. *Simple construction of almost k-wise independent random variables*. Random Struct. Algorithms, 3(3):289–304, 1992.

[Dementiev, 2007] R. Dementiev. *Algorithm engineering for large data sets: hardware, software, algorithms.* PhD thesis, Saarland University, 2007.

[Frigo et al., 1999] M. Frigo, C. E. Leiserson, H. Prokop, S. Ramachandran. *Cache-oblivious algorithms*. ACM Transactions on Algorithms, 8(1):4, 2012.

[Hu, et al., 2013] X. Hu, Y. Tao, C. W Chung. *Massive graph triangulation*. In Proc. SIGMOD, 325–336. ACM, 2013.

[Menegola, 2010] B. Menegola. *An external memory algorithm for listing triangles*. Technical report, Federal University of Rio Grande Sul, 2010.

[Vitter 2008] J.S. Vitter. *Algorithms and Data Structures for External Memory*. Now Publishers, 2008.

ACM SIGMOD/PODS Conference, June 22-27, Snowbird, Utah, USA.