Eric Cherny and Chaninder Rishi
DSC 190C: Algorithms in Big Data Due: Tuesday, June 7

Temporal Triangle Estimation, via Concentration Bounds

Abstract

Triangle counting contains research in multiple domains, for multiple different purposes, and with various kinds of constraints. This problem typically requires a different algorithm/different research based on the kind of graph and the kind of constraints one is working with.

The goal of this paper is to study the problem of counting triangles with respect to many different kinds of triangles, constraints, goals of analysis, and error tolerances in mind. The final point of this paper is to open questions for further research on how to estimate the number of temporal triangles.

We aggregate algorithm TETRIS for estimating triangles in static graphs and an algorithm DOTTT for counting the exact number of temporal triangles in temporal graphs. By utilizing TETRIS we are able to achieve under 5% error by just looking at 3% of the graph's vertices. We apply TETRIS logic to DOTTT to produce a different temporal triangle estimation algorithm.

As a result, we open thought for future research on how can we can merge TETRIS and DOTTT to design an algorithm for estimating temporal triangles, rather than precisely counting them. If successful, this new algorithm would produce a faster and an even more generalizable temporal triangle counting algorithm than the ones currently available in research.

The final paper on counting homomorphisms is a vital concept in counting graphs, but does not directly make it to help open research for our new algorithm because its scope is slightly different. We sparsely include it in this paper for analysis of counting triangles, but we do not refer to is as often as the first two described papers on TETRIS and DOTTT algorithms.

1 Introduction

Social network analysis refers to a broad field of studying relations between individuals, groups and communities to help us understand how and why these networks behave certain ways/how interactions within the networks occur. Counting triangles specifically helps us understand how networks evolve (for instance, in the computation of transitivity) as well as important parameters of social network analysis like identifying clustering coefficients.

1.1 Application and Contribution of TETRIS

Here we analyze the contributions TETRIS algorithm that estimates the number of triangles in a static graph with 5% error by looking at just 3% of the graph.

Given full access to the graph, using the brute force solution requires $O(m^{3/2})$ time complexity where m is number of edges. Using degree-based ordering, we can cut down the time complexity to $O(m\alpha)$ where α is the degeneracy of graph G. Many of the above mentioned algorithms require full access to the graph or access to some parameters of parts of the graph. TETRIS does not require access to the entire graph, or parameters, and its time (and space and query) complexity is sublinear. MCMC-based algorithms, that are an exception to the above and don't require access to parameters nor the full graph, and can be re-implemented similar to TETRIS to use concentration bounds. However, TETRIS performs with better consistency.

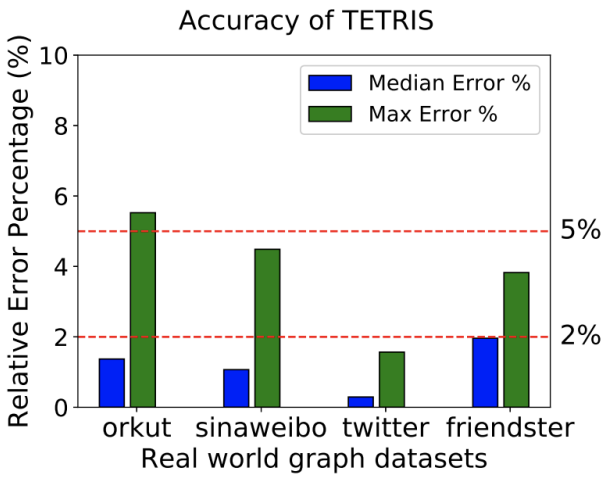


Figure 1: TETRIS accuracy across 100 runs per each of four networks

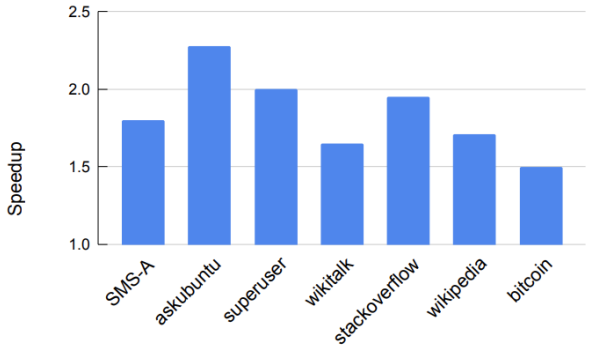
We can observe from the figure above that over 400 runs of the algorithm on different networks, the worst error TETRIS received was slightly above 5%. MCMC-based algorithms cannot achieve the same consistency. TETRIS above all other algorithms consistently achieves low error.

1.2 Application and Contribution for DOTTT

There are 8 types of directed triangles and DOTTT counts all, exactly, with only a logarithmic overhead over static counters, while also widening the constraints from the previous fastest performing model, namely PBL.

Figure 2 displays the speedup of the DOTTT algorithm in comparison to the previous best, PBL.

Figure 3 displays the increased expressivity, more precisely defining the different ratios of delta temporal triangles to portray the possibility of this kind of analysis by passing in



(a) Speedup

Figure 2: DOTTT vs. PBL speedup

various kinds of delta as opposed to PBL's constraint (δ, δ, δ).

2 Introduction to Algorithms

This is where we have graphs showing results of these algorithms and how these results differ in comparison to other algorithms.

2.1 TETRIS Algorithm Brief

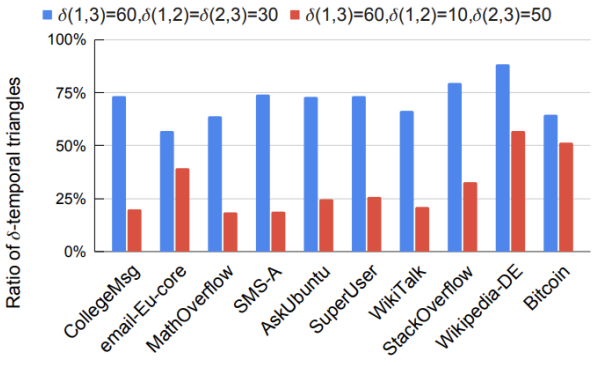
TETRIS is an acronym for Triangle Estimation Through Random Incidence Sampling. Let G be a graph G(V,E) that is undirected and static. We define t_e to be the number of triangles that contain an edge e, and define t_{max} as the maximum t_e across the entire graph.

Figure 4 shows the upper bound for the number of queries the TETRIS algorithm performs. The algorithm is sublinear because the terms l_{mix} and t_{max} are considered both necessary and categorically smaller than T.

Unlike other methods that have problems with the random access model that implement inefficient wedge sampling, we order the vertices by degree and then sample the edges proportional to the degree of the smaller degree endpoint (minimum vertex). This strategy biases away from high-degree vertices and speeds up wedge sampling.

The algorithm is an implementation of a random walk starting from an arbitrary seed vertex that we have access to (because we assume to not have access to the entire graph). The algorithm essentially grabs a sample of a sample: the first sample is the edges sampled throughout the random walk performed with time l_{mix} per edge sample, and the next is a sampling of l edges proportional to each edge's minimum degree.

To prove that the statistical properties from the random walk are passed onto the statistical properties of the original graph, we use carefully constructed concentration bounds.



(b) Expressivity

Figure 3: DOTTT vs. PBL expressivity

$$O\left(\log\left(\frac{1}{\delta}\right)\frac{\log n}{\epsilon^2}\left(\frac{m\,\ell_{\text{mix}}\,t_{\max}}{T}+\frac{m\alpha}{T}+\sqrt{m}\,\ell_{\text{mix}}\right)\right)$$

Figure 4: TETRIS query bound

2.2 DOTTT Algorithm Brief

DOTTT - degeneracy oriented temporal triangle totaler - deals with counting temporal triangles in temporal, directed graphs.

In temporal graphs, the edges come with timestamps and thereby yield more information than static graphs that come without timestamps. Temporal triangles are defined different based on each scenario, but in this paper they are defined as a constrains on the time difference between any two edges, formulated like so $(\delta_{13}, \delta_{12}, \delta_{23})$.

Using degeneracy ordering and combinatorial arguments in its proof, DOTTT is able to count triangles in $O(m\kappa * log(m))$ time, where m refers to the number of temporal edges (hence with the possibility of a multiplicity where multiple edges occur between the same two nodes differentiated by the timestamp differences) and κ defines the graph degeneracy (the maximum k-core explained in the preliminaries). This time complexity matches the time of the state-of-the-art static triangle counting algorithms while being able to count directed temporal triangles with constraints (thus, a more difficult setting than that in static triangles).

Counting temporal triangles in temporal directed networks can be different depending on the network and the goal because of the various constraints temporal triangles are exposed to (further covered in preliminaries).

With the above in mind, the purpose of DOTTT is two things:

1. Make a generalizable algorithm to cover all (or as many, as possible) constraints;
2. Push the algorithm closer to that of static triangles.

3 Preliminaries

These are the concepts we need to understand before attempting to understand how each algorithm is implemented and the proofs behind those implementations.

These are also the necessary terms for us to dive into algorithms and proofs.

3.1 K-core, Degeneracy and Degeneracy Ordering

In this subsection, we'll briefly explain a few concepts that are pertinent to understanding the papers that follow. Imagine we have a graph G where every node has a degree. The k-core is a graph that we generate by taking the original graph G and removing every node with a degree less than k. We iterate this removal process on the new subgraph that we generate until the graph converges (essentially, until the graph stops losing nodes). It's important to note that the only way a node actually stays as a k-core is if that k number of nodes is interconnected. The image below shows an input graph (left) and an output graph (right) which is the resulting graph (3-core) after we iteratively remove all nodes with a degree less than 3.

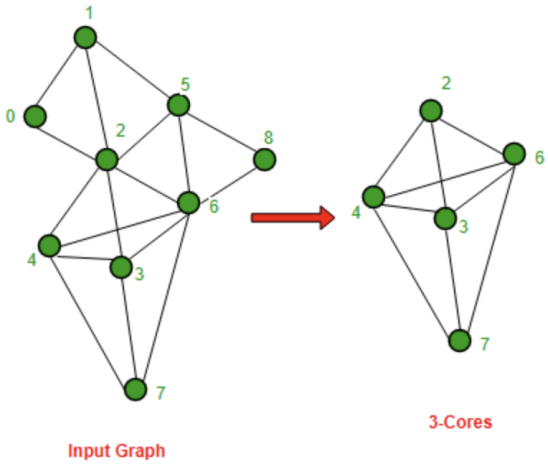


Figure 5: Example of a graph generated by K-core

Degeneracy is the maximum k-core that we can get without losing the entire graph. After we find the desired k-core of the graph, degeneracy ordering is the ordering of nodes we get after removing the vertex with the minimum degree in the remaining subgraph. To reiterate here, the degeneracy of the smallest value we can pick for k in k-core such that every node in our remaining graph still has at most k neighbors in the ordering.

3.2 Directed Temporal Networks and Temporal Triangles

Directed temporal networks are networks that are directed (edges with an arrow) and temporal meaning that edges come with a timestamp. Timestamped edges means that edges

have 3 identifiers as opposed to 2: vertex 1, vertex 2, and timestamp.

Algorithms that count temporal triangles are typically specialized to the constraints of the triangle, and we discussed in an earlier section how the DOTTT algorithm places constraint on triangles. In other algorithms temporal triangles may be related to the sequence of prior events and sequence of timestamps rather than cumulative or partial difference in each edge's timestamps.

4 Algorithms and Proofs

4.1 TETRIS Intuition and Algorithm

We can see the simplified TETRIS algorithm implementation in Figure 3.

Algorithm 1 TETRIS— Triangle Counting Estimator

procedure TETRIS(integer r, integer ℓ, integer $\widehat{\ell_{\text{mix}}}$)
  Let s be some arbitrary vertex provided by the model.
  Let R be the multiset of edges on a r-length random walk from s.
  for $i = 1$ to ℓ **do**
    Sample an edge $e \in R$ independently with prob. d_e/d_R.
    Query a uniform random neighbor w from $N(e)$.
    Using edge query, check if $\{e, w\}$ forms a triangle.
    If $\{e, w\}$ forms a triangle τ: query for degrees of all vertices in τ and determine if τ is associated to e.
    If τ is associated to e, set $Y_i = 1$; else set $Y_i = 0$.
  Set $Y = \frac{1}{\ell} \sum_{i=1}^{\ell} Y_i$.
  Let $\overline{m}$ = EdgeCountEstimator $(R, \widehat{\ell_{\text{mix}}})$
  Set $X = \frac{\overline{m}}{r} \cdot d_R \cdot Y$.
  return X.

Algorithm 2 Edge Count Estimator

procedure EdgeCountEstimator(edge set R, integer $\widehat{\ell_{\text{mix}}}$)
  for $i = 1$ to $\widehat{\ell_{\text{mix}}}$ **do**
    Let $R_i = (e_i, e_{i+\widehat{\ell_{\text{mix}}}}, e_{i+2\widehat{\ell_{\text{mix}}}}, \ldots)$.
    Set c_i =number of pairwise collision in R_i.
    Set $Y_i = \binom{|R_i|}{2}/c_i$.
  Set $Y = \frac{1}{\widehat{\ell_{\text{mix}}}} \sum_i Y_i$.
  return Y.

Figure 6: TETRIS General Implementation

The TETRIS algorithm takes in three parameters other than information from the input graph G:

1. Length of random walk r
2. Number of subsamples l
3. Estimate of l_{mix}, the mixing time

This algorithm assumes that we do not know the full graph, so we read in one arbitrary (not uniformly random) starting vertex. The goal of TETRIS is to minimize the number of queries.

For TETRIS implementation we begin at an arbitrary vertex and perform a random walk of length r; in the process of the random walk we collect the edges R. Next, we compute the degree of each edge e in R. With the degrees computed and stored, we now sample l edges from R where each edge is sampled with probability proportional to the degree of the edge (the minimum endpoint degree).

Now to recap, we have a set of edges of size l, sampled proportional to the degree of the lower degree endpoint. The algorithm now continues and samples a uniform random neighbor from the lower degree endpoint. Using an edge query, if the algorithm does not discover a triangle then it skips, if an algorithm does discover a triangle then we check if the edge found is a unique edge forming the triangle.

Finally, we make use of a collision-based estimator that estimates the total number of edges; designed in a paper by Ron and Tsur.

4.2 DOTTT Intuition and Algorithm

Define deltas (d_13, d_12, d_23)
Extract G(V, E_s) – static – from T(V, E) – temporal
Get degeneracy ordering of G
Orient edges G with respect to the degeneracy ordering to get DAG G Matula and Beck
Enumerate static triangles from DAG G (so that triangles are acyclic) O($m_s * k$), so far
m_s is number of static edges in G
k is degeneracy
In this DAG for any edge (u, v)
Enumerate the neighbors W of v
Check for each w if it connects with u (w, u) in graph T
When u is the source node:
Number of (u, v) and (v, w) temporal edges is limited by out degree of u Out degree of u is bounded by degeneracy k
(u, w) is not bounded by k, however, so we avoid enumerating it

4.3 Counting Homomorphisms

H-homomorphisms can be counted in near-linear time in bounded degeneracy graphs in certain cases. This paper defines the exact cases in which counting H-homomorphisms in bounded degeneracy graphs takes near-linear time.

Let's consider a pattern graph, H. We want to take into consideration every possible acyclic orientation of H. For each of those orientations, we want to compute the amount of H-homomorphisms. We then sum up the calculated number of H-homomorphisms over all the orientations we just referenced. What we can do now is create a tree and since all these

orientations have bounded out-degrees, the main idea here is to index the tree in such a way that we can retrieve the H-homomorphism counts in linear time.

Without diving too deep into the convoluted theorem behind it, we can conduct some lower bound analysis on this proposed algorithm. Let's take an orientation of the pattern graph and call it G. We choose P to be some arbitrary partition that splits G's vertices into k different sets. We're now dealing with partitioned-homomorphisms, which are essentially just H-homomorphism but each vertex in the orientation graph is mapped to a different set among the k sets that we split G's vertices among. The paper proposes that we choose an efficient P such that the problem of counting triangles in G essentially boils down to computing the count of partitioned homomorphisms times some constant value. The theorem tells us that we will eventually end up counting 2^k instances of individual homomorphism counts, where k is the number of sets we split our graph G among.

Conjecture 1.2 (Triangle Detection Conjecture [AW14]). *There exists a constant $\gamma > 0$ such that in the word RAM model of $O(\log n)$ bits, any algorithm to detect whether an input graph on m edges has a triangle requires $\Omega(m^{1+\gamma})$ time in expectation.*

Figure 7: TETRIS query bound

Theorem 1.1. *Let G be an input graph with n vertices, m edges, and degeneracy κ. Let $f : \mathbb{N} \to \mathbb{N}$ denote some explicit function. Let $\gamma > 0$ denote the constant from the* Triangle Detection Conjecture.

If $\mathrm{LICL}(H) \leqslant 5$: there exists an algorithm that computes $\mathrm{Hom}_H(G)$ in time $f(\kappa) \cdot m \log n$.

If $\mathrm{LICL}(H) \geqslant 6$: assume the Triangle Detection Conjecture. *For any function $g : \mathbb{N} \to \mathbb{N}$, there is no algorithm with (expected) running time $g(\kappa)o(m^{1+\gamma})$ that computes $\mathrm{Hom}_H(G)$.*

Figure 8: TETRIS query bound

5 Open Questions and Further Research

This is where we consider the possibility of a new and better algorithm and open other questions/possibilities.

6 References

[01] Mohammad Al Hasan, Vachik Dave, Triangle Counting in Large Networks: A Review, Department of Computer Science, IUPUI, IN 46202
[02] Mitchell Telatnik, How To Get Started with Social Network Analysis, A Complete Beginner's Guide to Getting Up and Running Making Beautiful Network Graphs, May 27, 2020
[03] Noujan Pashanasangi, C. Seshadhri, Faster and Generalized Temporal Triangle Counting, via Degeneracy Ordering,
[04] Suman K. Bera, C. Seshadhri, How to Count Triangles, without Seeing the Whole Graph
[05] Suman K. Bera, Noujan Pashanasangi, C. Seshadhri, University of California, Santa Cruz, Near-Linear Time Homomorphism Counting in Bounded Degeneracy Graphs: The Barrier of Long Induced Cycles