Decompositions of Triangle-Dense Graphs

Rishi Gupta*
Stanford University
rishig@cs.stanford.edu

Tim Roughgarden†
Stanford University
tim@cs.stanford.edu

C. Seshadhri
Sandia National Labs, CA‡
scomand@sandia.gov

Abstract

High triangle density — the graph property stating that most two-hop paths belong to a triangle — is a common signature of social networks. This paper studies triangle-dense graphs from a structural perspective. We prove constructively that most of the content of a triangle-dense graph is contained in a disjoint union of radius 2 dense subgraphs. This result quantifies the extent to which triangle-dense graphs resemble unions of cliques. We also show that our algorithm recovers planted clusterings in stable k-median instances.

1 Introduction

Can the special structure possessed by social networks be exploited algorithmically? Answering this question requires a formal definition of "social network structure." Extensive work on this topic has generated countless proposals but little consensus (see e.g. [CF06]). The most oft-mentioned (and arguably most validated) statistical properties of social networks include heavy-tailed degree distributions [BA99, BKM+00, FFF99], a high density of triangles [WS98, SCW+10, UKBM11] and other dense subgraphs or "communities" [For10, GN02, New03, New06, LLDM08], and low diameter and the small world property [Kle00a, Kle00b, Kle01, New01].

Much of the recent mathematical work on social networks has focused on the important goal of developing generative models that produce random networks with many of the above statistical properties. Well-known examples of such models include preferential attachment [BA99] and related copying models [KRR+00], Kronecker graphs [CZF04, LCK+10], and the Chung-Lu random graph model [CL02b, CL02a]. A generative model articulates a hypothesis about what "real-world" social networks look like, and is directly useful for generating synthetic data. Once a particular generative model of social networks is adopted, a natural goal is to design algorithms tailored to perform well on the instances generated by the model. It can also be used as a proxy to study the effect of random processes (like edge deletions) on a network. Examples of such results include [AJB00, LAS+08, MS10].

*Supported in part by the ONR PECASE Award of the second author.

†This research was supported in part by NSF Awards CCF-1016885 and CCF-1215965, an AFOSR MURI grant, and an ONR PECASE Award.

‡Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

This paper pursues a different approach. In lieu of adopting a particular generative model for social networks, we ask:

> *Is there a combinatorial assumption weak enough to hold in every "reasonable" model of social networks, yet strong enough to permit useful structural and algorithmic results?*

That is, we seek algorithms that offer non-trivial guarantees for *every* reasonable model of social networks, including those yet to be devised.

Triangle-Dense Graphs

We initiate the algorithmic study of *triangle-dense graphs*.

Definition 1 (Triangle-Dense Graph)**.** *The* triangle density *of an undirected graph $G = (V, E)$ is $\tau(G) := 3t(G)/w(G)$, where $t(G)$ is the number of triangles in G and $w(G)$ is the number of wedges (i.e., two-hop paths) in G. The class of* ϵ-triangle dense graphs *consists of the graphs G with $\tau(G) \geq \epsilon$.*

Since every triangle of a graph contains 3 wedges, and no two triangles share a wedge, the triangle density of a graph is between 0 and 1.[1] We use the informal term "triangle dense graphs" to mean graphs with constant triangle density.

For example, the triangle density of a graph is 1 if and only if it is the union of cliques. The triangle density of an Erdös-Renyi graph, drawn from $G(n, p)$, is concentrated around p. Thus, only very dense Erdös-Renyi graphs have constant triangle density. Social networks are generally sparse and yet have remarkably large triangle density; the Facebook graph, for instance, has triangle density 0.16 [UKBM11]. High triangle density is perhaps the least controversial signature of social networks (see related work below).

The class of ϵ-triangle dense graphs becomes quite diverse as soon as ϵ is bounded below 1. For example, the complete tripartite graph is triangle dense. Every graph obtained from a bounded-degree graph by replacing each vertex with a triangle is triangle dense. Any graph obtained by disjointly adding an $n^{1/3}$ vertex clique to an n-vertex graph is triangle dense. We give a litany of examples in §4. Can there be interesting structural or algorithmic results for this rich class of graphs?

Our Results: A Decomposition Theorem

Our main decomposition theorem quantifies the extent to which a triangle-dense graph resembles a union of cliques. The next definition gives our notion of an "approximate union of cliques." In it, we use $G|_S$ to denote the subgraph of a graph G induced by a subset S of vertices. Also, the *edge density* of a graph $G = (V, E)$ is $|E|/\binom{|V|}{2}$.

Definition 2 (Tightly Knit Family)**.** *Let $\rho > 0$. A collection $V_1, V_2, \ldots, V_k$ of disjoint sets of vertices of a graph $G = (V, E)$ forms a* ρ-tightly-knit family *if:*

- *Each subgraph $G|_{V_i}$ has both edge density and triangle density at least ρ.*
- *Each subgraph $G|_{V_i}$ has a radius at most 2.*

[1] In the social sciences, triangle density is usually called the *transitivity* of a graph [WF94]. We use triangle density because "transitivity" already has strong connotations in graph theory.

When ρ is a constant, we often refer simply to a tightly-knit family. Every "cluster" of a tightly-knit family is dense in edges and in triangles. In the context of social networks, an abundance of triangles is generally associated with meaningful social structure.

Our main decomposition theorem states that every triangle-dense graph contains a tightly-knit family that captures a constant fraction of graph's triangles. (The existence is stated in Theorem 13, and the algorithm is presented in §6.)

Theorem 3 (Main Decomposition Theorem)**.** *Consider a triangle-dense graph $G = (V, E)$ with w wedges. There exists a tightly-knit family that contains a constant fraction of the triangles of G. This family can be found on $O(|V| + |E| + w)$ time.*

We emphasize that Theorem 3 requires only that the input graph G is triangle dense — beyond this property, it could be sparse or dense, low- or high-diameter, and possess an arbitrary degree distribution. Graphs that are not triangle dense, such as sparse Erdös-Renyi random graphs, do not generally admit non-trivial tightly-knit families (even if the triangle density requirement for each cluster is dropped).

Requiring that the collection of disjoint clusters preserves much of the "interesting social information" of the original graph, in the form of the graph's triangles, makes Theorem 3 non-trivial. Extracting a single low-diameter cluster rich in edges and triangles is easy — large triangle density implies that typical vertex neighborhoods have these properties. But extracting such a cluster carelessly can do more harm than good, destroying many triangles that only partially intersect the cluster. Our proof of Theorem 3 shows how to repeatedly extract low-diameter dense clusters while preserving at least a constant fraction of the triangles of the original graph.

A triangle-dense graph need not contain a tightly-knit family that contains a constant fraction of the graph's *edges*; see the examples in §4. The culprit is that triangle density is a "global" condition and does not guarantee good local triangle density everywhere, allowing room for a large number of edges that are intuitively spurious. Under the stronger condition of constant local triangle density, however, we can compute a tightly-knit family with a stronger guarantee.

Definition 4 (Jaccard Similarity)**.** *The* Jaccard similarity *of an edge $e = (i, j)$ of a graph $G = (V, E)$ is the fraction of wedges including e that participate in triangles:*

$$J_e = \frac{|N(i) \cap N(j)|}{|N(i) \cup N(j) \setminus \{i, j\}|},$$

where $N(x)$ denotes the neighbors of a vertex x in G.

Definition 5 (Everywhere Triangle-Dense)**.** *A graph is* everywhere ϵ-triangle dense *if $J_e \geq \epsilon$ for every edge e.*

Observe that an everywhere ϵ-triangle dense graph is ϵ-triangle dense. The following is proven as Theorem 14.

Theorem 6 (Stronger Decomposition Theorem)**.** *Consider an everywhere triangle-dense graph $G = (V, E)$ with w wedges. There exists a tightly-knit family that contains a constant fraction of the* edges and *triangles of G. This family can be found in $O(|V| + |E| + w)$ time.*

Applications to Planter Cluster Models. We give an algorithmic application of our decomposition in §5, where the tightly knit family produced by our algorithm is meaningful in its own right. We consider the stable metric k-median instances introduced by Balcan et al. [BBG09].

Every solution of a stable instance that has near-optimal objective function value is structurally similar to the optimal solution. They reduce their problem to clustering a certain graph with "planted" clusters corresponding to the optimal solution. We prove that our algorithm recovers a close approximation to the the planted clusters, matching their guarantee.

1.1 Discussion

Structural Assumptions vs. Generative Models. Pursuing structural results and algorithmic guarantees that assume only a combinatorial condition (namely, constant triangle density), rather than a particular model of social networks, has clear advantages and disadvantages. The class of graphs generated by a specific model will generally have stronger structural and algorithmic properties than the class of graphs that share a single statistical property. On the other hand, algorithms and results tailored to a single model can lack robustness: they might not be meaningful if reality differs from the model, and are less likely to translate across different application domains which require different models. Our results for triangle-dense graphs are relevant for every model of social networks that generates such graphs with high probability, and we expect that all future social network models will have this property. And of course, our results can be used in any application domain that concerns triangle-dense graphs, whether motivated by social networks or not.

Beyond generality and robustness, a second reason to prefer a combinatorial assumption to a generative model is that the assumption can be easily verified for a given data set. Since computing the triangle density of a network is a well-studied problem, both theoretically and practically (see [WW10, SPK13] and the references therein), the extent to which a network meets the triangle density assumption can be quantified. By contrast, it is not clear how to argue that a network is a typical instance from a generative model, other than by verifying various statistical properties (such as triangle density). This difficulty of verification is amplified when there are multiple generative models vying for prominence, as is currently the case with social and information networks (e.g. [CF06]).

Why Triangle Density? Social networks possess a number of statistical signatures, as discussed above. Why single out triangle density? First, there is tremendous empirical support for large triangle density in social networks. This property has been studied for decades in the social sciences [HL70, Col88, Bur04, Fau06, FWVDC10], and recently there have been numerous large-scale studies on online social networks [SCW+10, UKBM11, SPK13]. Second, in light of this empirical evidence, generative models for social and information networks are explicitly designed to produce networks with high triangle density [WS98, CF06, SCW+10, VB12]. Third, the assumption of constant triangle density seems to impose more exploitable structure than the other most widely accepted properties of social and information networks. For example, the property of having small diameter indicates little about the structure of a network — every network can be rendered small-diameter by adding one extra vertex connected to all other vertices. Similarly, merely assuming a power-law degree distribution does not seem to impose significant restrictions on a graph [FPP06]. E.g. the Chung-Lu model [CL02b] generates power-law graphs with no natural decomposition. While constant triangle density is not a strong enough assumption to exclude all "obviously unrealistic graphs," it nevertheless enables non-trivial decomposition results. Finally, we freely admit that imposing one or more combinatorial conditions other than triangle density could lead to equally interesting results, and we welcome future work along such lines. Indeed,

recent work by Ugander et al [UBK13] shows empirical evidence of how subgraph frequencies are constrained in social networks.

Why Tightly-Knit Families? We have intentionally highlighted the existence and computation of tightly-knit families in triangle-dense graphs, rather than the (approximate) solution of any particular computational problem on such graphs. Our main structural result quantifies the extent to which we can "visualize" a triangle-dense graph as, approximately, a union of cliques. This is a familiar strategy for understanding restricted graph classes, analogous to using separator theorems to make precise how planar graphs resemble grids [LT79], tree decompositions to quantify how bounded-treewidth graphs resemble trees [RS86], and the regularity lemma to describe how dense graphs are approximately composed of "random-like" bipartite graphs [Sze78]. Such structural results provide a flexible foundation for future algorithmic applications. We offer a specific application to recovering planted clusterings and leave as future work the design of more applications.

2 An intuitive overview

We give an intuitive description of our proof. Our approach to finding a tightly-knit family is an iterative extraction procedure. We find a single member of the family, remove this set from the graph (called the extraction), and repeat. Let us start with an everywhere triangle-dense graph G, and try to extract a single set S. It is easy to check that every vertex neighborhood is dense and has many triangles, and would qualify as a set in a tightly-knit family. But for vertex i, there may be many vertices outside $N(i)$ (the neighborhood of i) that form triangles with a single edge contained in $N(i)$. By extracting $N(i)$, we could destroy too many triangles. We give examples in §4 where such a naïve approach fails.

Let us try a simple greedy fix to the procedure. We start by adding $N(i)$ and i to the set S. If any vertex outside $N(i)$ forms many triangles with the edges in $N(i)$, we just add it to S. It is not clear that we solve our problem by adding these vertices to S, since the extraction of S could still destroy many triangles. We prove that by adding up to d_i vertices (where d_i is the degree of i) with the highest number of triangles to $N(i)$, we can ensure that this does not happen. In other words, $G|_S$ will have a high density, obviously has radius 2 (from i), and will contain a constant fraction of the triangles incident to S.

Naturally, we can simply iterate this procedure and hope to get the entire tightly-knit family. But there's a small catch. We crucially needed the graph to be *everywhere* triangle-dense for the previous argument. After extracting S, this holds no more. So we employ a *cleaning* procedure that iteratively removes edges of low Jaccard similarity and produces an everywhere triangle-dense graph for the next extraction. This procedure also destroys some triangles, but we can upper bound this number. As an aside, removing low Jaccard similarity edges has been used for sparsifying real-world graphs by Satuluri et al [SPR11].

When we start with a triangle-dense graph G, we begin by cleaning the graph to get an everywhere triangle-dense graph. We may lose many edges during the initial cleaning, and this is inevitable, as examples in §4 show. However, putting it all together, this procedure (constructively) proves the existence of a tightly-knit family containing a constant fraction of the triangles in a triangle-dense graph.

When G is everywhere triangle-dense, we can ensure that the tightly-knit family contains a constant fraction of the *edges* as well. At the beginning (in G), every edge is incident to many triangles. We know that our final tightly-knit family has a constant fraction of the triangles, so it intuitively appears that too many edges could not have been removed. This is tricky to argue, since the degrees of the G can be arbitrary. The actual proof requires an intricate charging argument. By assigning an appropriate weight function to triangles and wedges, we can charge removed edges to removed triangles. This (constructively) proves the existence of a tightly-knit family with a constant fraction of edges and triangles.

3 Extracting tightly-knit families

In this section we walk through the proof outlined in §2 above. We first bound the losses from the cleaning procedure in §3.2. We then show how to extract a member of a tightly-knit family from a cleaned graph in §3.3. We combine these two procedures in Theorem 13 of §3.4 to obtain a full tightly-knit family from a triangle-dense graph. Finally, Theorem 14 of §3.5 shows that the procedure also preserves a constant fraction of the edges in an everywhere triangle-dense graph.

3.1 Preliminaries

We begin with some notation. Consider a graph $G = (V, E)$. We index vertices with $i, j, k, \ldots$. Vertex i has degree d_i. We use $w_i = \binom{d_i}{2}$ to denote the number of wedges where i is the middle vertex, and t_i for the number of triangles containing i. For edge $e = (i, j)$, we denote $w_i = d_i + d_j - 2$ to be the number of wedges including e, and t_e to be the number of triangles. As mentioned earlier, the triangle density τ is $3t/w$. We define the local version (also called clustering coefficients [WS98]) $\tau_i = t_i/w_i$. The Jaccard similarity J_e is t_e/w_e (conventionally, if $w_e = 0$, J_e is set to 0.)

Let S be a set of vertices. The number of triangles including some vertex in S is denoted t_S. We use $G|_S$ for the induced subgraph on G, and $t_S^{(I)}$ for the number of triangles in $G|_S$ (the I is for "internal"). We will be repeatedly dealing with subgraphs H of G. We use the $\ldots(H)$ notation for the respective quantities in H. So, $t(H)$ would denote the number of triangles in H, $d_i(H)$ denotes the degree of i in H, etc.

3.2 Cleaning a graph

It will be convenient to have a "cleaning" procedure that constructs an everywhere triangle-dense graph.

Definition 7. *Consider the following procedure $clean_\epsilon$ on a graph H that takes input $\epsilon \in (0, 1]$. Iteratively remove any edge with Jaccard similarity less than ϵ, as long as such an edge exists. Finally remove all isolated vertices. This is called* ϵ-cleaning. *Abusing notation, the output is denoted by $clean_\epsilon(H)$.*

Satuluri et al [SPR11] use a more nuanced version of cleaning for graph sparsification of social networks. They provide much empirical evidence that removal of low Jaccard similarity edges does not affect graph structure. Our arguments below may provide some theoretical justification.

Claim 8. *The number of triangles in $clean_\epsilon(H)$ is at least $t(H) - \epsilon w(H)$.*

Proof. The process $clean_\epsilon$ removes a sequence of edges $e_1, e_2, \ldots$. Let W_l and T_l be the set of wedges and triangles that are removed when e_l is removed. Since the Jaccard similarity of e_l *at this stage* is at most ϵ, $|T_l| \leq \epsilon(|W_l| - |T_l|) \leq \epsilon|W_l|$. All the W_ls (and T_ls) are disjoint. Hence, the total number of triangles removed is $\sum_l |T_l| \leq \epsilon \sum_l |W_l| = \epsilon w(H)$. □

We get an obvious corollary by noting that $t(H) = \tau(H) \cdot w(H)/3$.

Corollary 9. *The graph $clean_\epsilon(H)$ is everywhere ϵ-triangle dense and has at least $(\tau(H)/3-\epsilon)w(H)$ triangles.*

We also state a simple lemma on the properties of everywhere triangle-dense graphs.

Lemma 10. *If H is everywhere ϵ-triangle dense, then $d_i \geq \epsilon\, d_j$ for every edge (i,j). Furthermore, $N(i)$ is ϵ-edge dense for every vertex i.*

Proof. If $d_i \geq d_j$ we are done. Otherwise

$$\epsilon \leq J_{(i,j)} = \frac{|N(i) \cap N(j)|}{|(N(i) \setminus \{j\}) \cup (N(j) \setminus \{i\})|} \leq \frac{d_i - 1}{d_j - 1} \leq \frac{d_i}{d_j}$$

as desired. Let $S = N(i)$. The number of edges in S is at least

$$\frac{1}{2}\sum_{j \in S} |N(i) \cap N(j)| \geq \frac{1}{2}\sum_{j \in S} J_{(i,j)}(d_i - 1) \geq \frac{\epsilon d_i(d_i-1)}{2} = \epsilon \binom{d_i}{2}.$$

□

3.3 Finding a single cluster

Suppose we have an everywhere ϵ-triangle dense graph H. We show how to remove a single cluster of a tightly-knit family. Since the entire focus of this subsection is on H, we drop the $\ldots(H)$ notation.

For set S of vertices and $\rho \in (0,1]$, we say S is ρ-extractable if: $H|_S$ is ρ-edge dense, ρ-triangle dense, $H|_S$ has radius 2, and $t_S^{(I)} \geq \rho t_S$. We define an extraction procedure that finds a single cluster in graph H.

The extraction procedure: Let i be a vertex of maximum degree. For every vertex j, let θ_j be the number of triangles incident on j whose other two vertices are in $N(i)$. Let R be the set of d_i vertices with the largest θ_j values. Output $S = \{i\} \cup N(i) \cup R$.

It is not necessary to start with a vertex of maximum degree, but doing so provides a better dependence on ϵ.

Note: The $\{i\}$ above is redundant, since $i \in R$. *Proof:* Let (k_1, k_2) be an edge in $N(i)$. Since θ_i is maximum, for any $\theta_j = \theta_i$, (j, k_1, k_2) must be a triangle, and in particular (j, k_1) must be an edge in H. Since $d_{k_1} \leq d_i$, there are at most d_i such maximum θ_js, including θ_i.

We start with a simple technical lemma.

Lemma 11. *Suppose $x_1 \geq x_2 \geq \cdots > 0$ with $\sum x_j \leq \alpha$ and $\sum x_j^2 \geq \beta$. For all $r \leq 2\alpha^2/\beta$, $\sum_{j \leq r} x_j^2 \geq \beta^2 r/4\alpha^2$.*

Proof. If $x_{r+1} \geq \beta/2\alpha$, then $\sum_{j \leq r} x_j^2 \geq \beta^2 r/4\alpha^2$ as desired. Otherwise,

$$\sum_{j>r} x_j^2 \leq x_{r+1} \sum_j x_j \leq \beta/2.$$

Hence, $\sum_{j \leq r} x_j^2 = \sum x_j^2 - \sum_{j>r} x_j^2 \geq \beta/2 \geq \beta^2 r/4\alpha^2$, using the bound given for r. □

The main theorem of the section follows.

Theorem 12. *Let H be an everywhere ϵ-triangle dense graph. The extraction procedure outputs an $\Omega(\epsilon^4)$-extractable set S of vertices. Furthermore, the number of edges in $H|_S$ is an $\Omega(\epsilon)$-fraction of the edges incident to S.*

Proof. Let i be a vertex of maximum degree, and let $N = N(i)$.

We have $|S| \leq 2d_i$. By Lemma 10, $H|_N$ has at least $\epsilon\binom{d_i}{2}$ edges, so $H|_S$ is $\Omega(\epsilon)$-edge dense. By the size of S and maximality of d_i, the number of edges in $H|_S$ is an $\Omega(\epsilon)$-fraction of edges incident to S. It is also easy to see that $H|_S$ has radius 2. It remains to show that $H|_S$ is $\Omega(\epsilon^4)$-triangle dense, and that $t_S^{(I)} = \Omega(\epsilon^4) t_S$.

For any j, let η_j be the number of edges from j to N, and let θ_j be the number of triangles incident on j whose other two vertices are in N. Let $x_j = \sqrt{2\theta_j}$.

We use $\widetilde{\Theta}$ and similar to hide factors of ϵ. The main insight of this section is that $H|_S$ has $\widetilde{\Theta}(d_i^3)$ triangles, versus the trivial lower bound of $\widetilde{O}(d_i^2)$. We will upper bound $\sum_j x_j$, lower bound $\sum_j x_j^2$, and then use Lemma 11 to show that the sum of the d_i largest θ_js is $\widetilde{\Theta}(d_i^3)$.

We first upper bound $\sum_j x_j$.

$$\sum_j x_j \leq \sum_j \sqrt{2\binom{\eta_j}{2}} \leq \sum_j \eta_j = \sum_{k \in N} d_k.$$

The first inequality follows from $\theta_j \leq \binom{\eta_j}{2}$. The last equality is simply stating that the total number of edges to N is the same as the total number of edges from N.

Now, for any $e = (k_1, k_2)$, $t_e \geq J_e \cdot \max(d_{k_1} - 1, d_{k_2} - 1) \geq \epsilon \cdot \max(d_{k_1} - 1, d_{k_2} - 1)$. Since $\epsilon > 0$, all degrees are at least 2, and $d_k - 1 \geq d_k/2$ for all k. So

$$t_e \geq \epsilon \frac{\max(d_{k_1}, d_{k_2})}{2} \geq \epsilon \frac{d_{k_1} + d_{k_2}}{4} \qquad \text{for all } e = (k_1, k_2).$$

We can now lower bound $\sum_j x_j^2$. Abusing notation, $e \in H|_N$ refers to an edge in the induced subgraph.

$$\sum_j x_j^2 = \sum_j 2\theta_j = \sum_{e \in H|_N} 2t_e \geq \sum_{(k_1,k_2) \in H|_N} \frac{\epsilon}{2}(d_{k_1} + d_{k_2}) = \frac{\epsilon}{2} \sum_{k \in N} d_k(H|_N)\, d_k,$$

where $d_k(H|_N)$ is the degree of vertex k within $H|_N$. The two sides of the second equality are counting (twice) the number of triangles "to" and "from" the edges of N.

We now use Lemma 11 with $\alpha = \sum_{k\in N} d_k$, $\beta = \frac{\epsilon}{2}\sum_{k\in N} d_k(H|_N)\, d_k$, and $r = d_i$. We first check that $r \le 2\alpha^2/\beta$. Note that $d_i \ge d_k \ge \epsilon d_i$ for all $k \in N$, by Lemma 10 and by the maximality of d_i. Hence,

$$\frac{2\alpha^2}{\beta} = \frac{4}{\epsilon}\frac{\left(\sum_{k\in N} d_k\right)^2}{\sum_{k\in N} d_k(H|_N)\, d_k} \ge \frac{4}{\epsilon}\frac{\epsilon d_i |N| \sum_{k\in N} d_k}{d_i \sum_{k\in N} d_k} \ge 4d_i \ge r,$$

as desired. Let R be the set of $r = d_i$ vertices with the highest θ_j, or equivalently, with the highest x_j^2. By Lemma 11, $\sum_{j\in R} x_j^2 \ge \beta^2 r/4\alpha^2$, or $\sum_{j\in R}\theta_j \ge \beta^2 r/8\alpha^2$. We compute

$$\frac{\beta}{\alpha} = \frac{\epsilon}{2}\frac{\sum_{k\in N} d_k(H|_N)\; d_k}{\sum_{k\in N} d_k} \ge \frac{\epsilon}{2}\min_{k\in N} d_k(H|_N) \ge \frac{\epsilon^2 d_i}{4},$$

which gives $\sum_{j\in R}\theta_j \ge \epsilon^4 d_i^3/128$. For the first inequality above, think of the $d_k/\sum d_k$ as weights on the $d_k(H|_N)$. For the last inequality, $d_k(H|_N) = t_{(i,k)} \ge J_{(i,k)}(d_i - 1) \ge \epsilon d_i/2$ for all $k \in N$.

Recall $S = N \cup R$ and $|S| \le 2d_i$. We have

$$t_S^{(I)} \ge \frac{\sum_{j\in R}\theta_j}{3} \ge \frac{\epsilon^4 d_i^3}{384},$$

since triangles contained in N get overcounted by a factor of 3. Since both t^S and the number of wedges in S are bounded above by $|S|\binom{d_i}{2} = \Theta(d_i^3)$, $H|_S$ is $\Omega(\epsilon^4)$-triangle dense, and $t_S^{(I)} = \Omega(\epsilon^4)t_S$ as desired. □

3.4 Getting the entire family in a triangle-dense graph

We start with a graph G and explain how to get the desired entire tightly-knit family. Our procedure (called the decomposition procedure) takes as input a parameter ϵ.

The decomposition procedure: Clean the graph with $clean_\epsilon$, and run the extraction procedure to get a set S_1. Remove S_1 from the graph, run $clean_\epsilon$ again, and extract another set S_2. Repeat until the graph is empty. Output the sets $S_1, S_2, \ldots$.

We now prove most of our main theorem, Theorem 3, restated for convenience. The time bound is covered in §6.

Theorem 13. *Consider a τ-triangle dense graph G. The decomposition procedure with any $\epsilon \le \tau/4$ outputs an $\Omega(\epsilon^4)$ tightly-knit family with an $\Omega(\epsilon^4)$-fraction of the triangles of G.*

Proof. We are guaranteed by Theorem 12 that $G|_{S_i}$ is $\Omega(\epsilon^4)$-edge and $\Omega(\epsilon^4)$-triangle dense and has radius 2. It suffices to prove that an $\Omega(\epsilon^4)$-fraction of the triangles in G are contained in this family.

Consider the triangles that are *not* present in the tightly-knit family. We call these destroyed triangles. Such triangles fall in two categories: those destroyed in the cleaning phases, and those destroyed when an extractable set is removed. Let C be the triangles destroyed during cleaning, and let D_k be the triangles destroyed in the kth extraction. By the definition of extractable subsets and Theorem 12, $t(G|_{S_k}) = \Omega(\epsilon^4|D_k|)$. Note that C, D_k, and the triangles in $G|_{S_k}$ (over all k) partition the total set of triangles. Hence, we get that $\sum_k t(G|_{S_k}) = \Omega(\epsilon^4(t - |C|))$.

We now bound $|C|$. This follows the proof of Claim 8. Let $e_1, e_2, \ldots$ be all the edges removed during cleaning phases. Let W_l and T_l be the set of wedges and triangles that are destroyed when e_l is

removed. Since the Jaccard similarity of e_l at the time of removal is at most ϵ, $|T_l| \leq \epsilon(|W_l| - |T_l|) \leq \epsilon|W_l|$. All the W_ls (and T_ls) are disjoint. Hence, $|C| = \sum_l |T_l| \leq \epsilon \sum_l |W_l| = \epsilon w = 3\epsilon t/\tau \leq 3t/4$, and $\sum_k t(G|_{S_k}) = \Omega(\epsilon^4 t)$ as desired. □

3.5 Preserving edges in an everywhere triangle-dense graph

For an everywhere triangle-dense graph, we can also preserve a constant fraction of the *edges*. This requires a more subtle argument. The aim of this subsection is to prove the following (cf. Theorem 6).

Theorem 14. *Consider an everywhere γ-triangle dense graph G. The decomposition procedure with any $\epsilon \leq \gamma^2/4$ outputs an $\Omega(\epsilon^4)$ tightly-knit family with an $\Omega(\epsilon^4)$ fraction of the triangles of G and an $\Omega(\epsilon\gamma)$ fraction of the edges of G.*

The tightly-knit family and triangle condition follow directly from Theorem 13, so we focus on the edge condition. By Theorem 12, the actual removal of the clusters preserves a large enough fraction of the edges. The difficulty is in bounding the edge removals during the cleaning phases. Even though G is everywhere triangle-dense, once a cluster is extracted, this condition may no longer hold. Indeed, the later cleaning phases may remove many edges, and we have to relate the behavior to the original graph.

We use E, W and T to denote the set of edges, wedges, and triangles in G. We use E^c, W^c and T^c to be the respective sets destroyed during the cleaning phases. Let $T^s = T \setminus T^c$. Let E^s be the edges involved in at least one triangle of T^s, and V^s be the vertices involved in at least one triangle of T^s.

The main lemma is the following, showing that the cleaning phase does not destroy too many edges.

Lemma 15. $|E^s| \geq \gamma|E|/6$.

Using this lemma, we complete the proof of Theorem 14.

Proof. (of Theorem 14) As mentioned above, the tightly-knit family and triangle condition follow directly from Theorem 13.

The proof for the edge condition follows a parallel argument to that of Theorem 13. Let D_k be the edges destroyed in the kth extraction, and let E_k be the edges in $G|_{S_k}$. By Theorem 12, $|E_k| = \Omega(\epsilon|D_k|)$. Since E^c, D_k, and E_k (over all k) partition E, we have $\sum_k |E_k| = \Omega(\epsilon(|E| - |E^c|))$. Since $|E^c| + |E^s| \leq |E|$, by Lemma 15 we have $\sum_k |E_k| = \Omega(\epsilon|E^s|) = \Omega(\epsilon\gamma|E|)$ as desired. □

To prove Lemma 15, we need additional definitions. For a triangle $t = (i_1, i_2, i_3)$, define the *weight* $r(t) = 1/d_{i_1} + 1/d_{i_2} + 1/d_{i_3}$. For a wedge w with central vertex i, define the weight $r(w) = 1/d_i$. Let $r(X) = \sum_{x \in X} r(x)$. Note that weights are always with respect to the degrees in the original graph G.

For any edge e, W_e and T_e are the set of wedges and triangles incident to e. If e is removed by the cleaning phases, then let W_e^c and T_e^c denote the set of wedges and triangles destroyed when e is removed. Note that W_e^c is not necessarily W_e, since some wedges in W_e may be removed prior to the removal of e.

We prove a series of bounds on various weights, crucially using the everywhere triangle-denseness of G. We finally chain them together to prove Lemma 15.

Claim 16. $r(T) \geq \gamma r(W)$.

Proof. By Lemma 10 (every neighborhood is dense) and the everywhere γ-triangle denseness, $t_i \geq \gamma w_i$ for every i. Thus,

$$r(T) = \sum_i \frac{t_i}{d_i} \geq \sum_i \frac{\gamma\, w_i}{d_i} = \gamma\, r(W).$$

□

Claim 17. *For any edge e, $r(T_e) \geq 2\gamma$. If e is removed in a cleaning phase, $r(T_e^c) \leq (3\epsilon/\gamma) r(W_e^c)$.*

Proof. Let $e = (i, j)$, where $d_i \geq d_j$. For any triangle δ incident to e, $r(\delta) \geq 2/d_i$. By the everywhere triangle density, $|T_e| \geq \gamma d_i$, so $r(T_e) = \sum_{\delta \in T_e} r(\delta) \geq 2\gamma$.

By Lemma 10 (edge balance), for any $\delta \in T_e^c$, $r(\delta) \leq 3/(\gamma d_i)$, and for any $w \in W_e^c$, $r(w) \geq 1/d_i$. Since $|T_e^c| < \epsilon(|W_e^c| - |T_e^c|) < \epsilon |W_e^c|$, $r(T_e^c) \leq (3\epsilon/\gamma) r(W_e^c)$. □

Claim 18. $|E^s| \geq r(T)/4$.

Proof. We prove two bounds that imply the claim: $r(T^s) \geq r(T)/4$ and $|E^s| \geq r(T^s)$. For the first, we upper bound $r(T^c) = r(T) - r(T^s)$. We can express $r(T^c) = \sum_{e \in E^c} r(T_e^c)$. By Claim 17, $r(T_e^c) \leq (3\epsilon/\gamma) r(W_e^c)$. The second to last inequality in the chain below is Claim 16.

$$r(T^c) \leq \frac{3\epsilon}{\gamma}\, r(W^c) \leq \frac{3\epsilon}{\gamma}\, r(W) \leq \frac{3\epsilon}{\gamma^2}\, r(T) \leq \frac{3r(T)}{4}.$$

This implies $r(T^s) \geq r(T)/4$. Now to prove $|E^s| \geq r(T^s)$. For vertex $i \in V^s$, we use T_i^s for the set of triangles in T^s containing i. We have

$$r(T^s) = \sum_{(i_1, i_2, i_3) \in T^s} \left(\frac{1}{d_{i_1}} + \frac{1}{d_{i_2}} + \frac{1}{d_{i_3}} \right) = \sum_{i \in V^s} \frac{|T_i^s|}{d_i}.$$

Consider the subgraph G' obtained by only having the triangles of T^s. The (non-trivial) vertices of this subgraph are exactly V^s, and the edges are E^s. Denote the degree of $i \in V^s$ in G' as d_i'. Trivially, $|T_i^s| \leq d_i'(d_i' - 1)/2$. We can bound $d_i \geq d_i'$. Combining these, we get

$$r(T^s) \leq \sum_{i \in V^s} \frac{d_i'(d_i' - 1)}{2d_i} \leq \sum_{i \in V^s} \frac{d_i' - 1}{2} \leq |E^s| - \frac{|V^s|}{2} \leq |E^s|$$

as desired. □

We now have all the pieces needed to prove Lemma 15.

Proof. (of Lemma 15) We observe that $r(T) = \sum_{e \in E} r(T_e)/3$, and use the bound $r(T_e) \geq 2\gamma$ (Claim 17). Starting with the bound of Claim 18,

$$|E^s| \geq \frac{r(T)}{4} \geq \sum_{e \in E} \frac{r(T_e)}{12} \geq \frac{2\gamma}{12} |E|.$$

□

4 Gallery of "inconvenient" graphs

As we have mentioned, our guiding intuition is the simple statement when the triangle density is 1, the graph is a disjoint union of cliques. Our aim is to get as close to this structure as possible, when the triangle density is constant. The definition of tightly-knit families is one attempt at this. But why must the radius be 2, and not 1? And why can't we find a tightly-knit family containing a constant fraction of the edges in a triangle-dense graph?

The following three examples answer some of these questions. We hope that these indicate why our theorems are non-trivial, and why they are stated in their current form.

• **Why radius 2?** Consider the complete tripartite graph. This is everywhere triangle-dense. If we removed the 1-hop neighborhood of any vertex, we would destroy a $1 - \Theta(1/n)$-fraction of the triangles. The only tightly-knit family in this graph is the entire graph itself.

• **More on 1-hop neighborhoods.** All 1-hop neighborhoods in an everywhere triangle-dense graph are edge-dense, by Lemma 10. Maybe we could just take the 1-hop neighborhoods of an independent set, to get a tightly-knit family? Of course, the clusters would only be edge-disjoint (not vertex disjoint).

We construct an everywhere triangle-dense graph where this does not work. There are $m+1$ disjoint sets of vertices, $A_1, \ldots, A_m, B$ each of size m. The graph induced on $\cup_k A_k$ is just a clique on m^2 vertices. Each vertex $b_k \in B$ is connected to all of A_k. Note that B is a maximal independent set, and the 1-hop neighborhoods of B contain $\Theta(m^4)$ triangles in total. However, the total number of triangles in the graph is $\Theta(m^6)$.

• **Why we can't preserve edges?** Theorem 3 only guarantees that the tightly-knit family contains a constant fraction of triangles, not edges. Consider a graph that has a clique on $n^{1/3}$ vertices and an arbitrary (or say, a random) constant-degree graph on the remaining $n - n^{1/3}$ vertices. No tightly-knit family can involve vertices outside the clique, so most of edges must be removed. Of course, most edges in this case have low Jaccard similarity.

In general, the condition of constant triangle density is fairly weak, since it encompasses a seemingly wide variety of graphs. The following two examples provide some further intuition for this class of graphs.

• **Triangle-dense graph far from disjoint union of cliques.** Define the graph Bracelet(m, d), for m nodes of degree d, when $m > 4d/3$, as follows: Let $B_1, .., B_{3m/d}$ be sets of $d/3$ vertices each put in cyclic order. Note that $3m/d \geq 4$. Connect each vertex in B_k to each vertex in B_{k-1}, B_k and B_{k+1}. This is an everywhere triangle-dense d-regular graph on m vertices. Nonetheless, it is maximally far (i.e. $O(md)$ edges away) from a disjoint union of cliques. A tightly-knit family is obtained by taking $B_1 \cup B_2 \cup B_3$, $B_4 \cup B_5 \cup B_6$, etc.

• **Hiding a tightly-knit family.** Start with $n/3$ disjoint triangles. Now, add an arbitrary bounded-degree graph (say, an expander) on these n vertices. The resulting graph has constant triangle density, but most of the structure is irrelevant for a tightly-knit family.

5 A planted cluster setting

We study the planted cluster model derived in Balcan et al [BBG09]. We show that our algorithm gives guarantees similar to theirs. We note that the novelty of [BBG09] is in the definition of their setting, and the conversion of clustering into a graph problem. We do not claim to subsume their paper. We merely observe that their graph problem essentially involves finding a tightly-knit family in a triangle-dense graph. Their setting ensures that there is (up to minor perturbations) a unique such family.

In [BBG09] (Lemma 5), they convert their planted clustering instance into a *threshold graph*. This graph $G = (V, E)$ contains k disjoint cliques $\{X_a\}$, such that the cliques do not have any common neighbors. (We use X_a for the vertex set.) We use the notation $N^*(U) = N(U) \cup U$. So $N^*(X_a) \cap N^*(X_b) = \emptyset$, when $a \neq b$. Unlike [BBG09], we assume $|X_a| \geq 3$.

Let $B = V \setminus \bigcup_a X_a$. It is shown in [BBG09] that when $|B|$ is small, very good approximations to $\{X_a\}$ can be found efficiently. From our perspective, when $|B|$ is much smaller than $\sum_a |X_a|$, the overall graph has high triangle density. Furthermore, the clusters output by the extraction procedure of Theorem 12 are very close to the X_a's.

Suppose we want a k-clustering of a threshold graph. We iteratively use the extraction procedure (of Theorem 12) k times to get clusters $S_1, S_2, \ldots, S_k$. Basically, we use the same procedure as before, but do not bother to clean the graph. Using the exact procedure of Theorem 13 also works fine, but in this setting the cleaning does not buy us anything.

We say a k-clustering is Δ-*incorrect* if there is a permutation σ such that $\sum |X_{\sigma(a)} \setminus S_a| \leq \Delta$. The following parallels the main theorem of [BBG09] (Theorem 5), and the proof is also quite similar to theirs.

Theorem 19. *The output of the clustering algorithm above is $O(|B|)$-incorrect on G.*

Proof. We first map the algorithm's clustering to the true clustering $\{X_a\}$. Our algorithm outputs k clusters, each with an associated "center" (the starting vertex). These are denoted $S_1, S_2, \ldots$ with centers $s_1, s_2, \ldots$ in order of extraction. We determine if there exists some true cluster X_a, such that $s_1 \in N^*(X_a)$. If so, we map S_1 to X_a. (Since the $N^*(X_a)$'s are disjoint, X_a is unique if it exists.) If no X_a exists, we simply do not map S_1. We then perform this for $S_2, S_3, \ldots$, except that we also do not map S_k if we would be mapping it to an X_a that has previously been mapped to. We finally end up with a subset $P \subset [k]$, such that for each $a \in P$, S_a is mapped to some $X_{a'}$. By relabeling the true clustering, we can assume that for all $a \in P$, S_a is mapped to X_a. The remaining clusters (for $X_{a \notin P}$) can be labeled with an arbitrary permutation of $[k] \setminus P$.

Our aim is to bound $\sum_a |X_a \setminus S_a| = O(|B|)$.

We perform some simple manipulations.

$$\begin{aligned}
\bigcup_a (X_a \setminus S_a) &= \bigcup_{a \in P} (X_a \setminus S_a) \cup \bigcup_{a \notin P} (X_a \setminus S_a) \\
&\subseteq \bigcup_{a \in P} (X_a \cap \bigcup_{b<a} S_a) \cup \bigcup_{a \in P} (X_a \setminus \cup_{b \leq a} S_a) \cup \bigcup_{a \notin P} X_a \\
&\subseteq \bigcup_{a} (X_a \cap \bigcup_{b<a} S_a) \cup \bigcup_{a \in P} (X_a \setminus \cup_{b \leq a} S_a) \cup \bigcup_{a \notin P} X_a.
\end{aligned}$$

So we get the following sets of interest.

- $L_1 = \bigcup_a (X_a \cap \bigcup_{b<a} S_a) = \bigcup_b (S_b \cap \bigcup_{a>b} X_a)$ is the set of vertices that are "stolen" by clusters "before" S_a.
- $L_2 = \bigcup_{a \in P} (X_a \setminus (\cup_{b \le a} S_b))$ is the set of vertices that are "left behind" when S_a is created.
- $L_3 = \bigcup_{a \notin P} X_a$ is the set of vertices that are "never clustered".

The proof is completed by showing $|L_1| + |L_2| + |L_3| = O(|B|)$. This will be done through a series of claims.

Recall that each step we choose the vertex s_b with highest degree (at this point in the algorithm) d_b. We set N_b to be the d_b neighbors of s_b at this point, $N_b^* = N_b \cup \{s_b\}$, and N_b' to be the d_b vertices with the largest number of triangles to N_b^*. Then, $S_b = N_b^* \cup N_b'$. Note that $|S_b| \le 2d_b + 1 \le 3d_b$. Note that all these are with respect to the subgraph of G present when S_b is extracted. We first state a useful fact.

Claim 20. *Suppose for some b, $s_b \in N(X_b)$. Then N_b partitions into $N_b \cap X_b$ and $N_b \cap B$.*

Proof. Any vertex in $N_b \setminus X_b$ must be in B. This is because N_b is contained in a two-hop neighborhood from X_b, which cannot intersect any other X_a. □

Claim 21. $|S_b \cap \bigcup_{a>b} X_a| \le 6|S_b \cap B|$.

Proof. We split into three cases. For convenience, let U be the set of vertices $S_b \cap \bigcup_{a>b} X_a$.

- For some c, $s_b \in X_c$: Note that $c = b$ by the relabeling of clusters. Observe that S_b is contained in a two-hop neighborhood of s_b, and hence cannot interest any cluster X_a for $a \neq b$. Hence, $S_b \cap \bigcup_{a>b} X_a$ is empty.
- For some (unique) c, $s_b \in N(X_c)$: Again, $c = b$. By Claim 20, $d_b = |N_b| = |N_b \cap X_b| + |N_b \cap B|$. So one of these must be at least $d_b/2$. Suppose $|N_b \cap B| \ge d_b/2$. Then $|S_b \cap B| \ge |N_b \cap B| \ge d_b/2$. We can easily bound $|S_b| \le 3d_b \le 6|S_b \cap B|$.

Suppose $|N_b \cap X_b| > d_b/2$. Note that $|N_b \cap X_b|$ is a clique. Each vertex in $N_b \cap X_b$ makes $\binom{|N_b \cap X_b|}{2}$ triangles in N_b^*. On the other hand, any , the only vertices of N_b^* that any vertex in X_a for $a \neq b$ can connect to is in $N_b \cap B$. This forms less than $\binom{d_b/2}{2}$ triangles in N_b^*.

Consider the construction of S_b. We take the top d_b vertices with most triangles to N_b^*, and say we insert them in decreasing order of this number. Before any vertex of X_a ($a \neq b$) is added, all vertices of $N_b \cap X_b$ must be added. Hence, at most $d_b - |N_b \cap X_b| = |N_b \cap B| \le |S_b \cap B|$ vertices of $\cup_{a \neq b} X_a$ can be added to S_b. Therefore, $|U| \le |S_b \cap B|$.

- The vertex s_b is at least distance 2 from every X_c: Note that $N_b^* \subseteq S_b \cap B$. Hence, $|S_b| \le 3d_b \le 3|S_b \cap B|$. □

Claim 22. *Fix some $a \in P$. Then $|X_a \setminus (\cup_{b \le a} S_b)| \le |S_a \cap B|$.*

Proof. Since $a \in P$, either $s_a \in X_a$ or $s_a \in N(X_a)$. Suppose $s_a \in X_a$. Since X_a forms a clique, $N_a^* \supseteq X_a$. Hence $(X_a \setminus (\cup_{b \le a} S_b))$ is empty.

Suppose $s_a \in N(X_a)$. Consider the situation of the algorithm after the first $a-1$ sets $S_1, S_2, \ldots, S_{a-1}$ are removed. There is some subset of X_a that remains; call it $X_a' = X_a \setminus (\cup_{b<a} S_b)$. At this stage, the

degree of s_a is d_a. Because it has maximal degree and X'_a is a clique, $d_a \geq |X'_a|$. Note that $|X'_a \setminus S_a|$ is what we wish to bound, and $|X'_a \setminus S_a| \leq |X'_a \setminus N_a|$. By Claim 20, N_a partitions into $N_a \cap X_a = N_a \cap X'_a$ and $N_a \cap B$. We have $|X'_a \setminus N_a| = |X'_a| - |N_a \cap X_a| \leq d_a - |N_a \cap X_a| = |N_a \cap B| \leq |S_a \cap B|$. □

Claim 23. $|L_3| \leq |B| + |L_1|$.

Proof. Consider some X_a for $a \notin P$. Look at the situation when $S_1, \ldots, S_{a-1}$ are removed. There is a subset X'_a (forming a clique) left in the graph. All the vertices in $X_a \setminus X'_a$ are contained in L_1. By maximality of degree, $d_a \geq |X'_a|$. Furthermore, since $a \notin P$, $N_a \subseteq B$ implying $d_a \leq |S_a \cap B|$. Therefore, $|X'_a| \leq |S_a \cap B|$. We can bound $\bigcup_{a \notin P} (X_a \setminus X'_a) \subseteq L_1$, and $\sum_{a \in P} |X'_a| \leq |B|$, completing the proof. □

It remains to put it all together. By summing the bound of Claim 21 and Claim 22 over all a, we get $|L_1| \leq 6|B|$ and $|L_2| \leq |B|$. Claim 23 with the bound on $|L_1|$ yields $|L_3| \leq 7|B|$, completing the proof of Theorem 19. □

6 Implementation of algorithm

We provide an implementation of the decomposition procedure (cf. Theorem 13) that runs in $O(|V| + |E| + w)$ time and $O(|E|)$ space, completing the proofs of Theorem 3 and Theorem 6.

We make a few improvements over a completely naive implementation of the procedure to get the running time down to $O(|V| + |E| + w)$. There are three operations that could potentially cause a slowdown: cleaning, finding the maximum degree vertex i, and finding vertices with many triangles incident on the neighborhoods M.

To make cleaning fast we note that $J_{(i,j)}$ only changes when edges incident on i or j get removed. Hence, we maintain a set of "dirtied" vertices after extraction and after each iteration of the cleaning procedure, and only recompute Jaccard similarities for the edges incident on those dirtied vertices. We also keep track of the number of triangles t_e incident on each edge and the degree d_i of each vertex to make computing J_e an $O(1)$ operation. Computing J_e for every edge on every iteration of the cleaning procedure would have taken $O(|E|^2)$ time, which is too much.

For finding maximum degree vertices, we maintain a lookup from degree d to vertices of degree d, and also keep track of the maximum degree $d_{\max}$. Iterating over V to compute $d_{\max}$ after each extraction would have added an $O(|V|^2)$ term.

For finding vertices with many triangles incident on the neighborhoods M, it is okay to do $O(1)$ work for every triangle with an edge in M. We just need to make sure to enumerate them by computing T_e for each $e \in M$, rather than trying to directly compute the number of triangles from each vertex i, which instead costs $O(|V|w)$.

Sections §6.1 and §6.2 provide a detailed implementation and analysis of the procedure.

6.1 Data Structures

We will implicitly rely on the uniform hashing assumption, without comment. In particular, we assume the following primitives:

- A *hash table*, or key-value store. With m keys, we assume $O(m)$ space, amortized or expected $O(1)$ time look-up, insertion, and deletion of any key, $O(m)$ enumeration of keys, expected $O(1)$ peek of a "random" key, and $O(1)$ look-up of number of keys. We denote hash tables by $\text{HASH}(\mathcal{K}, k \to f(k))$, where $\mathcal{K}$ is the set of keys, and $k \to f(k)$ is the mapping from keys to values.
- A *set*, or key store. This is simply a degenerate hash table, where every value is 0. Note that the above properties imply $O(m_1 + m_2)$ time intersection of two sets with m_1 and m_2 keys respectively.
- A *multiset*, or key-count store. This is a hash table where the keys are the distinct elements, and the values are counts. We do not store keys with value zero.

Let V' be the set of non-isolated vertices, namely vertices i with $d_i > 0$. We maintain the following data structures for the course of the algorithm.

- A hash table $\mathcal{G} = \text{HASH}(V', i \to \text{SET}(N(i)))$ of vertex neighborhoods. This allows $O(N(i))$ lookup of $N(i)$.
- A hash table $\mathcal{D} = \text{HASH}(V', i \to d_i)$ of vertex degrees.
- An array $\mathcal{A}$, where the d'th entry is a pointer to all the vertices of degree d, stored as a set. Note that we implicitly keep track of $|\mathcal{A}[d]|$ via our definition of set.
- An integer $d_{\max} = \max\{d : |\mathcal{A}[d]| > 0\}$.
- A hash table $\mathcal{T} = \text{HASH}(E, e \to t_e)$ that keeps track of the number of triangles incident on e.

Note that we can iterate over all edges in $O(|E|)$ time, via $\mathcal{G}$. We can also iterate over all triangles in $O(w + |E|)$ time, by computing $N(i) \cap N(j)$ for each $(i, j) \in E$.

We define the following operations on the data structures above:

- DeleteEdge(i, j) removes edge (i, j).
- Jaccard(i, j) returns $J_{(i,j)} = \frac{\mathcal{T}(i,j)}{\mathcal{D}(i)+\mathcal{D}(j)-\mathcal{T}(i,j)}$, and takes $O(1)$ time.
- IsEmptyGraph() returns True if $d_{\max}$ is 0, and takes $O(1)$ time.
- MaxDegreeVertex() returns a vertex from $\mathcal{A}(d_{\max})$, and takes expected $O(1)$ time.

6.2 Procedure

We first define the two subroutines below.

Clean starts with a dirty set of vertices $\mathcal{V}$. It iterates over edges incident on $\mathcal{V}$ until it finds one with $J_{(i,j)} < \epsilon$, after which it deletes (i, j), adds i and j to $\mathcal{V}$, and starts over. If it iterates over all the edges of some $i \in \mathcal{V}$ without finding one with low Jaccard similarity, it removes i from $\mathcal{V}$.

```
1 function Clean(𝒱, ϵ)
2     while 𝒱 is non-empty do                    ▷ 𝒱 is the set of dirty vertices
3         for i ∈ 𝒱 do
4             for j ∈ N(i) do
5                 if Jaccard(i, j) < ϵ then
6                     DeleteEdge(e)
```

```
7                Add j to V and go to line 2.
8            Remove i from V.
```

Recall that θ_j is the number of triangles incident on j whose other two vertices are in $N(i)$. EXTRACT computes θ_j for each j where $\theta_j > 0$, by iterating over T_e for every edge $e \in N(i)$. It then takes the largest d_i such θ_js to form the extracted set S. It removes S from the graph, and dirties the vertices in the neighborhood of S for subsequent cleaning.

```
1  function Extract
2      i = MaxDegreeVertex()
3      θ = a multiset over vertices                ▷ θ(j) will count the number of triangles
4                                                    in N(i) ∪ j incident on j.
5      for j1 ∈ N(i) do
6          for j2 ∈ N(j1) ∩ N(i) do                 ▷ (j1, j2) iterates over the edges of N(i)
7              for j ∈ N(j1) ∩ N(j2) do
8                  Add j to θ.
9      R = the D(i) keys of θ with the highest counts
10     S = R ∪ N(i)
11     V = (∪_{s∈S} N(s)) \ S                       ▷ dirtied vertices
12     for e incident on S do
13         DeleteEdge(e)
14     return S, V
```

Finally, we glue the two subroutines together to get the main function below. PARTITION alternatively CLEANs and EXTRACTs until the graph is gone.

```
1  function Partition(G, ε)
2      Construct the data structures G, D, A, d_max, T from G.
3      P = empty list                               ▷ stores the partition
4      Clean(V', ε)
5      while not IsEmptyGraph() do
6          S, V = Extract()
7          Append S to P.
8          Clean(V, ε)
9      return (P)
```

Note that that PARTITION technically throws away isolated vertices from G. It is not too hard to modify the algorithm to include each isolated vertex as its own element of $\mathcal{P}$.

Theorem 24. *The algorithm above runs in expected $O(|V| + |E| + w)$ time and $O(|E|)$ space.*

Proof. The space bound is easy to check, so we focus on the time. We look at the total amount of time spent in DELETEEDGE, CLEAN, EXTRACT, and PARTITION in order; for each function, we ignore time spent in subroutines that have been previously accounted for. The *total cost* of a function or line of code refers to the time spent there over the course of the entire algorithm.

DELETEEDGE: Each edge gets deleted exactly once. It is easy to check that the total cost to $\mathcal{G}, \mathcal{D}$, and $\mathcal{A}$ is $O(|E|)$ each. The total cost to $d_{\max}$ is $O(|V|)$; every time $|\mathcal{A}(d_{\max})|$ drops to 0, we do a linear search downwards till we find a non-empty entry of $\mathcal{A}$. The total cost to $\mathcal{T}$ is at most

$\sum_{(i,j)\in E} d_i + d_j = O(|E| + w)$; on deletion of (i, j), $\mathcal{T}(e)$ is decremented for $e = \{(i, k), (j, k) : k \in N(i) \cap N(j)\}$.

CLEAN: Say a vertex k gets "chosen" every time $i = k$ in lines 4–8. Each vertex $i \in V'$ gets chosen once from line 4 of PARTITION, and at most once each time d_i changes to a value > 0. Since d_i only decreases, each i is chosen at most d_i times. The cost of choosing a vertex i is at most d_i, so the total cost here is at most $\sum_{i\in V} d_i^2 = O(|E| + w)$.

EXTRACT: Over the course of the algorithm each edge ends up as a (j_1, i) pair at most once, so the total cost of lines 5–6 is at most $\sum_{(j_1,i)\in E} d_{j_1} + d_i = O(|E| + w)$. Similarly, each edge ends up as a (j_1, j_2) pair at most once, and so the total cost of lines 7–8 is also $O(|E| + w)$.

Line 9 can be computed in time proportional to the number of keys in θ. Each vertex i can be in θ at most d_i times, since once $i \in \theta$ it loses at least one edge to lines 12–13. Hence the total cost of line 9 is $O(|E|)$.

Finally, each vertex can be in S at most once, and so the total cost of line 11 is $O(|E|)$.

PARTITION: The only non-trivial step here is line 2. Initial construction of $\mathcal{G}, \mathcal{D}, \mathcal{A}$ and $d_{\max}$ takes $O(|V| + |E|)$ time, if the graph is originally presented as either as a list of edges or as an adjacency list. Initial construction of $\mathcal{T}$ takes $O(|E| + w)$ time, via iterating over all triangles. □

References

[AJB00] R. Albert, H. Jeong, and A.-L. Barabsi. Error and attack tolerance of complex networks. *Nature*, 406:378–382, 2000. 1

[BA99] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, October 1999. 1

[BBG09] M.-F. Balcan, A. Blum, and A. Gupta. Approximate clustering without the approximation. In *Proceedings of the Symposium on Discrete Algorithms (SODA)*, pages 1068–1077, 2009. 4, 13

[BKM+00] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer Networks*, 33:309–320, 2000. 1

[Bur04] Ronald S Burt. Structural holes and good ideas. *American Journal of Sociology*, 110(2):349–399, 2004. 4

[CF06] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys*, 38(1), 2006. 1, 4

[CL02a] F. Chung and L. Lu. Connected components in random graphs with given degree sequences. *Annals of Combinatorics*, 6:125–145, 2002. 1

[CL02b] Fan Chung and Linyuan Lu. The average distances in random graphs with given expected degrees. *PNAS*, 99(25):15879–15882, 2002. 1, 4

[Col88] James S. Coleman. Social capital in the creation of human capital. *American Journal of Sociology*, 94:S95–S120, 1988. 4

[CZF04] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A recursive model for graph mining. In *SDM '04*, pages 442–446, 2004. 1

[Fau06] Katherine Faust. Comparing social networks: Size, density, and local structure. *Metodoloski zvezki*, 3(2):185–216, 2006. 4

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *Proceedings of SIGCOMM*, pages 251–262, 1999. 1

[For10] S. Fortunato. Community detection in graphs. *Physics Reports*, 486:75–174, 2010. 1

[FPP06] Alessandro Ferrante, Gopal Pandurangan, and Kihong Park. On the hardness of optimization in power law graphs. In *Proceedings of Conference on Computing and Combinatorics*, 2006. 4

[FWVDC10] Brooke Foucault Welles, Anne Van Devender, and Noshir Contractor. Is a friend a friend?: Investigating the structure of friendship networks in virtual worlds. In *CHI-EA '10*, pages 4027–4032, 2010. 4

[GN02] M. Girvan and M. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002. 1

[HL70] P. W. Holland and S. Leinhardt. A method for detecting structure in sociometric data. *American Journal of Sociology*, 76:492–513, 1970. 4

[Kle00a] Jon M. Kleinberg. Navigation in a small world. *Nature*, 406:845, 2000. 1

[Kle00b] Jon M. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of ACM Symposium on Theory of Computing*, 2000. 1

[Kle01] Jon M. Kleinberg. Small-world phenomena and the dynamics of information. In *Advances in Neural Information Processing Systems (NIPS)*, volume 14, 2001. 1

[KRR+00] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 57–65, 2000. 1

[LAS+08] H. Lin, C. Amanatidis, M. Sideri, R. M. Karp, and C. H. Papadimitriou. Linked decompositions of networks and the power of choice in polya urns. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, pages 993–1002, 2008. 1

[LCK+10] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *J. Machine Learning Research*, 11:985–1042, February 2010. 1

[LLDM08] J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2008. 1

[LT79] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979. 5

[MS10] A. Montanari and A. Saberi. The spread of innovations in social networks. *Proceedings of the National Academy of Sciences*, 107(47):20196–20201, 2010. 1

[New01] M. E. J. Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences*, 98(2):404–409, 2001. 1

[New03] M. E. J. Newman. Properties of highly clustered networks. *Phys. Rev. E*, 68:026121, August 2003. 1

[New06] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(3):036104, 2006. 1

[RS86] N. Robertson and P. D. Seymour. Graph minors iii: Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1986. 5

[SCW+10] Alessandra Sala, Lili Cao, Christo Wilson, Robert Zablit, Haitao Zheng, and Ben Y. Zhao. Measurement-calibrated graph models for social network experiments. In *WWW '10*, pages 861–870. ACM, 2010. 1, 4

[SPK13] C. Seshadhri, Ali Pinar, and Tamara G. Kolda. Fast triangle counting through wedge sampling. In *Proceedings of the SIAM Conference on Data Mining*, 2013. 4

[SPR11] V. Satuluri, S. Parthasarathy, and Y. Ruan. Local graph sparsification for scalable clustering. In *Proceedings of ACM SIGMOD*, pages 721–732, 2011. 5, 6

[Sze78] E. Szemerédi. Regular partitions of graphs. *Problèmes combinatoires et thorie des graphes*, 260:399–401, 1978. 5

[UBK13] J. Ugander, L. Backstrom, and J. Kleinberg. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *Proceedings of the World Wide Web Conference (WWW)*, 2013. 5

[UKBM11] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. The anatomy of the facebook social graph. Technical Report 1111.4503, Arxiv, 2011. 1, 2, 4

[VB12] J. Vivar and D. Banks. Models for networks: A cross-disciplinary science. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4(1):13–27, 2012. 4

[WF94] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994. 2

[WS98] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998. 1, 4, 6

[WW10] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proceedings of Foundations of Computer Science (FOCS)*, pages 645–654, 2010. 4