Faster Clustering Coefficient Using Vertex Covers

Oded Green, David A. Bader
College of Computing
Georgia Institute of Technology
Atlanta, Georgia, USA

***Abstract*—Clustering coefficients, also called triangle counting, is a widely-used graph analytic for measuring the closeness in which vertices cluster together. Intuitively, clustering coefficients can be thought of as the ratio of common friends versus all possible connections a person might have in a social network. The best known time complexity for computing clustering coefficients uses adjacency list intersection and is $O(V \cdot d_{max}^2)$, where d_{max} is the size of the largest adjacency list of all the vertices in the graph. In this work, we show a novel approach for computing the clustering coefficients in an undirected and unweighted graphs by exploiting the use of a vertex cover, $\hat{V} \subseteq V$. This new approach reduces the number of times that a triangle is counted by as many as 3 times per triangle. The complexity of the new algorithm is $O(\hat{V} \cdot \hat{d}_{max}^2 + t_{VC})$ where $\hat{d}_{max}$ is the size of the largest adjacency list in the vertex cover and t_{VC} is the time needed for finding the vertex cover. Even for a simple vertex cover algorithm this can reduce the execution time $10\% - 30\%$ while counting the exact number of triangles (3-circuits). We extend the use of the vertex cover to support counting squares (4-circuits) and clustering coefficients for dynamic graphs.**

***Keywords*-Graph Algorithms; Social Network Analysis; Network Science;**

I. Introduction

Clustering coefficients is a graph analytic that states how tightly bound vertices are in a graph [29]. The tightness is measured by computing the number of closed triangles in the graph, which can then imply the small-world property. Computing the clustering coefficients has been applied to many types of networks: communication [26], collaboration [27], social [27], citation [23], and biological [7]. In social networks, one can think of the local clustering coefficients as the ratio of actual mutual acquaintances versus all possible mutual acquaintances. Applications using clustering coefficients include DIMES [26], which is a distributed platform used for providing an accurate, and comprehensive map of the Internet and automatic Web-Spam detection [8].

There are two types of clustering coefficients: global and local. The global clustering coefficient is a single value computed for the entire graph, whereas the local clustering coefficient is computed per vertex. Both are computed in a similar fashion. Without the loss of generality, we consider the global clustering coefficient in this work; nonetheless our approach is applicable for computing local clustering coefficients. Table I presents the notations used in this paper.

Definition 1: Formally, the global clustering coefficient is:

$$CC_{global} = \sum_{v \in V} CC_v = \sum_{v \in V} \frac{tri(v)}{deg(v) \cdot (deg(v) - 1)}$$

There are several approaches for computing clustering coefficients:

1) Enumerating over all node-triples. This has an $O(V^3)$ upper bound complexity.
2) Using matrix multiplication. This has an $O(V^w)$ complexity where $w \leq 2.376$ [11].
3) Intersecting adjacency lists. This has an $O(V \cdot d_{max}^2)$ upper bound [25] where d_{max} is the vertex with largest adjacency.

In this paper we show a novel and intuitive way to improve the adjacency list intersection approach that removes redundant list intersections, therefore reducing the run-time of the algorithm. We accomplish this by only intersecting the adjacency lists of vertices that are marked in an arbitrary vertex cover.

We then extend our method for computing a circuit of any length and illustrate this with circuits of length 4. We show that our new approach can also be applied to dynamic graphs.

The remainder of the paper will be structured as follows. Section II discusses the related work and discusses real world graph properties and introduces vertex covers. In Section III, we present our new algorithm for counting 3-circuits (clustering coefficients), 4-circuits, and extend the vertex cover approach for computing clustering coefficients for dynamic graphs. In Section IV, we discuss our experimental methodology and present quantitative results. Finally, in Section V, we present our conclusions and discuss future work.

II. Related Work

In this section we review the literature that addresses the challenge of computing clustering coefficients for large dynamic graphs. These approaches take into account optimizations such as parallelization, approximation, and dynamic algorithms that make only local modifications to the graphs. These optimizations are crucial in order to analyze social networks such as Facebook [1] and Twitter [2] which can potentially have million of members and thousands of

978-0-7695-5137-1/13 $26.00 © 2013 IEEE
DOI 10.1109/SocialCom.2013.51

Table I
NOTATIONS IN THIS PAPER

Name	Description
CC_{global}	Global clustering coefficient
CC_v	Clustering coefficient for vertex v
$deg(v)$	Degree of vertex v.
$tri(v)$	Number of triangle that vertex v is in.
$\hat{V}$	A vertex cover of the graph, $\hat{V} \subseteq V$.
d_{max}	Vertex with maximal degree in the graph.
$\hat{d}_{max}$	Vertex with maximal degree from the vetex cover.
u, v, w, x	Vertices in the graph.

updates per second. The combination of these approaches allows scaling of the data set.

Clustering coefficients is an analytic that is relatively simple to parallelize because of the relatively large number of independent operations that can be executed. The exact parallel granularity that is used and whether it is local or global will define the synchronization methods required, which is important as the number of threads in a system increases and when full system utilization is desired. An additional benefit of computing clustering coefficients for dynamic graphs is that a given update requires modifying only local values.

We now discuss algorithms that have tackled the challenges mentioned above. We add that our new algorithm is an orthogonal optimization to the ones mentioned, meaning that our approach would also benefit from parallelization and the creation of a dynamic algorithm.

In [13], a massively multithreaded architecture, the Cray XMT2, is used for computing dynamic clustering coefficients for a graph with half a billion edges. The Cray XMT2 is a massively parallel system that supports several thousand light-weight concurrent threads with a Uniform Memory Architecture (UMA) that allows for fine-grain parallelism. To achieve high system utilization on the XMT2 for the dynamic case, the incoming updates are batched together and dealt with concurrently by the various threads. In [22] a GPU implementation of clustering coefficients is presented.

Approximation and dynamic data flows are the focus of [8], [10]. These show different techniques for approximating clustering coefficients. In [13] an additional approximate algorithm is given that is based on Bloom filters. The Bloom filters are created only for a subset of the vertices in the batch.

A. Real World Graph Properties

Many social networks share common properties. One of these is the small-world property. The first work to discuss the small-world property is due to Milgram [24]. Milgram suggests that people within the United States are not likely to be separated by more than six steps. This was later reconfirmed in [29] with additional experimentation. Watts and Strogatz first introduce the concept of clustering coefficients in [29].

Further research on graph properties advanced when it was shown in [6], [9], [16] that the world wide web follows a power law distribution on the number of adjacent edges a vertex has. These graphs are considerably sparse with an average degree that is constant. For example, the average degree for Facebook [1] is 189 [28], which is very small compared to the number of vertices in the graph. This power law distribution is important for an analytic such as clustering coefficients where the time complexity is based on the vertex with the highest degree.

B. Vertex Covers

A Vertex Cover is a set $\hat{V} \subseteq V$ such that for all $(a, b) \in E$, either $a \in \hat{V}$ or $b \in \hat{V}$ (or possibly both). A minimal vertex cover is the smallest $\hat{V} \subseteq V$ meeting the requirement above. Finding a vertex cover is a well studied problem [12], [19], [21], [5], [18]. Finding the minimal vertex cover is known to be NP-Complete (NPC) [17], [20]. We do not extend the discussion on this as it not relevant for this paper.

While finding the minimal vertex cover is intractable, there are numerous algorithms that can find some vertex cover in polynomial time, including linear time [12], [19], [21], [5]. In [21] a parallel algorithm for computing the vertex cover based on the A* formulation is given.

A theoretic PRAM algorithm is given in [19] where the vertex cover is found in iterative fashion using an Euler circuit. A total of $O(V + E)$ processors are required and the time complexity is $O(log^3(E))$.

III. Vertex Cover Clustering Coefficients

In this section we show a new algorithm for finding the clustering coefficients using vertex covers that avoid wasteful neighbor intersections.

A. Clustering Coefficients for Static Graphs

In the following subsections we discuss both theoretical and practical implications of the the vertex cover. For simplicity, assume that a vertex cover of $\hat{V}$ of $G = (V, E)$ is given.

Consider a triangle made up of three vertices u, v, w. These vertices may potentially have additional adjacencies; for our discussion these edges are not relevant. Now consider a vertex cover over this triangle. Fig. 1 depicts all legal vertex covers of a triangle. Note that a closed triangle, made up of $u, v, w \in V$, can be represented as six different tuples: $(u, v, w), (u, w, v), (v, u, w), (v, w, u), (w, u, v), (w, v, u)$. All these tuples need to be counted. Note, that by using a lexicographical sorting, only three of these tuples need to be counted. This however, does not change the relevance of our approach as our algorithm benefits from the lexicographical sorting as well.

Lemma 1: At least two out of the three vertices that make up a triangle are within the vertex cover, $\hat{V}$. By contradiction, consider the case that only one vertex is in

Algorithm 1: Pseudo-code for computing clustering coefficients given a vertex cover $\hat{V}$.

```
$triangles \leftarrow 0$;
for $v \in \hat{V}$ do
    for $u \in adj(v)$ do
        if $u = v$ then
            next $u$;
        if $u \in \hat{V}$ then
            $C \leftarrow intersect(v, adj(c), u, adj(v))$;
            for $c \in C$ do
                if $c \in \hat{V}$ then
                    $triangles \Leftarrow triangles + 1$;
                else
                    $triangles \Leftarrow triangles + 3$;
```

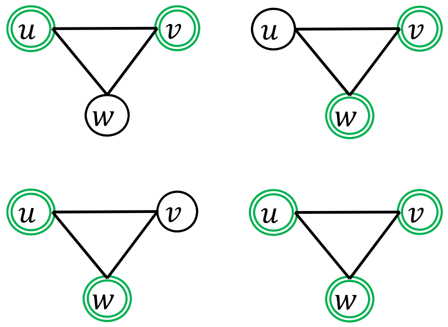

Figure 1. All the legal vertex covers of the triangle made up of u, v, w. The vertices in the vertex cover are marked with an additional circle.

the vertex cover, $\hat{V}$, without the loss of generality, assume $v \in \hat{V}$ and $u, w \notin \hat{V}$. As such there is an edge $(u, w) \in E$ that is not covered, meaning that $\hat{V}$ is not a vertex cover. ■

Lemma 2: Given two vertices $u, v \notin \hat{V}$, there are no triangles in the graph in which both u and v are in the same triangle. This is immediate from Lemma 1. ■

As a results of these two Lemmas, we only need to intersect the adjacency lists of two adjacent vertices when both vertices are marked in the vertex cover. That is, for $u, v \in V \wedge (u, v) \in E$ such that either:

1) $v \in \hat{V} \wedge u \notin \hat{V}$
2) $v \notin \hat{V} \wedge u \in \hat{V}$

the intersection between these vertices is not required.

In the original algorithm, this intersection is indeed computed. We now show that it is only necessary to intersect adjacency lists of adjacent vertices that are both in the vertex cover.

We denote C as the set of the intersections between the two vertices. If C is empty, these vertices do not have common neighbors. If C is not empty then for each $w \in C$ the following two scenarios can happen: 1) $w \in \hat{V}$ or 2) $w \notin \hat{V}$. Both these scenarios are depicted in Fig. 1.

Using the above observations we show that it is possible to compute the clustering coefficient by intersecting the adjacency lists of two vertices u, v such that $u, v \in \hat{V} \wedge (u, v) \in E$. We are required to show that all six tuples are accounted for. We start with the case that $w \in C \wedge w \in \hat{V}$, as such $u, v, w \in \hat{V}$. When intersecting u with v, w is found. When intersecting v with u, w is found again. The intersection will be repeated for intersecting u with v, v with w and their respective reverse orders. For each of the triangles found, the global triangle counter is increased. All six tuples have been accounted for.

For the second scenario when $w \in C \wedge w \notin \hat{V}$, the only intersection computed is between vertices u and v and vertices u and v. w is not intersected with other vertices as it is not in the vertex cover. As such, some intersections are omitted. Yet, the omitted intersections can be accounted for by a simple counting correction where each intersection is counted as 3 different triangles. Given the two computed intersections, u with v and v with u, all the six triangles are properly accounted for.

Given the above, it is possible to create an algorithm for computing the clustering coefficients given a vertex cover that reduces the number of neighbor intersections. The pseudo-code for this is given in Algorithm 1. Note the algorithm assumes that a vertex cover has been computed. We extend the discussion on the time complexity of vertex covers in a following subsection.

B. Complexity Analysis

In this section we do a complexity analysis of our new algorithm and discuss the cost of computing the vertex cover in terms of time complexity. The time complexity of the original algorithm is $O(V \cdot d_{max}^2)$ where d_{max} is the size of the largest adjacency list in the graph. The key difference between our new algorithm and the original algorithm is that only a subset of vertices requires adjacency list intersection: $\hat{V}$. Further, only the adjacency lists of vertices that are within the vertex cover are intersected. As such the $O(d_{max}^2)$ is replaced with $O(\hat{d}_{max}^2)$, where $\hat{d}_{max}$ is the size of the largest adjacency list in the vertex cover.

Intuitively, the computational bottleneck of the original algorithm is the vertex with the largest adjacency list. This is because this vertex is intersected with all its neighbors d_{max} times and each intersection requires d_{max} operations. Conceptually this is still the same for the new algorithm, however, the bottleneck is no longer the highest degree vertex in the graph, but, rather the highest degree vertex in the vertex cover. This gives the new algorithm a time complexity of $O(Time_{VC} + \hat{V} \cdot \hat{d}_{max}^2)$, where $O(Time_{VC})$ refers to the time complexity of finding a vertex cover.

For the new algorithm to be considered optimal and not to add additional overhead to the existing complexity, the following requirements needs to be met: $O(Time_{VC}) \leq$

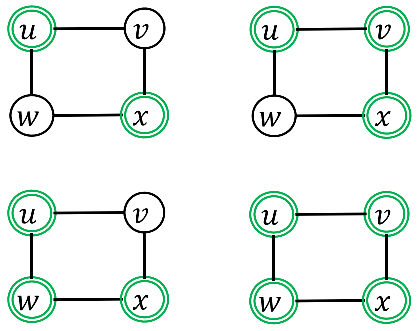

Figure 2. All legal vertex covers of the square made up of u, v, w, and x. Additional edges that are non-relevant have been removed. There are additional vertex covers for square. However, these follow one of the presented patterns. The vertices in the vertex cover are marked with an additional circle.

$O(V \cdot d_{max}^2)$. For practical purposes, an even tighter bound is needed: $O(Time_{VC}) \leq O(V \cdot d_{max}^2 - \hat{V} \cdot \hat{d}_{max}^2)$. These requirements imply that the time spent computing the vertex cover should not be greater than the reduced run-time.

While a trivial vertex cover of $\hat{V} \equiv V$ can be found in $O(1)$, this is essentially the original algorithm. Linear time approximations yield reasonable smaller vertex covers than V in practice. For most linear-time vertex cover algorithms the condition is met almost trivially as $O(V + E)$ is smaller than $O(\hat{V} \cdot \hat{d}_{max}^2)$. This is not true for considerably sparse graphs, where $E \cong 2 \cdot V$, as very little work is required for computing the list intersections. We extend this discussion in Section IV.

C. Circuits of length 4 and higher

A circuit is defined as a simple path of vertices where each vertex and edge is traversed once with the first and last vertices in the path being the same. For example, a triangle is a 3-circuit. Abdo *et al.* [3] suggest using larger circuits than 3-circuits for the purpose of graph analysis. Specifically, they discuss the impact of "long-distance" relationships on the underlying graph structure.

A circuit of length four can be considered a *square* and a circuit of length five can be considered a *pentagon*. From a social network analysis standpoint, finding a square in the network given a specific player, v, is equivalent to finding common friends of any two of v's friends. This can be extended to find circuits of all sizes.

Fig. 2 depicts a subset of different vertex covers for a square. In all these vertex covers, u and x are part of the cover. All non-relevant edges have been removed. Note, in all the examples of Fig. 2 at least one set of vertices that are across from each other (i.e two-hops away) are in the vertex cover. This is mandatory for the vertex cover to be maintained. In our examples, these vertices are u and x. We refer to these vertex-pair as cross-vertices.

Algorithm 2: Pseudo-code for counting 4-circuits given a vertex cover $\hat{V}$.

```
$squares \leftarrow 0$;
for $u \in \hat{V}$ do
    for $x \in \hat{V}$ do
        if $u = x$ then
            next $x$;
        $C \leftarrow intersect(u, adj(u), x, adj(x))$;
        $In\hat{V} \leftarrow \{\}$;
        $notIn\hat{V} \leftarrow \{\}$;
        for $c \in C$ do
            if $c \in \hat{V}$ then
                $In\hat{V} \leftarrow In\hat{V} \cup \{c\}$;
            else
                $notIn\hat{V} \leftarrow notIn\hat{V} \cup \{c\}$;
            $squares \leftarrow squares + |C| \cdot (|C| - 1)$
            $|notIn\hat{V}| \cdot (|notIn\hat{V}| - 1)$
            $+2 \cdot |notIn\hat{V}| \cdot (|In\hat{V}| - 1)$
```

Finding circuits of length four entails taking all vertex pairs and intersecting their adjacencies. Given the adjacency set of two cross-vertices C, if $|C| > 2$ (meaning that the vertices share two or more neighbors) then a square exists. To be exact there are $|C| \cdot (|C| - 1)$ different 4-circuits. We will elaborate on this.

The pseudo-code for finding the number of 4-circuits using the vertex cover approach can be found in Alg. 2. The pseudo-code makes the proper counting correction. Using the cross-vertices observation, we reduce the computed intersection from "all vertex pairs" to "all vertex-cover pairs" without missing a 4-circuit. To get the original algorithm for 4-circuits, the pseudo code requires a simple modification: replace $\hat{V}$ with V in the first two loops and get rid of the counting correction code.

Given u, x are cross-vertices (not necessarily in the vertex cover), now consider the following scenarios when $w, v \in V$ and w, v are cross-vertices (both pairs must be cross vertices for a square to exist). The following scenarios occur:

1) $w, v \notin \hat{V}$. If they are part of the 4-circuit, then they can be found when intersecting the adjacency list of u, x and therefore this intersection can be avoided as it is redundant. If they are not part of a 4-circuit, but simply vertices that are two hops away, then there is no point intersecting their adjacencies allowing a reduced number of intersections.
2) Either $w \in \hat{V} \wedge v \notin \hat{V}$ or $v \in \hat{V} \wedge w \notin \hat{V}$. Without the loss of generality, assume the second case. Once again, for w, v to be part of a 4-circuit, $u, x \in \hat{V}$ cross vertices need to be adjacent to them. For the same reasons as in 1), redundant intersections can be avoided.
3) $w, v \in \hat{V}$. It is possible that they are part of a 4-circuit. If $u, x \in \hat{V}$ are cross vertices, then they are

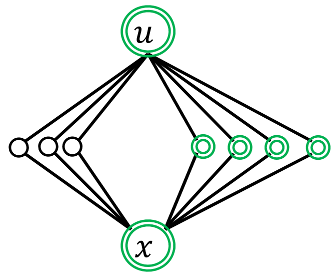

Figure 3. The cross-vertices (u, x) and their common neighboring vertices found in adjacency intersection. These neighbors are divided into groups: those in the vertex cover and those not in the vertex cover. The vertices in the vertex cover are marked with an additional circle.

part of a 4-circuit. Therefore, their adjacencies need to be intersected. As such, the 4-circuit will be found for the intersection of w with v, v with w, u with x, and x with u.

If $u, x \in \hat{V}$, it is possible to swap roles between w, v and u, x and go back to 1) which states that the only intersections that need to be computed are between the vertices in the cover.

The above essentially states the counting corrections that need to be made.

Assume that $u, x \in \hat{V}$, meaning that they are potentially cross-vertices and need to have their lists intersected. In the process of intersecting their adjacencies lists, the set C consists of two subsets, the first set consists of vertices in the vertex cover and is denoted $in\hat{V}$, and the second set consists of vertices that are not in the vertex cover, denoted as $notIn\hat{V}$. Fig. 3 depicts the intersections of two vertices u and x. The vertices belonging to $in\hat{V}$ have been marked with a double circle.

The number of ordered pairs in the intersection is $|C| \cdot (|C| - 1)$, each one of them specifies a different 4-circuit. Remember, that u will be intersected with x and x will be intersected with u and these give different 4-circuits [1].

Because not all vertex pairs have their lists intersected, a counting correction needs to be taken into account. Consider v, w common neighbors of the ordered pair (u, x). If $v, w \in \hat{V}$, then the ordered pair (v, w) will have its neighbors intersected and then u, x will be found. Therefore, no correction needs to be done.

If $v, w \notin \hat{V}$, then the ordered pair (v, w) will not have its adjacencies intersected, therefore, requiring a counting correction as if the adjacencies of the ordered pair (v, w) were intersected and found u, x as its common neighbors. This adds two additional 4-circuits because of the ordering of the neighbors. The ordered pair (w, v) also will not have its neighbors intersected; however, this counting correction is taken into account by the ordered pair (x, u). Therefore, for the 4-circuit consisting of vertices u, v, w, x all eight 4-circles are accounted for. The counting correction needed because of vertices that are not in the vertex cover is $|notIn\hat{V}| \cdot (|notIn\hat{V}| - 1)$.

For the case in which only one of w, v is in the vertex cover, the explanation from above repeats itself. Therefore, the number of cycles that need to be accounted for is essentially the number of ways to order w, v: $2 \cdot |in\hat{V}| \cdot (|notIn\hat{V}| - 1)$.

In this subsection we showed how to find 4-circuits using the vertex cover and applying additional counting correcting techniques. These techniques can be further extended to include the general K-circuit case for $K \geq 3$. However, the general K-circuit case is outside the scope of this paper.

D. Clustering Coefficients in Dynamic Graphs

In this subsection we show that our new vertex cover clustering coefficients approach can also be applied to dynamic graphs. The two types of operations that we consider for dynamic graphs are edge insertions and edge deletions. Vertex insertion and deletion are simple. A vertex insertion places a new vertex in the graph without any edges. A vertex deletion can be serialized as a set of edge deletions.

For an inserted edge, $e = (u, v)$, into the graph the following three scenarios can arise:

1) $u, v \notin \hat{V}$
2) $(u \in \hat{V} \wedge v \notin \hat{V})$ or $(v \in \hat{V} \wedge u \notin \hat{V})$
3) $u, v \in \hat{V}$

The global and local clustering coefficients are handled slightly differently. Again, we focus on the global case.

For the first case, where both vertices are not in the vertex cover, one of the vertices has to be added to the vertex cover to ensure that all edges are covered. For simplicity assume that v is added to the vertex cover. The inserted edge can potentially close several triangles. The second case is similar to the first but does not require modifying the vertex cover. Obviously, for the third case when both vertices in the vertex cover, the vertex cover does not require any modification.

Edge deletions can be dealt with in a similar fashion. However, when deleting $e = (u, v)$, the first scenario in which $u, v \notin \hat{V}$ is not possible as this violates the vertex cover's properties.

IV. Empirical Results

In this section we present empirical performance results of the new clustering coefficients algorithm for both triangle counting and square counting. In our tests we used an Intel i7-2600K system with 16GB of memory. The clock frequency is 3.4GHz. We used graphs taken from the 10th DIMACS Implementation Challenge on Graph Partitioning and Graph Clustering [4]. The graphs we used can be found in Table II. The global clustering coefficient value for these graphs are presented in Fig. 4.

[1]This can be avoided if lexicographical sorting is used.

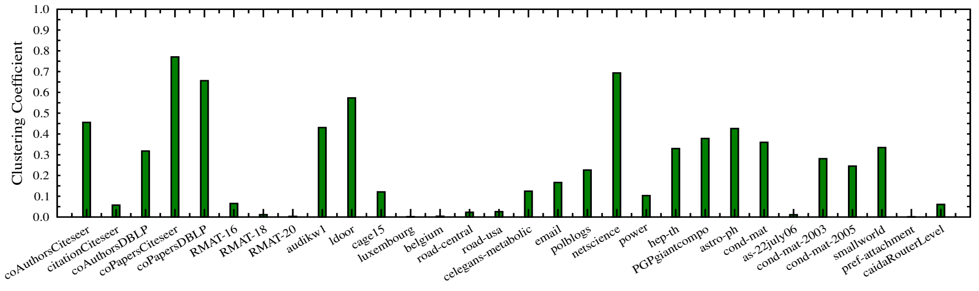

Figure 4. This chart shows the global clustering coefficient value for graphs from Table II.

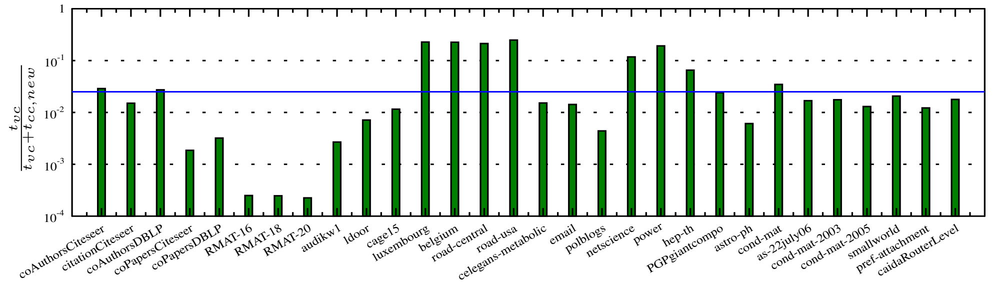

Figure 5. The ordinate presents the ratio of time spent finding the vertex cover as a function of the total time spent computing the vertex cover and the clustering coefficients. The ordinate is a logscale. An additional blue curve has been placed at 2.5%. The bars of about $2/3$ of the graphs are below this curve.

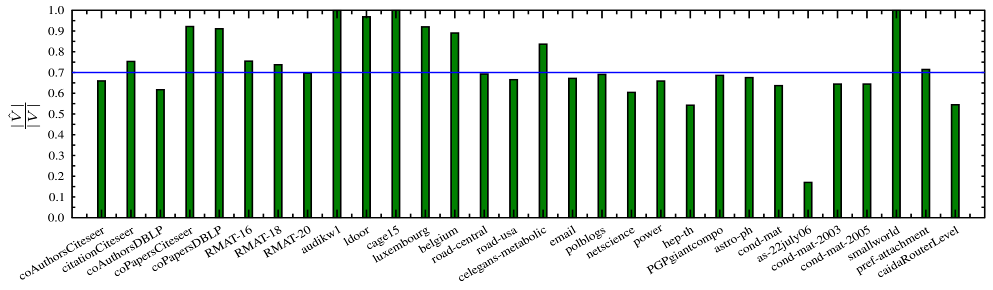

Figure 6. The ordinate is the size ratio of the vertex cover, $\hat{V}$, and the vertex set V. Note that $|\hat{V}| \leq |V|$. An additional blue curve has been added at 70%. The bars of about $3/5$ of the graphs are below this curve.

For benchmarking purposes, we used the clustering coefficient algorithm from [13]. This is a highly optimized CSR implementation of clustering coefficients. As such we have used this CSR clustering coefficient implementation as our baseline and have implemented our new algorithms based using the list intersection taken from the implementation.

We use is a simple linear time greedy algorithm, $O(V+E)$ for finding the vertex cover. For each vertex v in the graph, we check if v has any neighbors that are not within in the vertex cover. If this is the case, v is added to the vertex cover. This algorithm by no means ensures a small or minimal vertex cover. Nonetheless, for our needs it is sufficient as it gave small enough vertex covers for us to see improvements.

A. Static Clustering Coefficients

The overhead to compute a reasonable vertex cover is negligible for many graphs. Fig. 5 depicts the portion of time needed to compute the vertex cover out of the total needed for computing the clustering coefficients. Note that the ordinate of Fig. 5 is a log-scale which shows the time spent computing the vertex cover out of the total time of the new algorithm (time for the vertex cover and the clustering coefficient algorithm that uses the vertex cover), and lower is better. In Fig. 5 an additional (constant) curve has been added at 2.5%. For many of the graphs, the vertex cover takes less than 2.5% of the total time to compute.

Fig. 6 depicts the size of the vertex cover in comparison with the entire vertex set V. An additional constant curve has

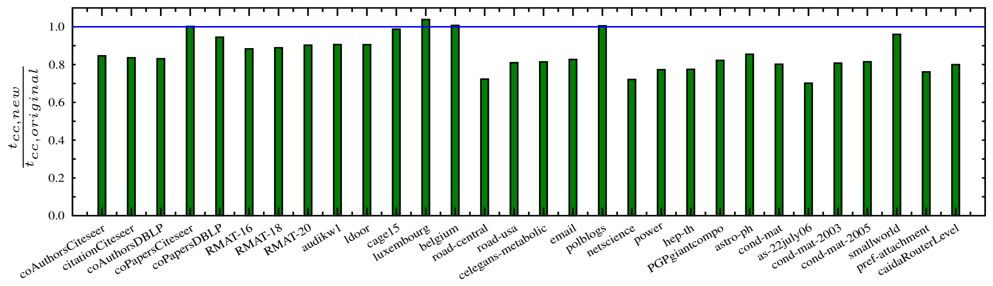

Figure 7. The ordinate is the time ratio of the new vertex cover-based clustering coefficients algorithm and the original clustering coefficient algorithm. The time of the new algorithm includes both the time needed to find the vertex cover and the time for computing the clustering coefficients. All bars below the blue curve occur when the new algorithm is faster, as such lower is better.

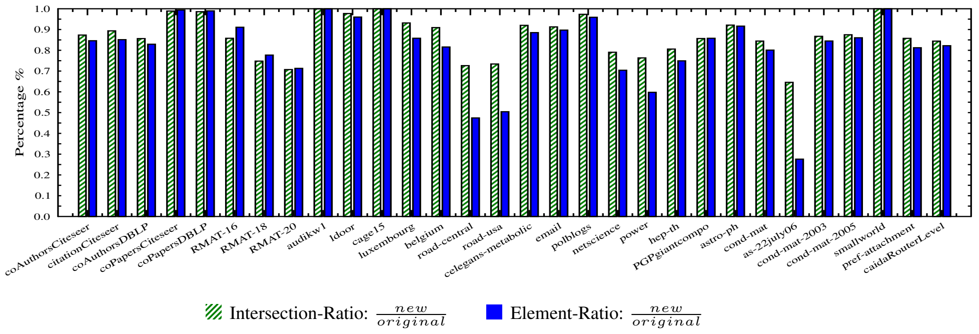

Figure 8. The ordinate is the ratio of the number of intersection that are necessary by the new algorithm in comparison with the original algorithm.

been added at 70%. For slightly more than half the graphs, the size of the vertex cover is less than 70% the size of the entire vertex set.

Fig. 7 depicts the ratio of the time it takes to compute the global clustering coefficients using our new method over the original algorithm. For this figure, lower is better as this means that the new algorithm takes a fraction of the original run-time. An additional blue curve has been added to the figure to state when the performance of the two algorithms is the same. There are several graphs that do not benefit from our new approach. These are graphs for which the vertex cover is roughly the size of $|V|$. A different vertex cover (by a different algorithm) might possibly reduce the size of the vertex cover and allow for shorter run times. However, this was not the focus of our work.

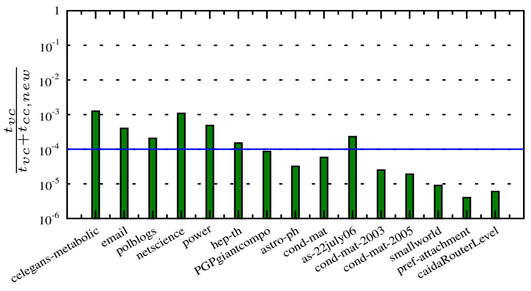

Figure 9. The ordinate presents the ratio of time spent finding the vertex cover as a function of the total time spent finding all the 4-circuits. Note that the ordinate is a log scale. The abscissa are the Clustering graphs from Table II.

For six graphs that we tested, the time for finding the vertex cover took over 10% of the total time. Out of these six graphs, four are road networks. The reason the vertex cover takes such a high percentage of the total time is due to the considerable sparsity of these graphs, where$|E| \leq 2 \cdot |V|$. These graphs have low clustering coefficients and the amount of intersections needed is considerably small. It is not surprising that for these graphs the total time of our new algorithm is greater than the time using the original approach as our vertex cover added overhead. Fortunately, this overhead is negligible.

In Fig. 8 two different bar charts are shown, both are ratio values from 0 to 1, such that the ratios are between our new algorithm and the original one. The green patterned bars are the ratio between the number of adjacency lists intersected using the new method and the respective number using the original method. The blue solid bars are the ratio between the actual number of elements intersected for both methods.

Table II
GRAPHS FROM THE 10TH DIMACS IMPLEMENTATION CHALLENGE THAT ARE USED IN OUR EXPERIMENTS, GROUPED BY TYPE AND SORTED WITHIN EACH GROUP BY VERTEX COUNT.

Name	$\|V\|$	$\|E\|$
Collaboration Networks		
coAuthorsCiteseer	227,320	814,134
citationCiteseer	268,495	1,156,647
coAuthorsDBLP	299,067	977,676
coPapersCiteseer	434,102	16,036,720
coPapersDBLP	540,486	15,245,729
Random Networks		
RMAT-16	65,536	2,456,071
RMAT-18	262,144	10,582,686
RMAT-20	1,048,576	44,619,402
matrix		
audikw1	943,695	38,354,076
ldoor	952,203	22,785,136
cage15	5,154,859	47,022,346
Road Networks		
luxembourg.osm	114,599	119,666
belgium.osm	1,441,295	1,549,970
road_central	14,081,816	16,933,413
road_usa	23,947,347	28,854,312
clustering		
celegans_metabolic	453	2025
email	1,133	5,451
polblogs	1,490	16,715
netscience	1,589	2,742
power	4,941	6,594
hep-th	8,361	15,751
PGPgiantcompo	10,680	24,316
astro-ph	16,706	121,251
cond-mat	16,726	47,594
as-22july06	22,963	48,436
cond-mat-2003	31,163	120,029
cond-mat-2005	40,421	175,691
smallworld	100,000	499,998
preferentialAttachment	100,000	499,985
caidaRouterLevel	192,244	609,066

These two ratios are slightly different. The first ratio states how many intersections need to be computed. The second ratio states how many comparisons need to be done as part of the list intersection. The second ratio takes into account that the lists are not equal length. This is expected in graphs following power-law distribution [6], [9], [16]. For graphs with a uniform distribution of edges, such as the Erdös-Rényi ([14], [15]) random graph model, it is very likely that these ratios would be similar. Further this non-uniformity plays a critical part of the time complexity of the algorithm, specifically d_{max}.

As expected, the graphs that had larger vertex covers (when $|\hat{V}| \cong |V|$) did not have a reduction in the number of intersections or the number of elements intersected. The graphs with the smaller vertex covers had fewer intersections, fewer comparisons, and reduced runtime.

As the number of intersections of our algorithm is bounded by the number of intersections of the original algorithm (which computes all the intersections) these ratios are smaller than one.

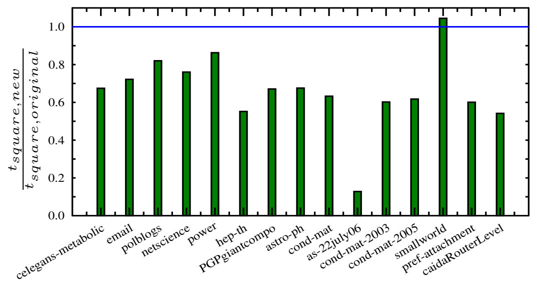

Figure 10. The ordinate is the time ratio of the new vertex cover based clustering coefficients algorithm and the original clustering coefficient algorithm. The time of the new algorithm includes both the time needed to find the vertex cover and the time for computing the clustering coefficients. The blue curve denotes equal run-times for the new and original algorithm. All bars below the blue curve state the new algorithm is faster, as such lower is better.

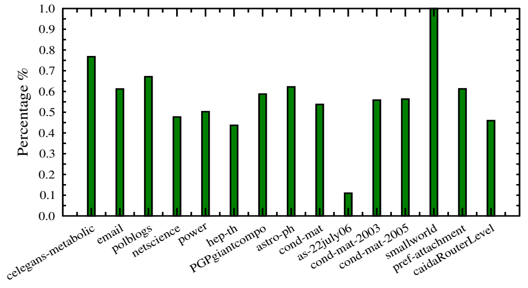

Figure 11. The ordinate is the ratio of the number of intersection that are necessary by the new algorithm in comparison with the original algorithm.

In the context of performance, the smaller these ratios are the better the performance will be as these ratios are correlated to the number of list intersections that need to be done. As such, the lower the ratios, the better.

B. Static 4-Circuits

In this subsection we present performance results for finding 4-circuits using the algorithm presented in Section III-C. A key challenge of the 4-circuit counting is the increased time complexity of the algorithm. As such, for the 4-circuits counting we present results for the smaller graphs that belong to the clustering subset of graphs taken from Table II. We note that finding the 4-circuits for larger graphs can be parallelized to further reduce the running time.

Fig. 9 depicts the time spent computing the vertex cover vs. the time to compute the vertex cover and find the 4-circuits. For almost all the graphs, the time to compute the vertex cover is less that 0.1%. For several graphs the vertex covers takes less than 0.001% of the total time. We use the same vertex cover algorithm as before and the running times are the same as before. As the circuits get longer the time spent on finding the vertex cover becomes a smaller fraction of the total running time.

Fig. 10 depicts the time ratio between the new algorithm (including time spent on finding the vertex cover) and the original algorithm for computing 4-circuits. The blue curve at $y = 1$ denotes when the execution time of both algorithms is the same. Once again, any bar below the blue curve means that the new algorithm is faster than the original algorithm. For half the graphs the new algorithm is 30% faster than the previous algorithm which is due to fewer intersections. Fig. 11 depicts the ratio of the number of comparisons made during the list intersections for the new and original algorithms.

V. Conclusions

In summary, in this paper we design and implement a new method for computing exact clustering coefficients using vertex covers. This method reduces the number of list intersections and avoids unnecessary element comparisons. The two key differences between our new algorithm and the original approach are as follows: 1) our algorithm computes a vertex cover of the graph and 2) our algorithm applies a counting correcting technique that makes up for triangles that are not counted multiple times.

We show that the new algorithm is both exact (gives the same results as the original algorithm) and correct (by proofs). We then show that our new approach can also be used on circuits of length four and can be extended to longer circuits. All these are followed by performance analysis in which we showed that the new algorithm is indeed faster, $15\% - 20\%$ for the 3-circuit and $30\% - 40\%$ for the 4-circuit, than the previous algorithms.

While the focus of work was to show the viability of the vertex cover for clustering coefficients, we believe that the vertex cover approach might be applicable to additional social network analytics, perhaps community detection and modularity-based algorithms. Several open problems related to this work are:

1) Finding additional analytics that can benefit from the vertex cover.
2) Testing the sensitivity of the algorithm by changing the vertex cover algorithm. This includes using additional vertex cover algorithms or using the same vertex cover algorithm as we did that accesses the vertices in a different order (this should give a different cover).
3) Similar to the previous issue, is it possible to select a vertex cover algorithm based on properties of the input graph.
4) For the dynamic graph case, how does the vertex cover change over time? Does the vertex cover gradually increase in size over time until its the size of the vertex set or is the vertex cover near constant over the insertion?
5) Creating an efficient algorithm for finding a K-circuit for some given K using the vertex cover approach.
6) Is there an effective way to parallelize the new algorithm?

Acknowledgments

Funding was provided by the U.S. Army Research Office (ARO) and Defense Advanced Research Projects Agency (DARPA) under Contract Number W911NF-11-C-0088. The content of the information in this document does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

The authors thank David Ediger, Robert McColl, Lluis Miquel Munguia, and Anita Zakrzewska, for their suggestion and useful comments that have improved the quality of this paper.

References

[1] "Facebook," 2013. [Online]. Available: https://www.facebook.com/

[2] "Twitter," 2013. [Online]. Available: https://twitter.com/

[3] A. H. Abdo and A. P. S. de Moura, "Clustering as a Measure of the Local Topology of Networks," *arXiv physics/0605235*, 2006.

[4] D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, *10th DIMACS Implementation Challenge on Graph Partitioning and Graph Clustering*. American Mathematical Society, 2013, vol. 588.

[5] R. Bar-Yehuda and S. Even, "A Linear-Time Approximation Algorithm for the Weighted Vertex Cover Problem," *Journal of Algorithms*, vol. 2, no. 2, pp. 198–203, 1981.

[6] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[7] A.-L. Barabási and Z. N. Oltvai, "Network Biology: Understanding the Cell's Functional Organization," *Nature Reviews Genetics*, vol. 5, no. 2, pp. 101–113, 2004.

[8] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis, "Efficient Semi-streaming Algorithms for Local Triangle Counting in Massive Graphs," in *Proceedings of the 14th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*. ACM, 2008, pp. 16–24.

[9] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, "Graph Structure in the Web," *Computer Networks*, vol. 33, no. 1, pp. 309–320, 2000.

[10] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler, "Counting Triangles in Data Streams," in *Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems*. ACM, 2006, pp. 253–262.

[11] D. Coppersmith and S. Winograd, “Matrix Multiplication via Arithmetic Progressions,” *Journal of Symbolic Computation*, vol. 9, no. 3, pp. 251–280, 1990.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT press, 2001.

[13] D. Ediger, K. Jiang, J. Riedy, and D. A. Bader, “Massive Streaming Data Analytics: A Case Study with Clustering Coefficients,” in *International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum (IPDPSW)*. IEEE, 2010, pp. 1–8.

[14] P. Erdös and A. Rényi, “On Random Graphs I,” *Publicationes Mathematicae*, pp. 290–297, June 1959.

[15] ——, “The Evolution of Random Graphs,” *Magyar Tud. Akad. Mat.*, pp. 17–61, 1960.

[16] M. Faloutsos, P. Faloutsos, and C. Faloutsos, “On Power-Law Relationships of The Internet Topology,” in *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4. ACM, 1999, pp. 251–262.

[17] M. R. Garey and D. S. Johnson, *Computers and Intractability*. freeman New York, 1979, vol. 174.

[18] D. S. Hochbaum, “Approximation Algorithms for the Set Covering and Vertex Cover Problems,” *SIAM Journal on Computing*, vol. 11, no. 3, pp. 555–556, 1982.

[19] A. Israeli and Y. Shiloach, “An Improved Parallel Algorithm for Maximal Matching,” *Information Processing Letters*, vol. 22, no. 2, pp. 57–60, 1986.

[20] S. Khuller, U. Vishkin, and N. Young, “A Primal-dual Parallel Approximation Technique Applied to Weighted Set and Vertex Covers,” *J. Algorithms*, vol. 17, no. 2, pp. 280–289, 1994.

[21] V. Kumar, K. Ramesh, and V. N. Rao, “Parallel Best-First Search of State-space Graphs: A Summary of Results,” in *Proceedings of the 1988 National Conference on Artificial Intelligence. Morgan Kaufmann*. Citeseer, 1988.

[22] A. Leist, K. Hawick, D. Playne, and N. S. Albany, “GPGPU and Multi-Core Architectures for Computing Clustering Coefficients of Irregular Graphs,” in *Proc. Int’l Conf. on Scientific Computing (CSC’11)*, 2011.

[23] J. Leskovec, J. Kleinberg, and C. Faloutsos, “Graphs Over Time: Densification Laws, Shrinking Diameters and Possible Explanations,” in *Proceedings of the 11th ACM SIGKDD Int’l Conf. on Knowledge Discovery in Data Mining*. ACM, 2005, pp. 177–187.

[24] S. Milgram, “The Small World Problem,” *Psychology Today*, vol. 2, no. 1, pp. 60–67, 1967.

[25] T. Schank and D. Wagner, “Finding, Counting and Listing All Triangles in Large Graphs, an Experimental Study,” in *Experimental and Efficient Algorithms*. Springer, 2005, pp. 606–609.

[26] Y. Shavitt and E. Shir, “DIMES: Let the Internet Measure Itself,” *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 71–74, 2005.

[27] S. H. Strogatz, “Exploring Complex Networks,” *Nature*, vol. 410, no. 6825, pp. 268–276, 2001.

[28] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow, “The Anatomy of the Facebook Social Graph,” *arXiv preprint arXiv:1111.4503*, 2011.

[29] D. J. Watts and S. H. Strogatz, “Collective Dynamics of “Small-World” Networks,” *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.