MASTER'S THESIS

Detecting Local Clusters in Large Networks Using HyperLogLog

Lotte Weedage
August 14, 2020

Master Applied Mathematics
Stochastic Operations Research

Supervisors:
Prof. Dr. N. V. Litvak
Dr. C. Stegehuis

Graduation committee:
Prof. Dr. N. V. Litvak
Dr. C. Stegehuis
Dr. J. D. Backhoff

UNIVERSITY OF TWENTE.

Abstract

In this research we investigate how to find local clusters in large networks based on three measures: conductance, number of triangles and transitivity. We introduce adaptations of the HyperBall algorithm [10], based on the idea of HyperLogLog counters [15] to find these measures. First of all, we find the conductance of ball subgraphs using the HyperBall algorithm. We also introduce a method to locally count triangles in large networks, where we find the triangle count of ball subgraphs around every node simultaneously. This is useful to identify where in a graph high densities of triangles can be found. Moreover, we introduce a method to find the ball subgraph transitivity in large networks and find clustered groups of nodes in these networks. Lastly, we show methods of extending the HyperBall algorithm to find these three measures in directed, weighted or temporal graphs. The three measures (conductance, number of triangles and transitivity) identify clusters in graphs themselves, but also turn out to be good seed sets for exact community detection algorithms as PageRank-Nibble. We tested the new algorithms on synthetic graphs generated using the LFR model [23] and real-life graphs. The generated graphs behave different than the real-life large networks, and in both kinds of networks the three introduced measures are promising to find clusters.

Keywords: HyperLogLog, Approximate counting, Conductance, Ball subgraphs, Triangles, Clusters, Community detection, Transitivity

Contents

1 Problem statement **3**
- 1.1 Introduction . . . 3
- 1.2 Contribution . . . 5
- 1.3 Notation and definitions . . . 5
- 1.4 Structure of thesis . . . 6

2 Approximate probabilistic counting in graphs **7**
- 2.1 HyperLogLog . . . 7
- 2.2 HyperBall . . . 10
- 2.3 HyperEdgeball . . . 11
- 2.4 Directed, weighted and temporal graphs . . . 11

3 Conductance **12**
- 3.1 Community detection . . . 12
- 3.2 Conductance using HyperBall . . . 14
- 3.3 Discussion . . . 20

4 Triangles **21**
- 4.1 Community detection . . . 21
- 4.2 Counting triangles using HyperBall . . . 24
- 4.3 Transitivity using HyperBall . . . 26
- 4.4 Discussion . . . 27

5 Results **28**
- 5.1 Numerical results for LFR graphs . . . 28
- 5.2 Numerical results for real life graphs . . . 38

6 Conclusion and Discussion **44**
- 6.1 Conclusion . . . 44
- 6.2 Discussion . . . 46

Bibliography **48**

A Glossary **51**

1 Problem statement

> "Networks are everywhere, from the internet, to social networks, and the genetic networks that determine our biological existence." – Albert-László Barabási

1.1 Introduction

Network science is a topic that has interested scientists for decades: everyone wants to uncover the hidden structures in networks around us to learn more about large networks in order to predict and control phenomena in these networks. Recent developments show this very well: when knowing more about virus spreading in large networks, pandemics can be detected earlier or hotspots can be identified. A lot of progress has already been made in finding interesting properties of networks, but since networks are growing and growing and becoming a larger part of people's lives, this topic keeps bringing up new questions.

In the last couple of years, *community detection* has gradually become one of the most popular topics in network science. For a lot of disciplines it is useful to find *communities*: a set of nodes with similar properties. For example, finding communities can be interesting to find similar products in a web shop or to find groups of people on social media. A way to find those communities is by finding clustered groups of nodes: groups of nodes that have a lot of connections within that group in comparison to the outside of that group. One of the challenges in finding these clusters is how to measure the quality, for which many different measures are developed.

Figure 1.1: Example of a graph with clusters

A measure that is commonly used to measure the quality of clusters in a graph is the conductance of a cluster. The conductance gives a ratio of how many edges there are connected to nodes outside the cluster relative to edges connected with nodes inside the cluster. Since conductance is based on the *min-cut*[39] of a graph, finding clusters using conductance is called a *cut-based* method. According to Schaeffer [34], using conductance as a measure to find communities is one of the most useful and important cut-based methods that exist. Finding the conductance of ball subgraphs $S_r(v)$: graphs that are generated by the nodes within radius r of node v, can be useful on its own to find clustered balls around a node, but the minimal conductance ball can also be used as a *seed set* for other more time- and memory-consuming algorithms such as PageRank Nibble [4] or the Multi Walker Chain model [8].

Another measure that can be used to find a clustered group of nodes in a graph is the number of triangles in that subgraph. A triangle in a graph is the most clustered subgraph consisting of three nodes, since there cannot be more than three edges between three nodes. In social graphs, there is an abundance of triangles since it is likely that friends of your friends are also your friends. Moreover, finding triangles on itself already has interesting applications in for example biological networks [40], spam detection [7] or link recommendations [41]. Finding clustered sets can be done by directly counting the number of triangles in a graph, or by finding the *transitivity* of a (sub)graph, which is the ratio of triangles versus wedges in a graph and therefore tells us how clustered this graph is (Figure 1.2).

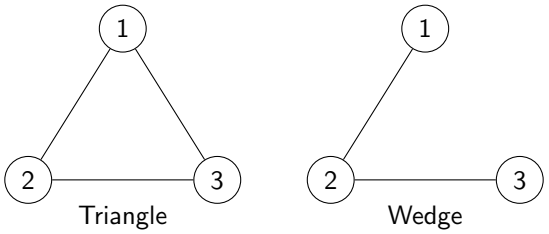


Figure 1.2: Example of a triangle and a wedge. It can be noticed that a triangle consists of three wedges.

Finding these clusters is not a trivial task when looking at the size of the graphs: scientists want to find clusters in graphs that are becoming larger and larger, which makes it impossible to store graphs in a reasonable amount of memory and to find these clusters in a reasonable amount of time. Since this makes it impossible to count the number of nodes, edges or triangles in an exact way, we need to process these large networks in a different way and resort to *approximate counting*. Instead of deterministically finding all properties in a graph, it is possible to make an approximation in less time and with less memory.

An example of approximate counting is given in [15], where cardinalities in large streams of data are found using HyperLogLog counters. The goal of these HyperLogLog counters is to probabilistically count the number of distinct elements in large streams of data, such as the number of unique visitors on a web page or the number of different genomes in biological data. Since these counters give an accurate estimate of large cardinalities, and moreover are memory-efficient, these HyperLogLog counters are proven to be very useful.

A couple years later, Boldi and Vigna [9] found out that these HyperLogLog counters could also be used to find centrality measures in large graphs. They adapted the HyperLogLog algorithm into the HyperBall algorithm, which counts the number of nodes in a ball subgraph $S_r(v)$ for every node v and every radius r. Last year, a bachelor's thesis about HyperBall expanded the idea of counting nodes in a ball subgraph to counting edges in a ball subgraph, which was introduced as the HyperEdgeball algorithm [19].

In this thesis, we find clustered subgraphs using the method of HyperLogLog counters in three ways: by taking the *conductance* of a subgraph as a measure for good communities and by looking into the number of triangles and wedges in order to find clustered ball subgraphs around a node based on the number of triangles or based on the transitivity.

1.2 Contribution

The contribution of this work is divided into three categories. First of all, we introduce new applications of the HyperBall algorithm in order to count triangles and wedges which makes it possible to find the transitivity in ball subgraphs. We introduce the out-edgeball of a node, which consists of all edges in an undirected graph as directed edges out of a node, in order to be able to use the HyperBall algorithm to estimate conductance as stated in Theorem 2 and Theorem 3. We also find error bounds for the triangle, transitivity and conductance estimator by using Chebyshev's inequality and Vysochanskij-Petunin's inequality.

Moreover, by using the HyperBall algorithm to find the conductance and transitivity or to count the number of triangles in a graph, we can find these three measures in all ball subgraphs around all nodes in a graph at the same time, which also shows us where high densities of nodes and edges are located.

The last contribution of this thesis is that we show that the nodes that have a small conductance in their corresponding ball subgraphs, a large number of triangles or a high transitivity, are promising seed sets for algorithms as PageRank-Nibble and the Multi Walker Chain model in order to easily find communities in large networks. These seed sets work better than using a group of high-degree nodes or a random set of nodes as seed sets.

1.3 Notation and definitions

In this thesis, we will use the following notation: let $G = (V, E)$ be a graph with $n = |V|$ nodes and $m = |E|$ edges. Moreover, we define the *neighbourhood* $\mathcal{N}(v)$ of node v as the set of all nodes incident to node v.

Definition 1 (Nodeball). *The nodeball $\mathcal{B}_r(v)$ consists of every node in a ball of radius r around node v. Let $\mathcal{B}_0(v) = \{v\}$, and for $r > 1$:*

$$\mathcal{B}_r(v) = \bigcup_{w:\{v,w\}\in E} \mathcal{B}_{r-1}(v) \cup \mathcal{B}_{r-1}(w).$$

Definition 2 (Edgeball). *The edgeball $\mathcal{E}_0(v)$ consists of every edge incident to node v. Then, for $r > 1$:*

$$\mathcal{E}_r(v) = \bigcup_{w:\{v,w\}\in E} \mathcal{E}_{r-1}(v) \cup \mathcal{E}_{r-1}(w).$$

Definition 3 (Triangle ball). *The triangle ball $\Delta_0(v)$ consists of every triangle that contains node v. Then, for $r > 1$:*

$$\Delta_r(v) = \bigcup_{w:\{v,w\}\in E} \Delta_{r-1}(w) \cup \Delta_r(v).$$

Definition 4 (Ball subgraph). *The ball subgraph $S_r(v)$ with center node v and radius r is a subgraph of G induced by the nodes in $\mathcal{B}_r(v)$.*

These definitions are depicted in Figure 1.3. In Table A.1 in the appendix, we made an overview of the notation that we have used throughout this thesis.

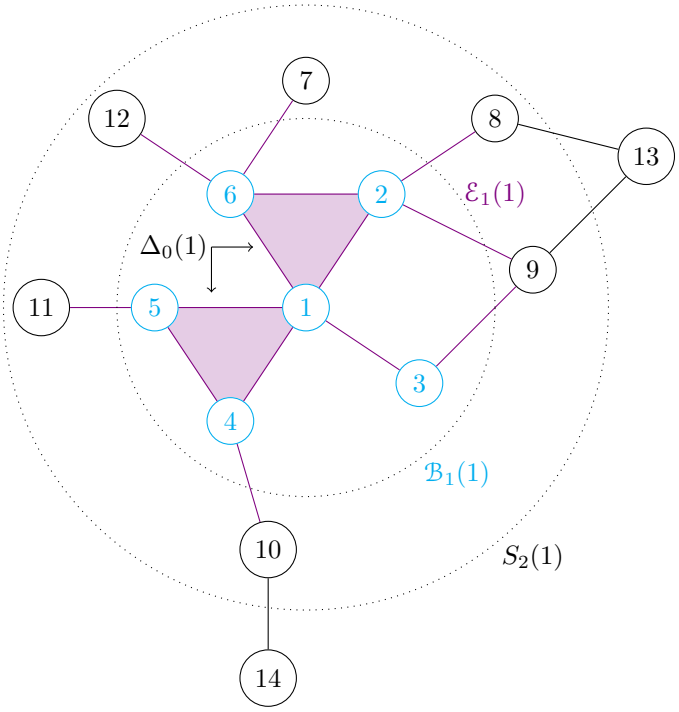


Figure 1.3: Example of a nodeball, edgeball, triangle ball and ball subgraph around node 1 in an undirected graph

1.4 Structure of thesis

The structure of this thesis is as follows. In Chapter 2 we introduce a method of counting nodes and edges in large graphs using HyperLogLog counters. We explain the idea behind these HyperLogLog counters, we introduce the HyperBall and HyperEdgeball algorithms and show the error bounds of the cardinality estimate in Theorem 1. In Chapter 3, we look into clustering by using conductance. We elaborate on two algorithms that use conductance as a measure for community detection and we introduce and prove two theorems that help us express conductance in terms of cardinalities that we can find with the HyperBall algorithm. We introduce a new initialisation for the HyperBall and HyperEdgeball algorithms in order to find an estimate for the conductance and we find the error bounds of this estimator. Then, in Chapter 4, we focus on finding clusters using triangle counting and we discuss several algorithms for exact and approximate triangle counting. We introduce another adaptation of the HyperBall algorithm that finds an estimate for the number of triangles in ball subgraphs and we again discuss error bounds of this estimator. In this chapter we also introduce a way of finding transitivity using the HyperBall algorithm. Chapter 5 is about the implementation of the algorithms and the numerical results. We show the error bounds of the estimators for conductance, number of triangles, transitivity and we compare these results to the numerical results found using real life graphs. Lastly, in Chapter 6, we elaborate on the main findings of this research and we reflect on the research.

2 Approximate probabilistic counting in graphs

In this chapter we explain how we can count the number of nodes and edges in a graph using HyperLogLog counters and we give in introduction of the HyperLogLog counters and the HyperBall and HyperEdgeball algorithm. The HyperBall algorithm, introduced in [10], gives an estimate of the size of the nodeball $\mathcal{B}_r(v)$. Similarly, the HyperEdgeball algorithm which was introduced by [19] estimates the size of the edgeball $\mathcal{E}_r(v)$. These algorithms use *HyperLogLog* counters to estimate the number of distinct elements in this node- or edgeball. In the following sections, we explain how these HyperLogLog counters work, we give the error bounds of the estimated cardinality and we discuss how the HyperBall and HyperEdgeball algorithm work in more detail.

2.1 HyperLogLog

The HyperLogLog algorithm is a probabilistic counting technique that estimates the cardinality of a large dataset by using HyperLogLog counters. A big advantage of this algorithm is that it drastically reduces the memory that is needed to count the cardinality of large datasets and therefore is very useful for these datasets where it is impossible to calculate the cardinality of a set deterministically.

The HyperLogLog algorithm uses a hash function $h : D \to \{0,1\}^\infty$ which assigns every element of the dataset a binary string. The hash function is defined in such a way that it can be assumed that *every bit of the hashed value is independent and has a probability of* $1/2$ *of occurring* [15]. This observation is important since this gives us the chance to use the principle of *bit-pattern observables*: a certain bit-pattern at for example the beginning of the binary string has a certain probability of happening and if we see that certain pattern, we can give an estimate of the cardinality of the observed set.

The pseudocode of the HyperLogLog algorithm is given in Algorithm 1. The input of this algorithm is a *multiset* $\mathcal{M}$ of data items: a stream of elements that are read in order of occurrence. The algorithm initialises an empty counter with $p = 2^b$ registers, where every register corresponds to an entry of the counter. The more registers we use, the more precise the cardinality estimate will be.

Algorithm 1 The HyperLogLog algorithm as described in [15], which approximates the cardinality of a data stream.

Let $h_b(x)$ be the first b bits of the hashed value of element x
Let $h^b(x)$ be the other part of the hashed value of element x
Let $\rho(h^b(x))$ be the position of the leftmost 1-bit ($\rho(001\cdots) = 3$).

initialise a collection of $p = 2^b$ registers, $M[1], \ldots, M[p], to -\infty$.

function $\textsc{Add}(M$:counter, x: item)
    $i \leftarrow h_b(x)$
    $M[i] \leftarrow \max\{M[i], \rho(h^b(x))\}$
end function

function $\textsc{Size}(M$: counter)
    $Z \leftarrow \left(\sum_{j=0}^{p-1} 2^{-M[j]}\right)^{-1}$
    $E \leftarrow \alpha_p p^2 Z$
    return E
end function

for each $x \in \mathcal{M}$ **do**
    $\textsc{Add}(M, x)$
end for

return $\textsc{Size}(M)$

The HyperLogLog algorithm approximates the size E of the multiset M by using the following formula:

$$E := \frac{\alpha_p p^2}{\sum_{j=1}^{p} 2^{-M[j]}}, \qquad \text{with } \alpha_p := \left(p \int_0^\infty \left(\log_2 \left(\frac{2+u}{1+u} \right) \right)^p \mathrm{d}u \right)^{-1} \tag{2.1}$$

The expectation and variance of this estimator are given in Theorem 1 of [15]:

Theorem 1. *Let the algorithm* HYPERLOGLOG *be applied to an ideal multiset of (unknown) cardinality n, using $p \geq 3$ registers, and let E be the resulting cardinality estimate.*

(i) The estimate E is asymptotically almost unbiased in the sense that, as $n \to \infty$,

$$\frac{1}{n}\mathrm{E}(E) = 1 + \delta_1(n) + o(1), \text{ where } |\delta_1(n)| < 5 \cdot 10^{-5} \text{ as soon as } p \geq 2^4.$$

(ii) The standard error defined as $\frac{1}{n}\sqrt{\mathrm{Var}(E)}$ satisfies as $n \to \infty$,

$$\frac{1}{n}\sqrt{\mathrm{Var}(E)} = \frac{\beta_p}{\sqrt{p}} + \delta_2(n) + o(1), \text{ where } |\delta_2(n)| < 5 \cdot 10^{-4} \text{ as soon as } p \geq 2^4,$$

the constants β_p being bounded, with $\beta_{16} = 1.106, \beta_{32} = 1.070, \beta_{64} = 1.054, \beta_{128} = 1.046$, and $\beta_\infty = \sqrt{3\log(2) - 1} \approx 1.03896$.

In (2.1), α_p is a constant that will correct the bias in the estimation. Since this algorithm is designed for large datasets, the algorithm does not work well for smaller datasets. When having smaller datasets, not all registers will be filled and therefore some registers still have the value $-\infty$ after execution of the algorithm. This will give large errors for small datasets. Moreover, for extremely

large datasets hash collisions become more and more likely which will also badly influence the quality of the estimation. To counteract for these facts the authors of [15] propose a slightly changed algorithm, described in Algorithm 2.

Algorithm 2 The improved HyperLogLog algorithm as described in [15], with bias correction for small and very large cardinalities.

1: *Let* $h_b(x)$ *be the first* b *bits of the hashed value of element* x
2: *Let* $h^b(x)$ *be the other part of the hashed value of element* x
3: *Let* $\rho(h^b(x))$ *be the position of the leftmost 1-bit (*$\rho(001\cdots) = 3$*).*
4:
5: **initialise** a collection of $p = 2^b$ registers, $M[1], \ldots, M[p]$, to 0.
6:
7: **function** $\textsc{Add}(M$:counter, x: item$)$
8: $\quad i \leftarrow h_b(x)$
9: $\quad M[i] \leftarrow \max\{M[i], \rho(h^b(x))\}$
10: **end function**
11:
12: **function** $\textsc{Size}(M$: counter$)$
13: $\quad Z \leftarrow \left(\sum_{j=0}^{p-1} 2^{-M[j]}\right)^{-1}$
14: $\quad E \leftarrow \alpha_p p^2 Z$
15: $\quad$ **return** E
16: **end function**
17:
18: **function** $\textsc{LinearCounting}(p, V)$
19: $\quad$ **return** $p \log(p/V)$
20: **end function**
21:
22: **for each** $x \in \mathcal{M}$ **do** ▷ start of the algorithm
23: $\quad \textsc{Add}(M, x)$
24: **end for**
25:
26: $E := \textsc{Size}(M)$
27: **if** $E \leq 5/2p$ **then**
28: $\quad$ **let** V be the number of registers equal to 0
29: $\quad$ **if** $V \neq 0$ **then**
30: $\quad\quad E^* := \textsc{LinearCounting}(p, V)$ ▷ small range corrections
31: $\quad$ **else**
32: $\quad\quad E^* := E$
33: $\quad$ **end if**
34: **else if** $E \leq \frac{1}{32} 2^{32}$ **then**
35: $\quad E^* := E$ ▷ intermediate range - no correction
36: **else**
37: $\quad E^* := -2^{32} \log(1 - E/2^{32})$ ▷ large range correction
38: **end if**
39:
40: **return** E^*

In this version, the counters are initialised to zero instead of $-\infty$ in line 5. Moreover, for small datasets, $\textsc{LinearCounting}$ [44] is used and for large datasets the probability for a hash collision is taken into account. With $\textsc{LinearCounting}$, instead of using all registers of the HyperLogLog counter, only the nonzero registers are taken into account to find the cardinality of the set, since it is not likely that all registers of the counter are visited when the cardinality of the set is small.

2.2 HyperBall

In 2013, *Boldi* and *Vigna* introduced the HyperBall algorithm [10]. This algorithm estimates the size of the ball around a node by using HyperLogLog counters [15]. The algorithm is an adaptation of the HyperANF algorithm [9], what is based on the fact that the nodeball around node v with radius r $\mathcal{B}_r(v)$ can be found iteratively (Definition 1).

Algorithm 3 The HyperBall algorithm as described in [10], which returns an estimation of the ball cardinality for each node. The functions ADD and SIZE of Algorithm 2 are used.

```
1:  c[−], an array of n HyperLogLog counters
2:
3:  function UNION(M: counter, N: counter)
4:  |  for each i < p do
5:  |  |  M[i] ← max{M[i], N[i]}
6:  |  end for
7:  end function
8:
9:  for each v ∈ V do                                   ▷ Initialisation
10: |  ADD(c[v], v)
11: end for
12:
13: r ← 0
14: repeat
15: |  for each v ∈ V do
16: |  |  a ← c[v]
17: |  |  for each w ∈ N(v) do
18: |  |  |  a → UNION(c[w], a)
19: |  |  end for
20: |  |  write ⟨v, a⟩ to disk, which estimates |B_{r+1}(v)|
21: |  end for
22: |  update the array c[−] with the new ⟨v, a⟩ pairs
23: |  r ← r + 1
24: until no counter changes its value
```

The HyperBall algorithm uses one HyperLogLog counter per node and for each iteration, the counters of the neighbours of this node are added to the node's own counter. After each iteration r, the size of this counter is calculated, which equals $|\hat{\mathcal{B}}_{r+1}(v)|$.

2.3 HyperEdgeball

Instead of counting nodes, it is also possible to count the edges around a node. This adaptation of HyperBall is called HyperEdgeball [19], which gives an approximation of the number of edges around a node after r iterations: $|\hat{\mathcal{E}}_{r+1}(v)|$. The only thing that changes in the algorithm is the initialisation phase (Algorithm 3, lines 9 - 11), since we are now counting edges instead of nodes.

Algorithm 4 The initialisation for the HyperEdgeball algorithm as described in [19], which returns an estimation of the edgeball cardinality $|\mathcal{E}_r(v)|$ for each node after running the HyperBall algorithm. Note: when we use directed graphs, we can use directed edges instead of undirected ones.

```
1: c[−], an array of n HyperLogLog counters
2:
3: for each v ∈ V do                                          ▷ Initialisation
4:     for each e ∈ {{v, w} ∈ E} do
5:         ADD(c[v], w)
6:     end for
7:     write ⟨v, c[v]⟩ to disk, which estimates |E_0(v)|
8: end for
```

2.4 Directed, weighted and temporal graphs

The HyperBall algorithm can easily be extended in order to work with directed or weighted graphs. For directed graphs, a node or edge can be added to the node counter or edge counter when this node or edge is reachable by an edge in the directed graph. This does not change the algorithm, as the underlying idea stays the same.

As described in [10], the HyperBall algorithm can also work for weighted graphs. However, we can only use integer weights. For every node with weight w we make w replicas of that node. When iterating over nodes and its neighbours, every replica of this nodes gets added to the counter. This means that we treat every replica as a new node or edge. Important in this case is that for every replica, a different hash is used, so it will usually be stored in a different register in the HyperLogLog counter. The same will work for weighted edges, since then we make replica's of edges. When using this method for weighted graphs, we need to keep the number of registers in a HyperLogLog counter in mind since adding weights to nodes or edges means that we count more nodes or edges which results in higher cardinalities.

In [5] a method for counting nodes in temporal graphs is introduced. In this method, the authors transform the temporal graph to an event graph, where an event is a group of nodes that is available at a certain time. There is a directed link between two events if there is a link between two nodes in these two events. By using this algorithm, which is based on HyperLogLog counters, it is possible to find the number of nodes available at certain time spans.

3 Conductance

In this chapter, we look into the problem of finding clusters using conductance. We introduce two theorems that make it possible to find the conductance of a ball subgraph by using the HyperEdgeball algorithm. Moreover, we extend the HyperEdgeball algorithm in order to estimate the conductance of a ball subgraph and we discuss the error bounds of this estimator.

3.1 Community detection

A community in a graph is a group of nodes that is very clustered. As said before, finding these communities can be very useful to for example recommend similar articles to customers in a web shop or to identify social groups in large social networks. However, the quality of a cluster can be defined in many ways, which can give us different communities based on the measure that we use. There are a lot of different measures that can be used to qualify these communities, for example the distance between nodes in a cluster, the adjacency of nodes in a cluster, the connectivity of a cluster, the density of a cluster or the cut size, as described in [34]. This survey defines the *minimal conductance set*, found by calculating the conductance of a subgraph, as 'the most important cut-based measure in the context of clustering'.

Definition 5 (Conductance)**.** *For a graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, the conductance of a subgraph $S \subset G$ is:*

$$\phi(S) = \frac{|\delta(S)|}{\min\left(\mathit{vol}(S), 2m - \mathit{vol}(S)\right)},$$

where $\delta(S) = \{\{x, y\} \in E | x \in S, y \notin S\}$ and the volume of S is the sum of the degrees of the nodes in subgraph S.

3.1.1 Conductance-based local clustering methods

In this section we discuss two algorithms that try to find communities by using minimal conductance as a measure. These algorithms are classified as local graph clustering methods, which means that the clusters are found by starting from a single node in the graph. Contrary to global graph clustering, this costs less memory since only a small part of the graph is explored at once. Since methods for global graph clustering use the information of the whole graph to find all clusters in that graph at once, this is unfeasible for large graphs. There are many more local graph clustering algorithms [34], but we chose the PageRank-Nibble algorithm and the Multi Walker Chain Model since these are two commonly used methods that use seed sets to find minimal conductance sets.

PageRank-Nibble

Andersen, Chung and Lang presented an algorithm for local graph partitioning by using PageRank vectors [4] which they called PageRank-Nibble, based on Nibble from [38]. The general idea is

to find communities by starting in a node and finding a set of nodes containing this node that has low conductance by using PageRank vectors. They use a starting vector s consisting of zeros except for a 1 on the index of the starting node, also called *seed*, and then solve the linear system

$$pr_\alpha(s) = \alpha s + (1-\alpha) pr_\alpha(s) W,$$

where $\alpha \in (0, 1]$ is the *teleport probability* and $W = \frac{1}{2}(I + D^{-1}A)$ is the lazy random walk transition matrix as defined in [4]. When having found the PageRank vectors of the seed node, they add the node to the minimal conductance set in order of importance in the PageRank vector until the minimal conductance set is found.

However, it is not possible to find the PageRank vector in an exact way in a reasonable amount of time, and therefore an approximation of the PageRank vector is made by using another PageRank vector $pr_\alpha(s-r)$, where r is nonnegative and $r(v) \leq \epsilon d(v)$ for every $v \in V$ (*ϵ-approximate PageRank vector*, introduced by [21]). They compute this vector r and $p = pr_\alpha(s-r)$ by the $\texttt{ApproximatePR}(s, \alpha, \epsilon)$-algorithm, where they start with $p = 0$ and $r = s$ and look at all the vectors one by one and put some probability from r to p whenever $r(v) \geq \epsilon d(v)$. This approximate algorithm is used since this has a running time of $O\left(\frac{1}{\epsilon\alpha}\right)$ and a smaller upper bound for the support of p in comparison to the exact PageRank algorithm.

The PageRank-Nibble algorithm gives the best community around a seed node, so in order to find all communities in a graph, all nodes need to be tested separately in the PageRank-Nibble algorithm. Therefore, it is useful to pre-select promising nodes that are likely to be in the center of a community. For the PageRank-Nibble algorithm, random starting nodes are used until all communities are found. Another method of selecting starting nodes (*seeds*) is described in [16], where the authors discovered that in graphs with a large global clustering coefficient and a heavy-tailed degree distribution, the vertex neighbourhoods with good conductance scores are a good seed set for the PageRank-Nibble algorithm. Since most large social networks satisfy those two conditions, finding the conductance of ball subgraphs of radius 1, the vertex neighbourhood, could be enough to find communities in these graphs.

Multi Walker Chain model

Bian, Ni et al. also presented a local graph partitioning algorithm, the Multi Walker Chain model (MWC, [8]) which is based on PageRank-Nibble. Instead of only 1 starting node they use k *walkers*. These *walkers* all influence each other. The numerical experiments in [8] show that the MWC model performed better based on the F-score and the variation of the F-score as defined in [8]. A reason for this improvement of the F-score is that the walker has less chance of going outside the community (Figure 1 and 2 of [8]) since it gets averaged by multiple walkers. In each iteration τ, all walkers walk one step and after the iteration the node visiting probability vector of the walkers $\mathbf{x}_k^{(\tau)}$ gets updated:

$$\begin{cases} \mathbf{x}_k^0 = 1 \text{ for the query node} \\ \mathbf{x}_k^{(\tau)} = \alpha P^t \mathbf{x}_k^{(\tau)} + (1-\alpha)\mathbf{v}_k^{(\tau)}, \end{cases}$$

where $\mathbf{v}_k^{(\tau)}$ is the influential vector, which indicates which nodes are influencing the node that we are investigating. When the nodes with highest probability are found, the minimal conductance set can be found by using the same method as stated in [4].

3.1.2 Community detection in directed graphs

When working with directed graphs instead of undirected graphs, it is also possible to find (directed) communities, as described in [27]. Communities in directed graphs can be useful to for example find the predator-prey relationships in a biological network [12], or to cluster disciplines in citation networks where it might be the case that scientists in a certain expertise field cite to publications in various different expertise fields, but vice versa this is not the case [33]. The authors of [27] describe different ways of finding communities in directed networks, for example by transforming the directed network to an undirected network or to make use of the directedness of edges to find motifs that indicate communities. Moreover, they describe the PageRank-Nibble algorithm for directed graphs, where conductance is defined as follows:

Definition 6 (Directed conductance)**.** *For a graph $G = (V, E)$ with $n = |V|$ and $m = |E|$, the conductance of a subgraph $S \subset G$ is:*

$$\phi(S) = \frac{|\delta^{\pm}(S)|}{\min\left(vol(S), 2m - vol(S)\right)},$$

where $\delta^{\pm}(S) = \{(x, y) \in E | x \in S, y \notin S\}$ and the volume of S is the sum of the degrees of the nodes in subgraph S.

The only change with respect to Definition 5 is that we use a directed edge boundary instead of an undirected one.

3.2 Conductance using HyperBall

We can simplify Definition 5 since in the graphs that we are going to analyse the volume of the subgraph is in general much smaller than the volume of its complement. Therefore, we will use the following definition for conductance in the graphs that we use in the rest of this thesis:

$$\phi\big(S_r(v)\big) = \frac{|\delta\big(S_r(v)\big)|}{\text{vol}\big(S_r(v)\big)}. \tag{3.1}$$

Our goal is to find expressions for the numerator and the denominator of (3.1) in terms of quantities that we can find with the HyperBall algorithm. In order to do so, we transform our (undirected) graph into a directed variant of the graph where every undirected edge becomes two directed edges:

Definition 7 (Out-edgeball)**.** *The out-edgeball with radius r around node v is defined as follows:*

$$\mathcal{E}_r^-(v) = \{(x, y) \in E \,\big|\, x \in B_r(v)\}.$$

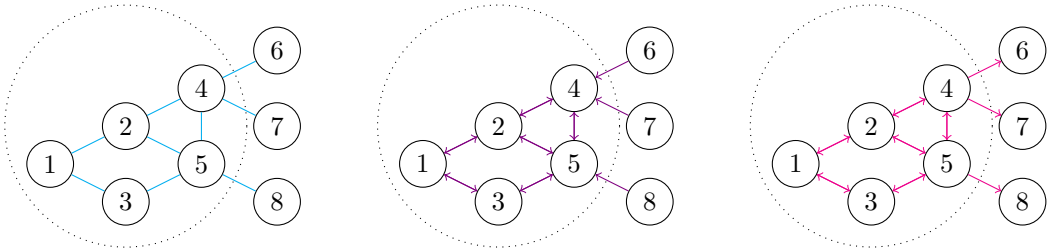
Similarly, we can also define an in-edgeball:

Definition 8 (In-edgeball)**.** *The in-edgeball with radius r around node v is defined as follows:*

$$\mathcal{E}_r^+(v) = \{(x, y) \in E \,\big|\, y \in \mathcal{B}_r(v)\}.$$

The different kinds of edgeballs are illustrated in Figure 3.1. For the in- and out-edgeball it holds that when an edge is a boundary between a node inside and a node outside the ball of radius r, this edge is only counted once. All other edges are counted twice because both nodes are in the nodeball $\mathcal{B}_r(v)$.

In order to find the edge boundary of a ball subgraph, we can compare the number of edges in the directed out-edgeball to the number of edges in the undirected edgeball, since it can be seen in Figure 3.1 that we count every edge in the directed out-edgeball twice, except for the edges in the edge boundary. This leads to two theorems:



(a) The edgeball $\mathcal{E}_2(1)$ **(b)** The in-edgeball $\mathcal{E}_2^+(1)$ **(c)** The out-edgeball $\mathcal{E}_2^-(1)$

Figure 3.1: Three methods for counting edges in an undirected graph.

Theorem 2 (Size of the edge boundary of a ball subgraph)**.** *For an undirected ball subgraph $S_r(v)$ the size of its edge boundary, $|\delta(S_r(v))|$, is given by*

$$|\delta(S_r(v))| = 2|\mathcal{E}_r(v)| - |\mathcal{E}_r^-(v)|,$$

where $|\mathcal{E}_r(v)|$ is the size of the edgeball around node v with radius r and $|\mathcal{E}_r^-(v)|$ is the size of the directed out-edgeball around node v with radius r.

To prove this theorem, we first need an extra definition:

Definition 9 (In-out-edgeball)**.** *The in-out-edgeball with radius r around node v is defined as follows:*

$$\mathcal{E}_r^{\pm}(v) = \{(x,y) \in E \,|\, x \in \mathcal{B}_r(v) \cup y \in \mathcal{B}_r(v)\}.$$

Proof of Theorem 3. We will show that the edge boundary can be written in terms of the undirected edgeball and the directed out-edgeball. The idea behind this proof is that the edge boundary is equal to the difference between the out-edge ball and the in-edgeball, since the only difference between the out-edgeball and the in-edgeball are the edges in the edge boundary (Figure 3.1).

Since we are working with directed edges, we use the definition of the directed edge boundary, as stated in Definition 6 and will show that the cardinality of the directed edge boundary is equal to the cardinality of the undirected edge boundary afterwards.

Every node in the ball subgraph $S_r(v)$ is contained in the nodeball with radius r around node v. Therefore, we can write the edge boundary in terms of this nodeball:

$$\begin{aligned}\delta^{\pm}(S_r(v)) &= \{(x,y) \in E \,|\, x \in V(S_r(v)), y \notin V(S_r(v))\} \\ &= \{(x,y) \in E \,|\, x \in \mathcal{B}_r(v), y \notin \mathcal{B}_r(v)\}. \end{aligned} \tag{3.2}$$

When a node x is in the nodeball $\mathcal{B}_r(v)$, this means that by definition every edge $(x,y) \in E$ is contained in the out-edgeball $\mathcal{E}_r^-(v)$. Similarly, it is true by definition that for a node $y \in B_r(v)$, every edge $(x,y) \in E$ is contained in the in-edgeball $\mathcal{E}_r^+(v)$:

$$\begin{aligned} x \in B_r(v) &\iff (x,y) \in \mathcal{E}_r^-(v), \\ y \in B_r(v) &\iff (x,y) \in \mathcal{E}_r^+(v), \\ y \notin B_r(v) &\iff (x,y) \notin \mathcal{E}_r^+(v), \end{aligned}$$

Now, we can rewrite (3.2) in terms of the in- and out-edgeball:

$$\begin{aligned}\delta^{\pm}(S_r(v)) &= \{(x,y) \in E \,|\, (x,y) \in \mathcal{E}_r^-(v), (x,y) \notin \mathcal{E}_r^+(v)\} \\ &= \{(x,y) \in E \,|\, (x,y) \in \mathcal{E}_r^-(v)\} \backslash \{(x,y) \in E \,|\, (x,y) \in \mathcal{E}_r^+(v)\} \\ &= \mathcal{E}_r^-(v) \backslash \mathcal{E}_r^+(v). \end{aligned} \tag{3.3}$$

By Definition 9, it follows that can be seen that $\mathcal{E}_r^{\pm}(v) = \mathcal{E}_r^{-}(v) \cup \mathcal{E}_r^{+}(v)$, which means that (3.3) can be written as:

$$\begin{aligned}\delta^{\pm}\big(S_r(v)\big) &= \big(\mathcal{E}_r^{-}(v) \cup \mathcal{E}_r^{+}(v)\big) \backslash \mathcal{E}_r^{+}(v) \\ &= \mathcal{E}_r^{\pm}(v) \backslash \mathcal{E}_r^{+}(v).\end{aligned}$$

When we use cardinalities instead of sets, the size of the directed edge boundary is as follows:

$$\begin{aligned}|\delta^{\pm}\big(S_r(v)\big)| &= |\mathcal{E}_r^{\pm}(v) \backslash \mathcal{E}_r^{+}(v)| \\ &= |\mathcal{E}_r^{\pm}(v)| - |\mathcal{E}_r^{+}(v)|,\end{aligned}$$

where the second equality holds because $\mathcal{E}_r^{+}(v) \subseteq \mathcal{E}_r^{\pm}(v)$. The size of the in-out-edgeball $|\mathcal{E}_r^{\pm}(v)|$ is equal to $2|\mathcal{E}_r(v)|$, since in $\mathcal{E}_r^{\pm}(v)$ all edges of the undirected edgeball are counted twice and therefore,

$$|\delta^{\pm}\big(S_r(v)\big)| = |\mathcal{E}_r^{\pm}(v)| - |\mathcal{E}_r^{+}(v)| = 2|\mathcal{E}_r(v)| - |\mathcal{E}_r^{+}(v)|,$$

However, since we have used the directed edge boundary in this proof, we still need to show that the size of the directed edge boundary is equal to the size of the (undirected) edge boundary. By definition, the directed edge boundary of ball subgraph $S_r(v)$ contains every edge between a node inside the nodeball $\mathcal{B}_r(v)$ and a node outside the nodeball $\mathcal{B}_r(v)$ exactly once. Since the undirected edge boundary also contains every edge between a node inside and a node outside the ball subgraph $S_r(v)$ once, the cardinalities of these two kinds of edge boundaries are the same and therefore:

$$|\delta\big(S_r(v)\big)| = |\delta^{\pm}\big(S_r(v)\big)| = 2|\mathcal{E}_r(v)| - |\mathcal{E}_r^{-}(v)|,$$

which concludes the proof. □

The next theorem we introduce makes it possible to express the volume of the ball subgraph $S_r(v)$ in terms of the out-edgeball:

Theorem 3. *For a ball subgraph $S_r(v)$, the volume of $S_r(v)$ is equal to:*

$$\mathit{vol}\big(S_r(v)\big) = |\mathcal{E}_r^{-}(v)|,$$

where $|\mathcal{E}_r^{-}(v)|$ is the size of the directed out-edgeball around node v with radius r.

Proof. This theorem can easily be proven by using Theorem 2. The sum of the degrees of all nodes in $S_r(v)$ is equal to $2|\mathcal{E}_r(v)|$ and therefore we can use the fact that the number of edges contained in the ball subgraph $S_r(v)$ is equal to the number of edges in the edgeball with radius r around node v, minus the edge boundary of $S_r(v)$ (see Figure 3.1):

$$\begin{aligned}\mathrm{vol}\big(S_r(v)\big) &= \sum_{x \in S_r(v)} d(x) \\ &= 2|\mathcal{E}_r(v)| - |\delta\big(S_r(v)\big)| \\ &= 2|\mathcal{E}_r(v)| - (2|\mathcal{E}_r(v)| - |\mathcal{E}_r^{-}(v)|) \\ &= |\mathcal{E}_r^{-}(v)|,\end{aligned}$$

which proves the theorem. □

By using Theorem 2 and 3, the conductance of ball subgraph $S_r(v)$ can now be described in terms of the edgeball and the out-edgeball cardinalities:

$$\phi\big(S_r(v)\big) = \frac{|\delta\big(S_r(v)\big)|}{\mathrm{vol}\big(S_r(v)\big)} = \frac{2|\mathcal{E}_r(v)| - |\mathcal{E}_r^{-}(v)|}{|\mathcal{E}_r^{-}(v)|}. \tag{3.4}$$

Since the cardinalities of the edgeball and the out-edgeball can be found by using the HyperEdgeball algorithm, we can now use these results to find the conductance in a ball subgraph $S_r(v)$. Therefore, we introduce the following adaptation to the HyperEdgeball algorithm:

Algorithm 5 The new initialization for out-edgeballs which uses the HyperEdgeball algorithm to find the cardinality. Note: instead of undirected edges, we now use directed edges.

```
1: c[−], an array of n HyperLogLog counters
2:
3: for each v ∈ V do                                   ▷ Initialisation
4:     for each e ∈ {(w, v) ∈ E} do
5:         ADD(c[v], e)
6:     end for
7: end for
```

When using Algorithms 4 and 5 together, we find $|\hat{\mathcal{E}}_r(v)|$ and $|\hat{\mathcal{E}}_r^-(v)|$ for every node v and radius r, which is enough to estimate conductance in a ball subgraph $S_r(v)$. It can be noticed that all these definitions and theorems also work for directed networks, which makes it possible to find directed communities, as discussed in Section 3.1.2.

3.2.1 Hash function

In the HyperLogLog algorithm, 32-bit hashes for every node and edge are used. Every node is assigned a unique hash. For edges, we hash the concatenation of the two node hashes, similar to the method described in [28]. Since we want the same hash for the same edge, we always write the edge as *(smallest node index, largest node index)* for an undirected graph. For the directed graph, we write the edge as *(out node, in node)*. Finally, we hash the directed and undirected edge hashes again with a 'marker': a hash for the type of edge (i.e. directed or undirected) in order to make the hashes for the directed and undirected edges independent from each other. This means that when hashing the same edge, for example edge (i, j), this will give a different hash for the undirected and the directed variant of that edge:

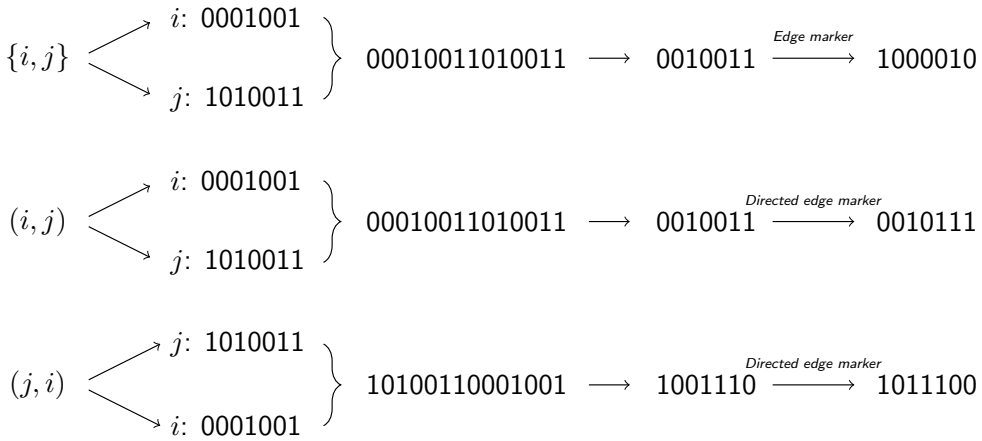


Figure 3.2: Example of hashing the undirected and directed edges between node i and j: first, node i and j are hashed separately. Then, we concatenate the two hashes into a new hash and we hash the concatenation. In order to make the three edges ($\{i, j\}, (i, j)$ and (j, i)) unique, we concatenate the three resulting hashes with a 'marker' and hash this concatenation.

3.2.2 Error bounds

We introduce the estimator $\hat{\phi}\big(S_r(v)\big)$ for conductance of the ball subgraph $S_r(v)$. We can rewrite Equation (3.4):

$$\hat{\phi}\big(S_r(v)\big) = \frac{2|\hat{\mathcal{E}}_r(v)| - |\hat{\mathcal{E}}_r^-(v)|}{|\hat{\mathcal{E}}_r^-(v)|} = 2\frac{|\hat{\mathcal{E}}_r(v)|}{|\hat{\mathcal{E}}_r^-(v)|} - 1, \tag{3.5}$$

where $|\hat{\mathcal{E}}_r(v)|$ and $|\hat{\mathcal{E}}_r^-(v)|$ are respectively the estimators for the edgeball and the directed edgeball around node v with radius r. The expectation and variance of these estimators, as the cardinality of the estimated set goes to infinity, are given in Theorem 1:

$$\begin{aligned}
\mathrm{E}\big[|\hat{\mathcal{E}}_r(v)|\big] &= |\mathcal{E}_r(v)| \cdot \big(1 + \delta_1\big(|\mathcal{E}_r(v)|\big) + o(1)\big) \leq |\mathcal{E}_r(v)| \cdot \big(1 + \delta_1 + o(1)\big), \qquad (3.6)\\
\mathrm{E}\big[|\hat{\mathcal{E}}_r^-(v)|\big] &= |\mathcal{E}_r^-(v)| \cdot \big(1 + \delta_1\big(|\mathcal{E}_r^-(v)|\big) + o(1)\big) \leq |\mathcal{E}_r^-(v)| \cdot \big(1 + \delta_1 + o(1)\big),\\
\mathrm{Var}\big[|\hat{\mathcal{E}}_r(v)|\big] &= |\mathcal{E}_r(v)|^2 \cdot \Big(\frac{\beta_p}{\sqrt{p}} + \delta_2\big(|\mathcal{E}_r(v)|\big) + o(1)\Big) \leq \eta^2 \cdot |\mathcal{E}_r(v)|^2,\\
\mathrm{Var}\big[|\hat{\mathcal{E}}_r^-(v)|\big] &= |\mathcal{E}_r(v)|^2 \cdot \Big(\frac{\beta_p}{\sqrt{p}} + \delta_2\big(|\mathcal{E}_r(v)|\big) + o(1)\Big) \leq \eta^2 \cdot |\mathcal{E}_r(v)|^2. \qquad (3.7)
\end{aligned}$$

The coefficient η is defined as follows:

$$\eta = \frac{\beta_p}{\sqrt{p}} + \delta_2 + o(1). \tag{3.8}$$

The upper bounds are derived by using the fact that $|\delta_1(x)| \leq 5 \cdot 10^{-5} = \delta_1$ for all x and $|\delta_2(x)| \leq 5 \cdot 10^{-4} = \delta_2$ for all x, when the number of registers is larger or equal to 2^4 (Theorem 1). For the rest of this thesis, we assume that the number of registers is always larger than 2^4.

We now introduce Theorem 5 and 7 that give a lower and upper bound for the conductance estimator $\hat{\phi}\big(S_r(v)\big)$ based on Chebyshev's inequality (Theorem 4) and Vysochanskij-Petunin's inequality (Theorem 6) and we show the derivation of these bounds.

Theorem 4 (Chebyshev's inequality)**.** *Assume that the random variable X has variance $\mathrm{Var}(X) = \sigma^2$. Then, for $a > 0$,*

$$P\big(|X - \mathrm{E}(X)| \geq a\big) \leq \frac{\sigma^2}{a^2}.$$

Theorem 5 (Chebyshev bound for the conductance estimator)**.** *The conductance estimator $\hat{\phi}\big(S_r(v)\big)$ of a ball subgraph $S_r(v)$, as defined in (3.5), has the following error bound when the number of edges and directed edges in this subgraph tend to infinity:*

$$P\left[\hat{\phi}\big(S_r(v)\big) \in \left(\frac{1-\epsilon}{1+\gamma} \cdot \phi\big(S_r(v)\big), \frac{1+\epsilon}{1-\gamma} \cdot \phi\big(S_r(v)\big)\right)\right] \geq 1 - \eta^2\left(\frac{|\mathcal{E}_r(v)|^2}{p_1^2} + \frac{|\mathcal{E}_r^-(v)|^2}{p_2^2}\right),$$

with $\epsilon = \frac{p_1}{|\mathcal{E}_r(v)|} + \delta_1 + o(1)$, $\gamma = \frac{p_2}{|\mathcal{E}_r^-(v)|} + \delta_1 + o(1)$ and $p_1, p_2 > 0$, where $\delta_1 = 5 \cdot 10^{-5}$.

Proof. To find the error bound of this estimator $\hat{\phi}\big(S_r(v)\big)$ we use Chebyshev's inequality. The estimator $\hat{\phi}\big(S_r(v)\big)$ consists of two other estimators with known expectation and variance, as described in Equations (3.6) - (3.7). Our goal is to use these estimators to find a lower and upper bound for

$\hat{\phi}\big(S_r(v)\big)$. Following Theorem 4, we get the following inequalities for $p_1, p_2 > 0$ when the number of edges and directed edges in this subgraph tend to infinity:

$$P\Big(\big||\hat{\mathcal{E}}_r(v)| - \mathrm{E}\big[|\hat{\mathcal{E}}_r(v)|\big]\big| \geq p_1\Big) \leq \frac{|\mathcal{E}_r(v)|^2 \cdot \eta^2}{p_1^2}, \tag{3.9}$$

$$P\Big(\big||\hat{\mathcal{E}}_r^-(v)| - \mathrm{E}\big[|\hat{\mathcal{E}}_r^-(v)|\big]\big| \geq p_2\Big) \leq \frac{|\mathcal{E}_r^-(v)|^2 \cdot \eta^2}{p_2^2}, \tag{3.10}$$

with η as defined in (3.8). We can rewrite the left-hand side of (3.9), and the same holds for (3.10):

$$\begin{aligned}
P\Big(\big||\hat{\mathcal{E}}_r(v)| - \mathrm{E}\big[|\hat{\mathcal{E}}_r(v)|\big]\big| \geq p_1\Big) &= P\Big(\big||\hat{\mathcal{E}}_r(v)| - |\mathcal{E}_r(v)|\big(1 + \delta_1\big(|\mathcal{E}_r(v)|\big) + o(1)\big)\big| \geq p_1\Big) \\
&= P\left(\left|\frac{|\hat{\mathcal{E}}_r(v)| - |\mathcal{E}_r(v)|\big(1 + \delta_1\big(|\mathcal{E}_r(v)|\big) + o(1)\big)}{|\mathcal{E}_r(v)|}\right| \geq \frac{p_1}{|\mathcal{E}_r(v)|}\right) \\
&= P\left(\frac{|\hat{\mathcal{E}}_r(v)|}{|\mathcal{E}_r(v)|} \notin \left(1 + \delta_1\big(|\mathcal{E}_r(v)|\big) + o(1) - \frac{p_1}{|\mathcal{E}_r(v)|}, 1 + \delta_1\big(|\mathcal{E}_r(v)|\big) + o(1) + \frac{p_1}{|\mathcal{E}_r(v)|}\right)\right) \\
&\leq P\left(\frac{|\hat{\mathcal{E}}_r(v)|}{|\mathcal{E}_r(v)|} \notin (1-\epsilon, 1+\epsilon)\right) \leq \frac{|\mathcal{E}_r(v)|^2 \cdot \eta^2}{p_1^2},
\end{aligned}$$

with $\epsilon = \frac{p_1}{|\mathcal{E}_r(v)|} + \delta_1 + o(1)$. Similarly, we obtain that for $\gamma = \frac{p_2}{|\mathcal{E}_r^-(v)|} + \delta_1 + o(1)$

$$P\left(\frac{|\hat{\mathcal{E}}_r^-(v)|}{|\mathcal{E}_r^-(v)|} \notin (1-\gamma, 1+\gamma)\right) \leq \frac{|\mathcal{E}_r^-(v)|^2 \cdot \eta^2}{p_2^2},$$

In order to find an error bound for the conductance estimate instead of the two estimators $|\hat{\mathcal{E}}_r(v)|$ and $|\hat{\mathcal{E}}_r^-(v)|$, we define the following two events:

$$\begin{aligned}
A &: \frac{|\hat{\mathcal{E}}_r(v)|}{|\mathcal{E}_r(v)|} \in (1-\epsilon, 1+\epsilon), \\
B &: \frac{|\hat{\mathcal{E}}_r^-(v)|}{|\mathcal{E}_r^-(v)|} \in (1-\gamma, 1+\gamma).
\end{aligned}$$

By using the union bound, we can express the intersection of events A and B as follows:

$$P\big(A \cap B\big) \geq 1 - P\big(\bar{A} \cup \bar{B}\big) \geq 1 - P\big(\bar{A}\big) - P\big(\bar{B}\big) \geq 1 - \frac{|\mathcal{E}_r(v)|^2 \cdot \eta^2}{p_1^2} - \frac{|\mathcal{E}_r^-(v)|^2 \cdot \eta^2}{p_2^2}. \tag{3.11}$$

Now, we rewrite the event $A \cap B$ to obtain probabilistic bounds for $\hat{\phi}\big(S_r(v)\big)$:

$$\begin{aligned}
P\big(A \cap B\big) &\leq P\left[\frac{\frac{|\hat{\mathcal{E}}_r(v)|}{|\mathcal{E}_r(v)|}}{\frac{|\hat{\mathcal{E}}_r^-(v)|}{|\mathcal{E}_r^-(v)|}} \in \left(\frac{1-\epsilon}{1+\gamma}, \frac{1+\epsilon}{1-\gamma}\right)\right] \\
&= P\left[\frac{\frac{|\hat{\mathcal{E}}_r(v)|}{|\hat{\mathcal{E}}_r^-(v)|}}{\frac{|\mathcal{E}_r(v)|}{|\mathcal{E}_r^-(v)|}} \in \left(\frac{1-\epsilon}{1+\gamma}, \frac{1+\epsilon}{1-\gamma}\right)\right] \\
&= P\left[2\frac{|\hat{\mathcal{E}}_r(v)|}{|\hat{\mathcal{E}}_r^-(v)|} - 1 \in \left(\frac{1-\epsilon}{1+\gamma} \cdot \Big(2\frac{|\mathcal{E}_r(v)|}{|\mathcal{E}_r^-(v)|} - 1\Big), \frac{1+\epsilon}{1-\gamma} \cdot \Big(2\frac{|\mathcal{E}_r(v)|}{|\mathcal{E}_r^-(v)|} - 1\Big)\right)\right] \\
&= P\left[\hat{\phi}\big(S_r(v)\big) \in \left(\frac{1-\epsilon}{1+\gamma} \cdot \phi\big(S_r(v)\big), \frac{1+\epsilon}{1-\gamma} \cdot \phi\big(S_r(v)\big)\right)\right].
\end{aligned} \tag{3.12}$$

Combining (3.11) and (3.12) results in the error bounds for the conductance estimator:

$$P\left[\hat{\phi}\big(S_r(v)\big) \in \left(\frac{1-\epsilon}{1+\gamma}\cdot\phi\big(S_r(v)\big), \frac{1+\epsilon}{1-\gamma}\cdot\phi\big(S_r(v)\big)\right)\right] \geq 1-\eta^2\left(\frac{|\mathcal{E}_r(v)|^2}{p_1^2}+\frac{|\mathcal{E}_r^-(v)|^2}{p_2^2}\right).$$

□

When our estimators $|\hat{\mathcal{E}}_r(v)|$ and $|\hat{\mathcal{E}}_r^-(v)|$ follow a unimodal distribution, we can also use Vysochanskij-Petunin's (VP) inequality.

Theorem 6 (Vysochanskij-Petunin's inequality)**.** *Assume that the random variable X has a unimodal distribution with finite mean $\mathrm{E}(X)$ and variance $\mathrm{Var}(X)=\sigma^2$. Then, for $\lambda/\sigma > \sqrt{8/3}$,*

$$P\big(|X-\mathrm{E}(X)| \geq \lambda\big) \leq \frac{4\sigma^2}{9\lambda^2}.$$

When we use this inequality instead of Chebyshev's inequality, we can obtain tighter error bounds. These error bounds are as follows:

Theorem 7 (Vysochanskij-Petunin bound for the conductance estimator)**.** *The conductance estimator $\hat{\phi}\big(S_r(v)\big)$ of a ball subgraph $S_r(v)$, as defined in (3.5), has the following error bound when the number of edges and directed edges in this subgraph tend to infinity and when the distribution of the estimator is unimodal:*

$$P\left[\hat{\phi}\big(S_r(v)\big) \in \left(\frac{1-\epsilon}{1+\gamma}\cdot\phi\big(S_r(v)\big), \frac{1+\epsilon}{1-\gamma}\cdot\phi\big(S_r(v)\big)\right)\right] \geq 1-\frac{4}{9}\eta^2\left(\frac{|\mathcal{E}_r(v)|^2}{\lambda_1^2}+\frac{|\mathcal{E}_r^-(v)|^2}{\lambda_2^2}\right),$$

with $\lambda_1 > \sqrt{8/3\cdot\mathrm{Var}\big(|\mathcal{E}_r(v)|\big)}$, $\lambda_2 > \sqrt{8/3\cdot\mathrm{Var}\big(|\mathcal{E}_r^-(v)|\big)}$, $\epsilon = \frac{\lambda_1}{|\mathcal{E}_r(v)|}+\delta_1+o(1)$ and $\gamma = \frac{\lambda_2}{|\mathcal{E}_r^-(v)|}+\delta_1+o(1)$, where $\delta_1 = 5\cdot 10^{-5}$.

The derivation of this error bound goes in the same way as the proof of Theorem 5, but now we use the Vysochanskij-Petunin inequality. As stated in the theorem, these bounds can only be used when the distribution of the estimator is unimodal. We can check whether a distribution is unimodal by using the DIP test of unimodality [18]. When this test gives a low p score, we can say that the distribution is unimodal.

3.3 Discussion

The advantage of finding conductance using HyperLogLog counters as described in the previous sections is that the conductance of all ball subgraphs in a graph are found at the same time with very low memory usage, which makes it more memory- and time-efficient than other algorithms that try to find minimal conductance sets. However, balls are not always the same as sets and therefore it is possible to find an even lower conductance set using the minimal conductance ball subgraphs as seeds sets in the algorithms described in Section 3.1.1 in order to do a fast and precise preselection of promising nodes that could be in the center of a minimal conductance set, such as PageRank-Nibble or the Multi Walker Chain model.

4 Triangles

Instead of using conductance to find clusters in a graph, we can also look at the number of triangles in a graph to identify how clustered a graph or subgraph is. Related to counting triangles is calculating the transitivity, which in fact is the triangle density of a graph. We introduce another adaptation of the HyperBall algorithm to count triangles and wedges and we find the error bounds of these estimates. Moreover, we show how to find the transitivity in ball subgraphs and give error bounds of this transitivity estimate.

4.1 Community detection

A triangle, also described as a 3-cycle, contains as many edges as a group of 3 nodes can have and is therefore very connected. This also indicates that when a subgraph contains a lot of triangles, this is a very clustered subgraph and could therefore be a good community. The transitivity also tells us something about how connected a graph is [43].

Definition 10 (Transitivity)**.** *The transitivity of a graph G is defined as*

$$t(G) = \frac{3|\Delta(G)|}{|w(G)|},$$

where $|w(G)| = \binom{vol(G)}{2}$ is the number of wedges in G and $|\Delta(G)|$ the number of triangles in this graph.

The authors of [30] empirically found that subgraphs with high transitivity also have a prominent community structure, which makes transitivity a useful measure for finding communities in graphs. Finding the number of wedges in a subgraph is an easy problem, but unfortunately it is not trivial to count triangles in graphs, especially not for large graphs. The *brute-force* method to count triangles in graphs is to try all possible combinations of nodes to find triangles, which costs $O(n^3)$ operations and that is unfeasible for large graphs. We have looked into methods for counting triangles in graphs, which we describe in the following section.

4.1.1 Triangle counting

There are different types of triangle counting algorithms (Figure 4.1), which is summarised in [2]. In this report we focus on random access counting algorithms, which are algorithms that use graphs with available adjacency lists and a known size. Restricted access methods are based on the assumption that not the whole graph is known and that we need to use for example random walks to explore the graph. In this section we discuss the different triangle counting algorithms, where we make the difference between *exact* counting algorithms and *approximate* counting algorithms.

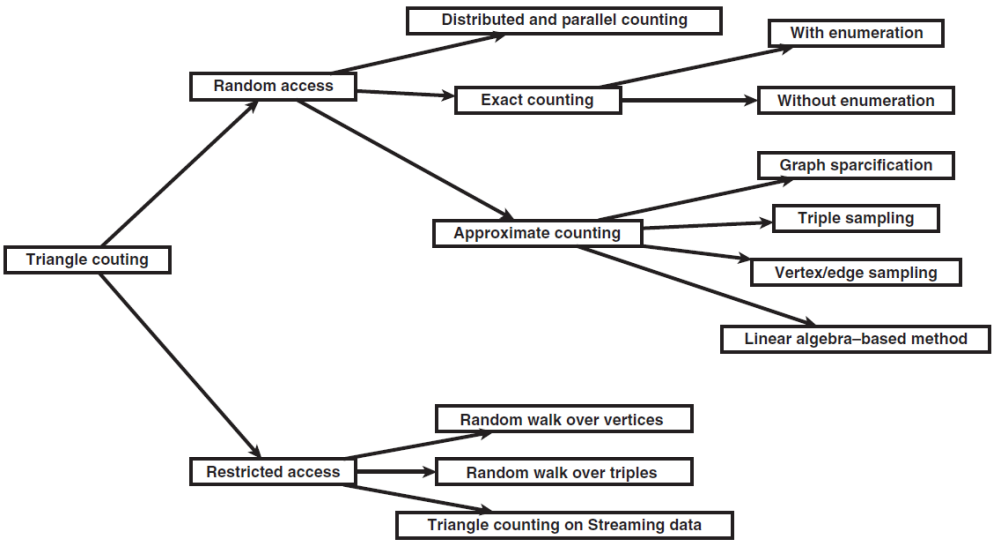


Figure 4.1: Classification of triangle counting works, from [2].

Exact counting

The most naive method of triangle counting is by iterating over a node and its neighbours and then checking whether its neighbours are also neighbours. This *brute-force* method of counting and enumerating triangles has complexity $O(n^3)$. This algorithm also *enumerates* the triangles, i.e. it gives a list of all triangles in a graph, not only the count.

When having access to the whole graph and its adjacency matrix A, it is possible to count triangles by using the fact that

$$\Delta(G) = \frac{1}{6} Tr(A^3), \tag{4.1}$$

where $Tr(A)$ is the trace of matrix A. It can be seen that the method of (4.1) only counts the number of triangles and does not list the triangles. However, since we are dealing with very large graphs and matrix multiplication is still quite costly, the complexity of this method is not much lower than the brute-force method of counting triangles. In [3], the authors give an algorithm for exact triangle counting using a different method for high- and low-degree vertices which has a time complexity of $O(m^{2\omega/(\omega+1)})$, where ω is the exponent of fast matrix multiplication and $m = |E|$, since this algorithm is defined by using matrix multiplication. Following the algorithm of [11], the time complexity of matrix multiplication is $O(m^{\omega})$, where $\omega \leq 2.376$.

An even smaller time complexity can be found when using algorithms like *tree-lister* [20], *edge iterator* [6] or *node iterator* [37]. The first one, *tree-lister*, makes a spanning tree of a graph and then makes use of the fact that every triangle needs at least one edge in the tree. However, constructing a spanning tree of a graph is non-trivial.

Node iterator and *edge iterator* work more or less in the same way. The node variant iterates over all neighbours and neighbour's neighbours in order to find triangles. This algorithm does so in a smart way, i.e. using a sorted list so all triangles are only found once. The edge iterator iterates over edges $\{u, w\} \in E$ and then checks whether node $v \in N$ is in both the neighbourhood of u and w. Both these algorithms have a time complexity of $O(m^{3/2})$. However, in practice most of the times edge iterator is faster [35]. There are also improvements of the node and edge iterator algorithm, for example *ayz*, *listing-ayz* [3], *node iterator-core*, *forward* [37] and *compact-forward* [25]. These

algorithms use the basic idea of the node and edge iterator algorithms, but compress the memory use or divide the graph in smart way in order to improve the performance.

Schank [35] showed that from these algorithms, *compact-forward* is the most time and space efficient in practice for graphs with a skewed degree distribution, while *edge iterator* works best for graphs with degrees that do not differ much from their average degree.

Approximate counting

Since we are dealing with very large graphs and a time complexity of $O(m^{3/2})$ can still be too high, an option is to approximate the triangle count. There are four different kinds of methods that deal with approximate triangle counting, which we will discuss in this section.

Graph sparsification

The first type, graph sparsification-based methods, reduces a large graph by probabilistically removing some edges from the original graph. The idea is to count the triangles in the sparse graph and then extrapolate this count to the large graph. Since the remaining sparse graph is much smaller, it is possible to use an exact counting algorithm for this graph in a reasonable amount of time. DOULION [42], *colorful triangle counting* [31] and an adaptation of DOULION [14] are examples of graph sparsification methods.

In DOULION, the edges in the graph are removed with probability $1 - p$ in order to find a sparse graph. This means that the expected count of the triangles in the original graph is $\hat{\Delta}(G) = \frac{1}{p^3}\Delta(G_s)$, where G_s is the sparse graph.

In *colourful triangle counting*, the idea is to give all vertices a different color and only keep vertices in the sparse graph that are connected to the same color. The probability that a triangle is still there in the sparse graph is then p^2, since if two edges are in the sparse graph, so is the third. This gives an expected triangle count of $\hat{\Delta}(G) = \frac{1}{p^2}\Delta(G)$. This method has a better accuracy than DOULION since it samples more triangles.

The last method in graph sparsification is the adaptation of DOULION, which also samples edges from a graph G with probability p, but besides that checks if the edge of an open triple is also in the original graph. If so, this triple is counted as a triangle. Since this is an extra check in the graph, this will take more time.

Triple sampling

Instead of sampling triangles, what has been done in the graph sparsification-based algorithm, it is also possible to count triples in a graph and from this estimate the number of triangles. This makes use of the fact that the *transitivity* $t(G)$ of a graph can be estimated and therefore when knowing the number of triples, the number of triangles can easily be approximated: $\Delta(G) = \frac{1}{3}t(G) \cdot |\Pi|$, where $|\Pi|$ is the number of triples in a graph. However, uniformly sampling triples in a graph by sampling a node u and then sampling two neighbours v and w of u gives us oversampling of low-degree nodes and under-sampling of high-degree nodes.

The algorithm described in [36] deals with this problem by sampling vertices proportional to the number of triples in this vertex and then uniformly sampling on of those triples which makes the sampling uniform. After the triples are sampled the number of triangles is found by multiplying with the transitivity (Definition 10). This transitivity estimate has been improved by Al Hasan [1] by using independent sampling.

Vertex and edge sampling

Vertex and edge sampling are two approximate triangle counting algorithms based on the exact triangle counting algorithms edge iterator and node iterator. The idea of these methods is to uniformly sample edges or vertices and based on that find the approximate number of triangles. An example of this method is shown in [32], which shows that the accuracy of these types of algorithms is very good for large networks and that edge sampling works better than vertex sampling.

Linear algebra-based methods

The last type of approximate triangle counting methods is based on linear algebra. As shown in equation (4.1), the number of triangles in a graph can be found by using the trace cubed of the adjacency matrix, which is equal to the sum of the eigenvalues cubed. Since the graphs we are dealing with are very large and sparse and usually follow a power-law distribution, this means that the eigenvalues are very skewed which makes it possible to only use the *top-k* eigenvalues. These eigenvalues can be approximated by using the Lanczos method [24]. However, this method does not give guarantees about the accuracy of the estimation.

4.2 Counting triangles using HyperBall

Instead of counting nodes or edges with the HyperLogLog counters, we can also count the number of distinct triangles in a graph and enumerate them. To do so, we use the same algorithm as described in Algorithm 4 (page 11), but we initialise the counter differently in order to count triangles instead of edges (Algorithm 6). With the HyperLogLog counters we can find an estimate of the size of the number of triangles in a ball of radius r around node v, $|\hat{\Delta}_r(v)|$ (Definition 3). In Algorithm 6, we initialise the HyperLogLog counter by finding every triangle around every node and then we hash it to add this triangle to the counter. We hash a triangle in a similar way as described in Section 3.2.1: we hash every node independently and then hash the concatenation of the three nodes together with a marker to create unique hashes for every triangle.

Instead of the naive method described in Algorithm 6 for the initialisation, we can also count triangles by using one of the exact counting algorithms as described in Section 4.1.1, such as *compact-forward* or *edge iterator* in order to make the triangle counting in the initialisation even faster.

Algorithm 6 Initialisation for the Triangle ball algorithm.

```
1: c[−], an array of n HyperLogLog counters
2:
3: for each v ∈ V do                                          ▷ Initialisation
4:     for each i ∈ N(v) do
5:         for each j ∈ N(v), i ∈ N(j) do
6:             ADD(c[v], (v, i, j))
7:         end for
8:     end for
9:     write ⟨v, c[v]⟩ to disk, which estimates |Δ_0(v)|
10: end for
```

4.2.1 Error bounds

We want to know the expectation and variance of the estimator $\left|\hat{\Delta}_r(v)\right|$, which estimates the number of triangles in a ball around node v with radius r. Since this estimator is equal to the size of the HyperLogLog counter for the number of triangles in Algorithm 6, this means that we know the expectation and variance of this estimator when the number of triangles goes to infinity (Theorem 1):

$$\mathrm{E}\left[\left|\hat{\Delta}_r(v)\right|\right] = \left|\Delta_r(v)\right| \cdot \left(1 + \delta_1\left(\left|\Delta_r(v)\right|\right) + o(1)\right), \tag{4.2}$$

$$\mathrm{Var}\left[\left|\hat{\Delta}_r(v)\right|\right] = \left|\Delta_r(v)\right|^2 \cdot \left(\frac{\beta_p}{\sqrt{p}} + \delta_2\left(\left|\Delta_r(v)\right|\right) + o(1)\right)^2. \tag{4.3}$$

Since the number of triangles in a graph is much lower than the number of edges, we need larger graphs in order to have a reasonable error bound on the triangle count.

We again use the Chebyshev inequality (Theorem 4) in order to find a lower and upper error bound for this triangle estimator:

Theorem 8 (Chebyshev bound for the triangle estimator)**.** *The triangle estimator $\left|\hat{\Delta}_r(v)\right|$ of a ball subgraph $S_r(v)$ has the following error bound when the number of triangles in this subgraph tend to infinity:*

$$P\left(\left|\hat{\Delta}_r(v)\right| \in \left(\mathrm{E}\left(\left|\hat{\Delta}_r(v)\right|\right) - a, \mathrm{E}\left(\left|\hat{\Delta}_r(v)\right|\right) + a\right) \geq 1 - \frac{\eta^2 \left|\Delta_r(v)\right|^2}{a^2},$$

for $a > 0$ and $\eta = \frac{\beta_p}{\sqrt{p}} + \delta_2 + o(1)$, as defined in (3.8).

Proof. The proof of this theorem is straightforward: we can use in Chebyshev's inequality for the triangle estimator, fill in the expectation and variance from Equations (4.2) and (4.3) and then rewrite the equation in order to get the error bounds stated in Theorem 8:

$$P\left(\left|\left|\hat{\Delta}_r(v)\right| - \mathrm{E}\left(\left|\hat{\Delta}_r(v)\right|\right)\right| \geq a\right) \leq \frac{\mathrm{Var}\left(\left|\hat{\Delta}_r(v)\right|\right)}{a^2},$$

$$P\left(\left|\left|\hat{\Delta}_r(v)\right| - \mathrm{E}\left(\left|\hat{\Delta}_r(v)\right|\right)\right| \geq a\right) \leq \frac{\left|\Delta_r(v)\right|^2 \cdot \left(\frac{\beta_p}{\sqrt{p}} + \delta_2\left(\left|\Delta_r(v)\right|\right) + o(1)\right)^2}{a^2},$$

$$P\left(\left|\hat{\Delta}_r(v)\right| \in \left(\mathrm{E}\left(\left|\hat{\Delta}_r(v)\right|\right) - a, \mathrm{E}\left(\left|\hat{\Delta}_r(v)\right|\right) + a\right) \geq 1 - \frac{\left(\frac{\beta_p}{\sqrt{p}} + \delta_2\left(\left|\Delta_r(v)\right|\right) + o(1)\right)^2 \left|\Delta_r(v)\right|^2}{a^2},$$

for $a > 0$. Since $\eta = \frac{\beta_p}{\sqrt{p}} + \delta_2 + o(1) \geq \frac{\beta_p}{\sqrt{p}} + \delta_2\left(\left|\Delta_r(v)\right|\right) + o(1)$, as defined in (3.8), this concludes the proof. □

The Vysochanskij-Petunin inequality (Theorem 6) can also be used in order to get a slightly tighter error bound:

Theorem 9 (Vysochanskij-Petunin bound for the triangle estimator)**.** *The triangle estimator $\left|\hat{\Delta}_r(v)\right|$ of a ball subgraph $S_r(v)$ has the following error bound when the number of triangles in this subgraph tend to infinity and when the distribution of the estimator is unimodal:*

$$P\left(\left|\hat{\Delta}_r(v)\right| \in \left(\mathrm{E}\left(\left|\hat{\Delta}_r(v)\right|\right) - \lambda, \mathrm{E}\left(\left|\hat{\Delta}_r(v)\right|\right) + \lambda\right)\right) \geq 1 - \frac{4\eta^2 \left|\Delta_r(v)\right|^2}{9\lambda^2},$$

for $\lambda > \sqrt{8/3 \cdot \mathrm{Var}\left(\left|\Delta_r(v)\right|\right)}$.

This derivation goes in the same way as the proof of Theorem 8, but now we use the Vysochanskij-Petunin inequality.

4.3 Transitivity using HyperBall

In order to find the transitivity of ball subgraphs, we need to find the number of wedges, $|w(S_r(v))|$, in these ball subgraphs. By using the HyperLogLog algorithm, we can find the number of wedges in $S_r(v)$ in the same way as finding the number of triangles. The algorithm stays the same but we leave out the check whether two neighbours of a node are also each other's neighbours (line 5 in Algorithm 6) since this is not required to have a wedge. There is an easier way to find the number of wedges in a graph, as stated in Definition 10, but to use this we need to know the list of nodes in all ball subgraphs. This is not available using the HyperLogLog algorithm and therefore we need to use the same method that we use for triangle counting. Moreover, the advantage of this method is that we find the number of wedges in a ball around a node without knowing which nodes are in every ball subgraph.

4.3.1 Error bounds

In a similar way as we did in Section 3.2.2, we can find the Chebyshev and Vysochanskij-Petunin error bounds of our estimator for transitivity. The transitivity estimator is

$$\hat{t}(S_r(v)) := \frac{3|\hat{\Delta}_r(v)|}{|\hat{w}(S_r(v))|}. \tag{4.4}$$

The expectation and variance of our estimators $|\hat{\Delta}_r(v)|$ and $|\hat{w}(S_r(v))|$ are given in Theorem 1, which results in the following theorem for the Chebyshev error bound of the transitivity estimator:

Theorem 10 (Chebyshev bound for the transitivity estimator)**.** *The transitivity estimator $\hat{t}(S_r(v))$ of a ball subgraph $S_r(v)$, as defined in (4.4), has the following error bound when the number of wedges and triangles in this subgraph tends to infinity:*

$$P\left[\hat{t}(S_r(v)) \in \left(\frac{1-\epsilon}{1+\gamma}\cdot t(S_r(v)), \frac{1+\epsilon}{1-\gamma}\cdot t(S_r(v))\right)\right] \geq 1-\eta^2\left(\frac{|\Delta(S_r(v))|^2}{p_1^2}+\frac{|w(S_r(v))|^2}{p_2^2}\right),$$

with $\epsilon = \frac{p_1}{|\Delta_r(v)|}+\delta_1+o(1)$, $\gamma = \frac{p_2}{|w(S_r(v))|}+\delta_1+o(1)$ *and* $p_1, p_2 > 0$, *with* $\delta_1 = 5\cdot 10^{-5}$.

When this transitivity coefficient has a unimodal distribution amongst the nodes, we can again use the Vysochanskij-Petunin inequality in order to get tighter error bounds:

Theorem 11 (Vysochanskij-Petunin bound for the transitivity estimator)**.** *The transitivity estimator $\hat{t}(S_r(v))$ of a ball subgraph $S_r(v)$, as defined in (4.4), has the following error bound when the number of wedges and triangles in this subgraph tends to infinity and the distribution of the estimator is unimodal:*

$$P\left[\hat{t}(S_r(v)) \in \left(\frac{1-\epsilon}{1+\gamma}\cdot t(S_r(v)), \frac{1+\epsilon}{1-\gamma}\cdot t(S_r(v))\right)\right] \geq 1-\frac{4}{9}\eta^2\left(\frac{|\Delta(S_r(v))|^2}{\lambda_1^2}+\frac{|w(S_r(v))|^2}{\lambda_2^2}\right),$$

with $\lambda_1 > \sqrt{8/3\cdot \mathrm{Var}(|\Delta_r(v)|)}$, $\lambda_2 > \sqrt{8/3\cdot \mathrm{Var}(|w(S_r(v))|)}$, $\epsilon = \frac{\lambda_1}{|\Delta_r(v)|}+\delta_1+o(1)$ *and* $\gamma = \frac{\lambda_2}{|w(S_r(v))|}+\delta_1+o(1)$, *with* $\delta_1 = 5\cdot 10^{-5}$. .

The derivation of Theorems 10 and 11 goes the same as the derivation of Theorems 5 and 7.

4.4 Discussion

All algorithms that are described in Section 4.1.1 find either a list of all triangles in a graph or the exact/approximate number of triangles in a graph. With the adapted HyperBall algorithm for counting triangles, this idea gets extended to finding triangles in a ball around a node. Moreover, this algorithm is able to locate high densities of triangles in the entire graph. When finding the number of triangles in a ball around a node using the methods in Section 4.1.1, we first need to find these ball-subgraphs and then find the triangles. Finding these ball-subgraphs is not trivial memory- and time-wise. Therefore, the HyperLogLog algorithm is very suitable for this type of triangle counting. However, this makes it also difficult to compare this algorithm to other triangle-counting algorithms.

In fact, the goal of this algorithm is to find the number of triangles in a ball of a certain radius around ever node in a graph, which is a different problem than finding all triangles in a graph. After the initialization phase the proposed algorithm only calculates the union of two counters, which is much more efficient memory- and time-wise than comparing the union of two sets which is needed in most triangle counting algorithms. Since we simultaneously find the number of triangles in all ball subgraphs of the graph, this is an addition to the existing algorithms.

There are already algorithms that try to find the transitivity in graphs using for example wedge sampling. An example of this is the algorithm described in [22], where they make use of the fact that when sampling 23 wedges, the probability that the two vertices in a wedge share another edge is $1/2$. This algorithm finds the transitivity coefficient of a graph in $O(\sqrt{n})$ space, but nothing is said about the time requirements. Moreover, this algorithm does not use ball subgraphs which makes it difficult to compare.

In addition to finding the transitivity, we can also find the location of ball subgraphs with high transitivity in the whole graph while using less memory with the HyperBall algorithm. Since high transitivity subgraphs result in good communities, these ball subgraphs can already be good communities or can be used as seed sets for for example PageRank-Nibble, as described in Section 3.1.1, in order to find even better communities.

5 Results

In this chapter we will show the numerical results of our tests of Algorithms 3, 5 and 6 with several graphs. We have done three types of experiments:

1. Checking the error bounds of our estimators for conductance, triangles, transitivity and cycles using generated LFR graphs: synthetic graphs with ground-truth communities.
2. Finding communities using conductance, triangles and transitivity and comparing these communities to the ground-truth communities in LFR graphs.
3. Finding conductance, the number of triangles and transitivity in large real life networks and using these measures as seed sets for PR-Nibble.

We start with showing and discussing the numerical results done in LFR graphs, where we explain how the LFR graphs are generated and discuss the results of the error bounds of the conductance, number of triangles and transitivity. Then, we discuss how to find communities in these LFR graphs with the found measures and show the results of the PR-Nibble algorithm with different seed sets. Lastly, we show the numerical results of the large real life graphs and compare these results to the synthetic LFR graphs.

5.1 Numerical results for LFR graphs

5.1.1 LFR model

LFR graphs [23] are a type of generated graphs where the communities are already known. For these graphs, we can choose a *mixing parameter* μ which indicates the probability of an edge connecting to a node outside its community. There are also other parameters to indicate the minimum and maximum community size $(|C_{min}|, |C_{max}|)$, the average and maximum degree $(\bar{d}, d_{max})$ and two parameters for the power law distribution of the node degrees (τ_1) and the community size distribution (τ_2). The LFR graph is generated as follows [23]:

1. Generate nodes with a power law degree distribution with exponent τ_1, by using the configuration model [29].
2. For every node, a fraction $1-\mu$ links is reserved for connections to nodes inside its community, and a fraction μ links is reserved for connections to nodes outside its community.
3. The sizes of the communities are found by using another power law distribution with exponent τ_2.
4. Every node and its links are randomly added to a community, until the maximum size of this community is reached.
5. There is some rewiring to ensure that the mixing parameter μ is still valid for all nodes.

For these experiments, we have used the LFR graph generator of NetworkX 2.4 in Python 3.6.9 to make three types of graphs with the parameters as shown in Table 5.1. We have used graphs with 1000 and 5000 nodes because this enabled us to also find the exact number of edges, directed edges, wedges and triangles in these graphs.

Graph	$n = \|V\|$	τ_1	τ_2	$\|C_{min}\|$	$\|C_{max}\|$	$\bar{d}$	d_{max}	μ
LFR-1	1000	2	3	10	50	10	50	0.1 - 0.9
LFR-2	1000	2	3	20	100	20	100	0.1 - 0.9
LFR-3	5000	2	3	10	50	10	50	0.1 - 0.9

Table 5.1: Parameters for the generated LFR-graphs

In Figure 5.1, we have plotted the estimated conductance of a graph with 1000 nodes with different mixing parameters. When changing this mixing parameter μ, it can be seen that also the distribution of the conductance in $S_1(v)$ changes. With $\mu = 0.05$, there are ball subgraphs with a conductance of 0, while with $\mu = 0.9$ the conductance is in general much higher. This makes sense since with $\mu = 0.05$ the communities nearly have no links to the outside, while with $\mu = 0.9$ the communities have more links to other communities than within their own community. However, it can also be seen that for every mixing parameter there are some ball subgraphs with a very high conductance.

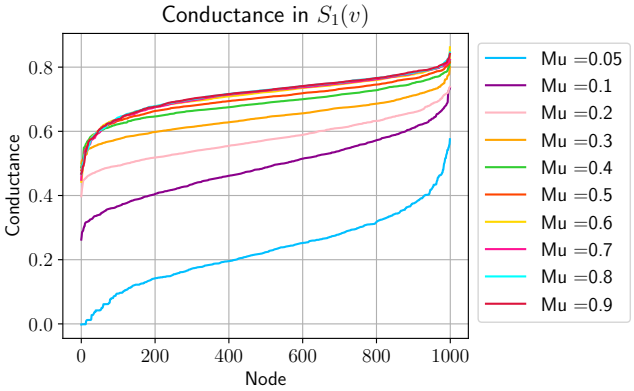


Figure 5.1: Conductance in $S_1(v)$ for different μ, in a LFR-1 graph with $p = 2^{14}$ registers, sorted from smallest to largest conductance.

For the experiments done on LFR graphs, we have used a mixing parameter $\mu = 0.3$, since this was the smallest mixing parameter with no isolated nodes in every generated graph.

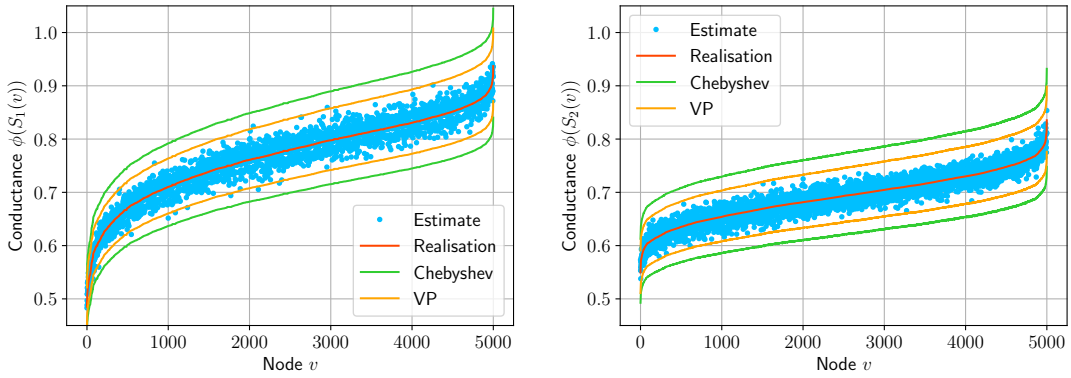
5.1.2 Error bounds

In this section, we show the outcomes of our introduced algorithms for finding the conductance, number of triangles and the transitivity in the above described LFR graphs. For these graphs, we are interested in the conductance in ball subgraphs of radius 1 and 2. A ball larger than this radius consists of a large part of the entire graph and is therefore not interesting anymore to find clusters.

Conductance

In Figure 5.2, the real and estimated conductance in a LFR-3 graph with a mixing parameter $\mu = 0.3$ in two ball subgraphs is plotted. The estimation was made using $p = 2^{14}$ registers. Since the DIP test for unimodality [18] gives a $p-$value smaller than 0.0085, we can reasonably assume that the

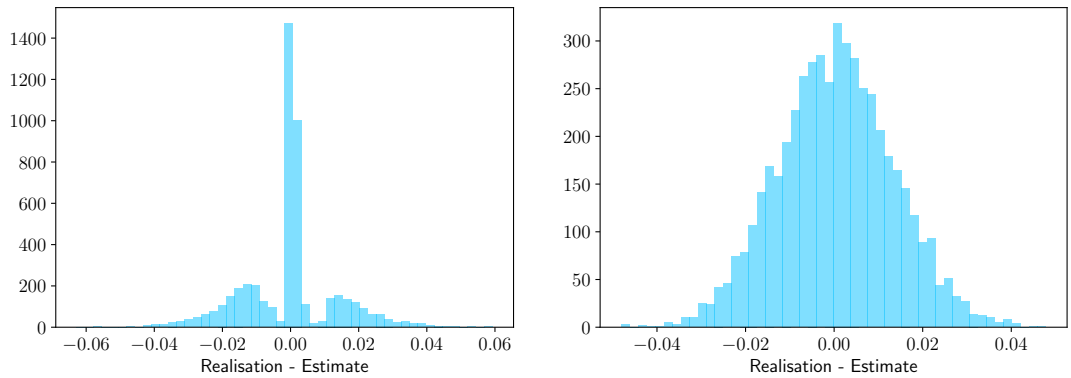
conductance estimator is unimodal and therefore we can use the Vysochanskij-Petunin error bounds (Theorem 7).



(a) Conductance in $S_1(v)$ **(b)** Conductance in $S_2(v)$

Figure 5.2: Estimated and real conductance in the LFR-3 graph with $\mu = 0.3$ and $p = 2^{14}$ registers. The nodes are sorted by realised conductance in ascending order.

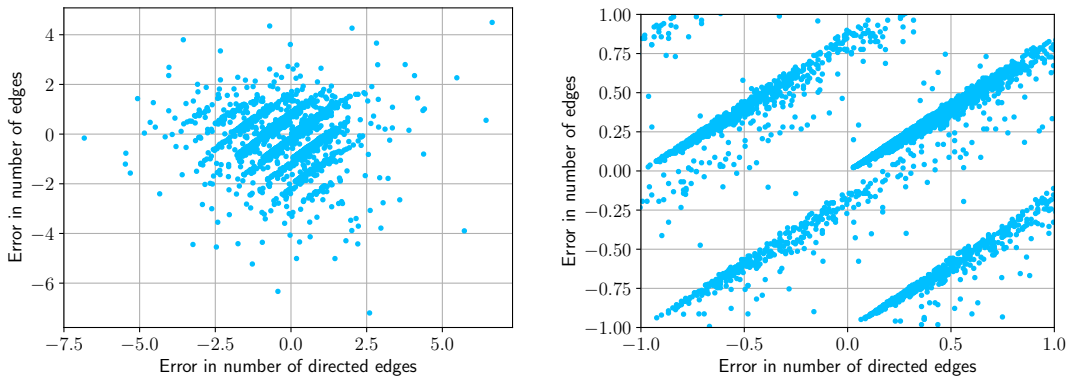
It can be seen that the Vysochanskij-Petunin bounds are tight and represent the 95%-margin of the estimation quite well. The estimation error, which is the difference between the realisation and the estimation, is given in Figure 5.3. It can be seen that the error is centered around 0, which means that our estimator is (almost) unbiased. Moreover, the histogram shows that the error has a small relative variance.



(a) Error in estimated conductance in $S_1(v)$ **(b)** Error in estimated conductance in $S_2(v)$

Figure 5.3: Histogram of the estimation error of the conductance in $S_1(v)$ and $S_2(v)$ in the LFR-3 graph with $\mu = 0.3$ and $p = 2^{14}$ registers.

The difference between these two histograms is very interesting: it can be seen that the histogram in Figure 5.3b, which depicts the estimation error in the ball subgraph $S_2(v)$, has a normal distribution, while the estimation error in the ball subgraph $S_1(v)$, Figure 5.3a, has a different distribution where the mass in a radius of approximately 0.01 around 0 is all moved to 0. Since this only happens in balls of radius 1, balls with a small number of edges and directed edges, this tells us it has something to do with the error in linear counting.



(a) Error in $|\hat{\mathcal{E}}_1(v)|$ versus $|\hat{\mathcal{E}}_1^-(v)|$

(b) Error in $|\hat{\mathcal{E}}_1(v)|$ versus $|\hat{\mathcal{E}}_1^-(v)|$, zoomed in

Figure 5.4: Error in the number of estimated edges and estimated directed edges in $S_1(v)$ in a LFR-3 graph with $\mu = 0.3$, $p = 2^{14}$ registers.

With linear counting, the size E of a counter is estimated by:

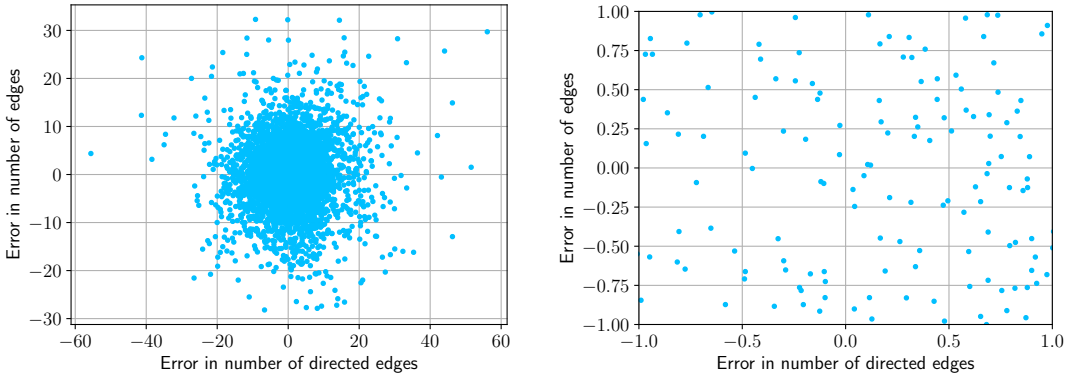
$$E = p \cdot \log(p/V),$$

where V is the number of zeros in the counter and p is the number of registers. The size E therefore only increases in small steps of ± 0.43 when having small cardinalities. This is the case for the estimated cardinality of the edge counter and the directed edge counter, which means that these errors are correlated. This can also be seen in Figure 5.4, where we have plotted the error in the estimated number of edges against the estimated number of directed edges. Since we divide these two estimators in order to get the conductance of a ball subgraph, this correlation is an important factor.

Linear counting for estimating the size is done until a cardinality of $5/2 \cdot p$, which is $5/2 \cdot 2^{14} = 40960$ edges in this case. This means we should also observe this phenomenon in the estimate of the conductance of ball subgraphs with radius 2. This is not the case, however, as can be seen in Figure 5.5. An explanation for this is that the range of the error is much larger since the size of the edge balls varies between 100 edges and 4200 edges instead of between 30 and 500 edges in $S_1(v)$.

In Table 5.2 we show the calculated Chebyshev and Vysochanskij-Petunin error bounds of the conductance for the three LFR graphs, together with the experimental results. For the experimental results, we generated 100 different LFR-1, LFR-2 and LFR-3 graphs and took the mean and variance of the error and the maximum absolute error of the conductance in these graphs in order to show that the HyperBall algorithm works well and is consistent for these graphs. It can be seen that the Vysochanskij-Petunin and Chebyshev error bounds are more or less constant for $p = 2^{14}$ registers over the different LFR graphs and the different radii. Moreover, the mean error is close to zero, which confirms that the conductance estimator is almost unbiased. When increasing the number of nodes, this mean error will only become closer to zero.

We have also looked at different numbers of registers in the LFR-1 graph to show how the precision increases when increasing the number of registers, which is shown in Table 5.3. This table shows that when increasing the number of registers, the Vysochanskij-Petunin and Chebyshev error bounds and the experimental error decreases very fast and the mean error decreases but keeps oscillating around 0, as already expected by Theorem 1.



(a) Error in $|\hat{\mathcal{E}}_2(v)|$ versus $|\hat{\mathcal{E}}_2^-(v)|$

(b) Error in $|\hat{\mathcal{E}}_2(v)|$ versus $|\hat{\mathcal{E}}_2^-(v)|$, zoomed in

Figure 5.5: Error in the number of estimated edges versus estimated directed edges in $S_2(v)$ in a LFR-3 graph with $\mu = 0.3$, $p = 2^{14}$ registers.

Graph		VP	Chebyshev	Mean error	Variance error	Max error
LFR-1	$S_1(v)$	± 0.07560	± 0.1155	$3.693 \cdot 10^{-5}$	$1.904 \cdot 10^{-4}$	0.09165
	$S_2(v)$	± 0.07622	± 0.1164	$-8.806 \cdot 10^{-5}$	$1.673 \cdot 10^{-4}$	0.05536
LFR-2	$S_1(v)$	± 0.07544	0.1153	$-1.838 \cdot 10^{-4}$	$2.056 \cdot 10^{-4}$	0.07314
	$S_2(v)$	± 0.07773	0.1186	$-4.630 \cdot 10^{-4}$	$1.530 \cdot 10^{-4}$	0.05178
LFR-3	$S_1(v)$	± 0.07560	0.1155	$-8.756 \cdot 10^{-6}$	$1.932 \cdot 10^{-4}$	0.1172
	$S_2(v)$	± 0.07588	± 0.1159	$-2.289 \cdot 10^{-5}$	$1.795 \cdot 10^{-4}$	0.07239

Table 5.2: Error bounds (95%) and experimental bounds for conductance in the ball subgraphs $S_1(v)$ and $S_2(v)$ in 100 generated LFR-1, LFR-2 and LFR-3 graphs.

Triangles

For the estimator of the number of triangles, we show error bounds of Theorems 8 and 9. In Figure 5.6, we show the relative size of these Vysochanskij-Petunin and Chebyshev error bounds with an increasing number of registers. It can be seen that for the triangles, the relative error is already less than 5% of the real number of triangles when $p = 2^{12}$ for the Vysochanskij-Petunin bounds and when $p = 2^{13}$ for the Chebyshev bounds.

Since the triangle estimate also has a low $p-$value on the DIP test for unimodality $(p < 0.01)$, we can again use the Vysochanskij-Petunin bounds. In Figure 5.7 we show the number of triangles in the LFR-3 graph in three different ball subgraphs: $S_1(v), S_2(v)$ and $S_3(v)$. There is a large difference in number of triangles between the balls of radius 2 and 3, which can be explained by the fact that ball subgraphs of radius 3 are more or less the entire graph in these graphs, so it makes sense that these subgraphs have a large numbers of triangles. Moreover, it can be seen that when increasing the number of registers, the estimation quickly becomes closer to the realisation (Figure 5.7b).

p	VP	Chebyshev	Mean error	Variance error	Max error
2^8	± 0.7506	± 1.367	$3.858 \cdot 10^{-3}$	0.01485	0.6556
2^{10}	± 0.3219	± 0.5230	$3.987 \cdot 10^{-4}$	$3.236 \cdot 10^{-3}$	0.2529
2^{12}	± 0.1520	± 0.2377	$2.945 \cdot 10^{-4}$	$7.631 \cdot 10^{-4}$	0.1723
2^{14}	± 0.07560	± 0.1155	$-8.756 \cdot 10^{-6}$	$1.932 \cdot 10^{-4}$	0.1172
2^{16}	± 0.03929	± 0.05954	$-6.172 \cdot 10^{-5}$	$4.950 \cdot 10^{-5}$	0.07150
2^{18}	± 0.02158	± 0.03285	$2.175 \cdot 10^{-5}$	$1.183 \cdot 10^{-5}$	0.06403

Table 5.3: Average error bounds and experimental bounds for conductance in the ball subgraph $S_1(v)$ in 100 generated LFR-1 graphs with $\mu = 0.3$ with different numbers of registers.

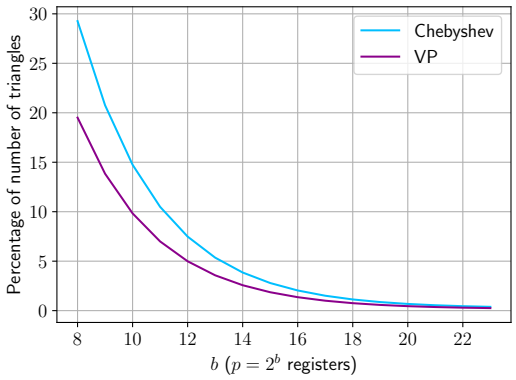


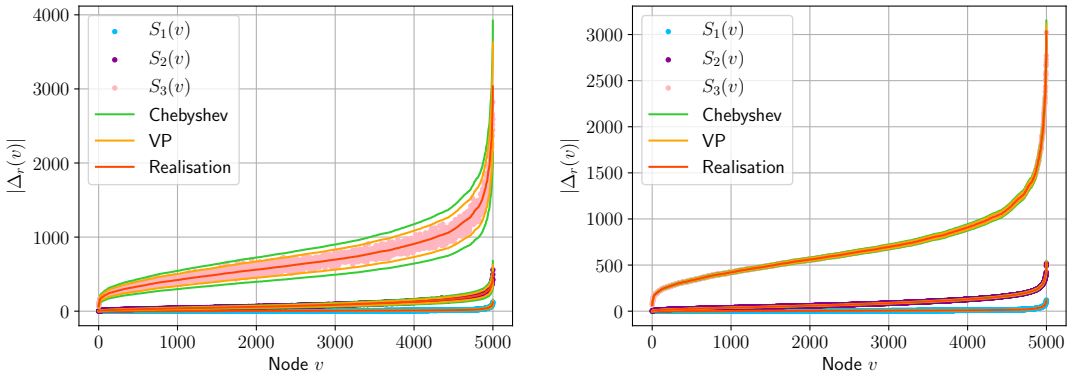
Figure 5.6: Error bounds (95%) for triangle counting as a percentage of the number of triangles with different numbers of registers.

Transitivity

For the transitivity estimate, we can also give the Vysochanskij-Petunin and Chebyshev error bounds, as described in Theorems 10 and 11. In Figure 5.8, we show the transitivity in the ball subgraph $S_1(v)$ (Figure 5.8a) and $S_2(v)$ (Figure 5.8b).

The DIP test for unimodality gives a $p-$value lower than 0.01 so the Vysochanskij-Petunin error bounds can again be used. Interesting to see in these plots is that the transitivity in ball subgraphs of 2 is already very small, which means that there are much more wedges than triangles in these ball subgraphs. This is again an indication that the best communities in these kind of graphs are between the ball subgraphs of radius 1 and the ball subgraphs of radius 2, as also stated in [16].

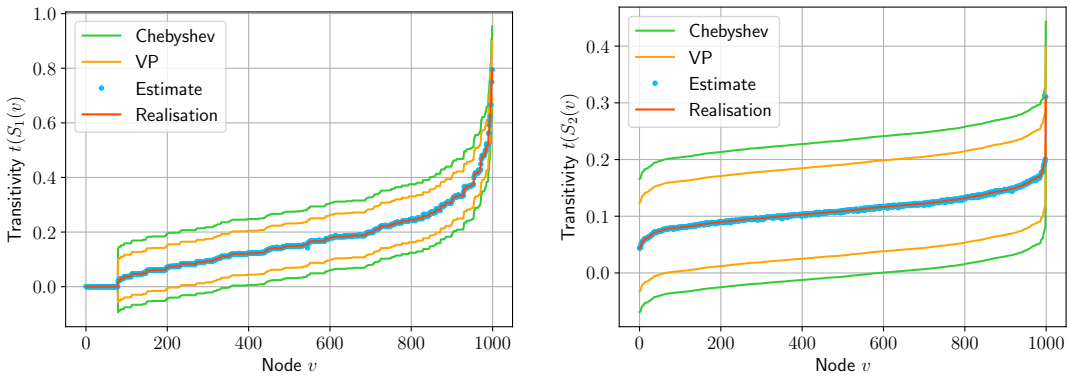
In Figure 5.9 we plotted the number of triangles and the number of wedges over the three ball subgraphs $S_1(v)$, $S_2(v)$ and $S_3(v)$. It can be seen that there is a clear relation between the number of wedges and the number of triangles in all of these ball subgraphs and that it is nearly linear, which also explains the range of the transitivity in $S_2(v)$ is very small (between 0.1 and 0.2). Since the LFR graphs are generated in such a way that links are added with a certain probability, this very strong relation between wedges and triangles is a particular property of these LFR graphs.



(a) Number of triangles in $S_1(v)$, $S_2(v)$ and $S_3(v)$ with $p = 2^8$ registers

(b) Number of triangles in $S_1(v)$, $S_2(v)$ and $S_3(v)$ with $p = 2^{14}$ registers

Figure 5.7: Triangle estimate and realisation in a LFR-3 graph with $\mu = 0.3$. The nodes are sorted by realised number of triangles in ascending order.



(a) Transitivity in $S_1(v)$

(b) Transitivity in $S_2(v)$

Figure 5.8: Transitivity estimate in a LFR-1 graph with $\mu = 0.3$ and $p = 2^{14}$ registers. The nodes are sorted by realised transitivity in ascending order.

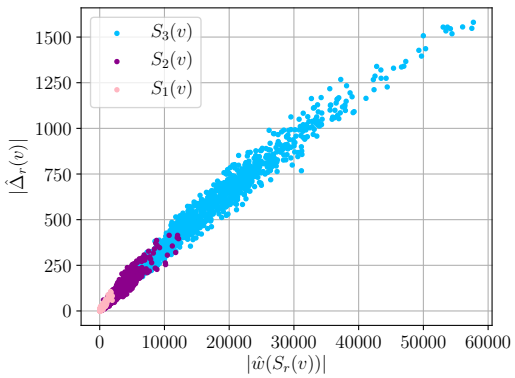
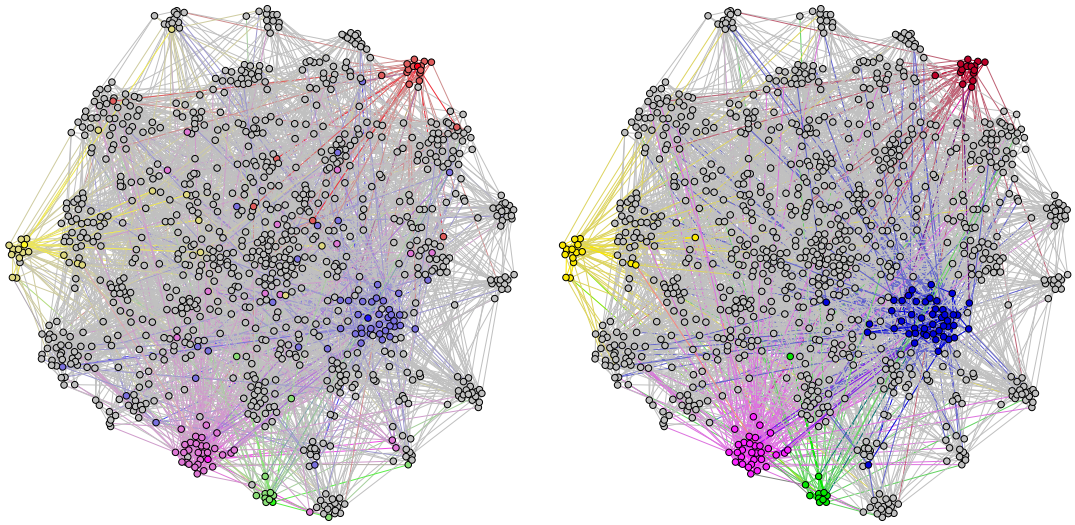


Figure 5.9: Triangles versus wedges in $S_1(v)$, $S_2(v)$ and $S_3(v)$ in LFR-1 with $\mu = 0.3$ and $p = 2^{14}$ registers.

5.1.3 Communities in LFR graphs

When finding the conductance in the ball subgraphs of radius 1, it can be seen that the balls with lowest conductance are already a quite good representation of a community. In Figure 5.10, the ball subgraphs of radius 1 of the 5 nodes with smallest conductance are depicted on the left. These 'communities' are more or less similar to the ground truth communities of the LFR-graph, which are shown on the right. This is in line with what is stated in [16]: in graphs with a large global clustering coefficient and a heavy-tailed degree distribution the ball subgraph of radius 1 is already a good candidate for a minimal conductance set.

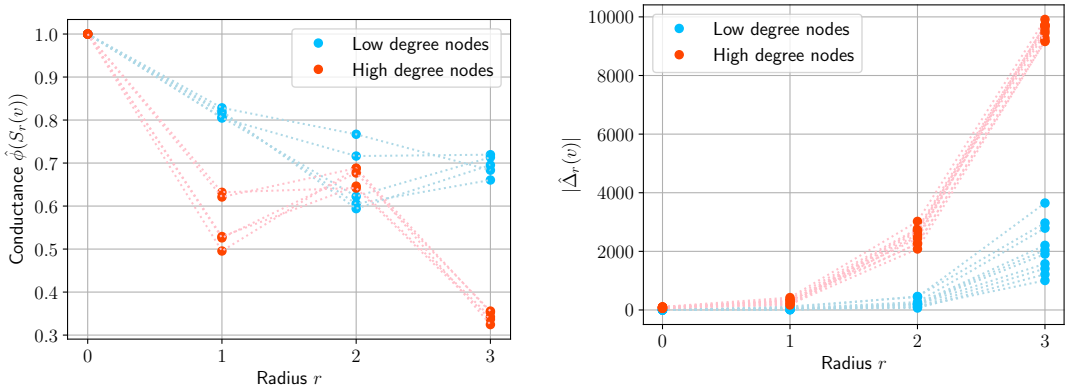


(a) 5 smallest conductance communities in $S_1(v)$ **(b)** Corresponding ground-truth communities

Figure 5.10: 5 communities in a LFR graph with 1000 nodes and $\mu = 0.3$, based on smallest conductance and ground truth.

Moreover there is a relation between the degree of a node and the conductance of the ball around this node in LFR graphs. In Figure 5.11a, we plot the conductance over different radii of the ball subgraph of the five highest and lowest degree nodes. It can be noticed that the high-degree nodes have smaller conductance in the ball subgraph of radius 1, but in the ball subgraph of radius 2 this is not the case. This again confirms the hypothesis that communities can be found in LFR graphs by using ball subgraphs of radius 1 as a seed set, since these subgraphs already have low conductance.

When making a similar plot of the number of triangles in ball subgraphs of different radii, Figure 5.11b, we observe a similar phenomenon: the high-degree nodes have in general more triangles than the low-degree nodes over all radii, which also means that these high-degree nodes have a lower conductance.



(a) Conductance in ball subgraphs of different radii

(b) Number of triangles in ball subgraphs of different radii

Figure 5.11: Conductance and number of triangles of the 5 highest and 5 lowest degree nodes in a LFR-3 graph with $\mu = 0.3$ and $p = 2^{14}$ registers.

Seed sets for PageRank-Nibble

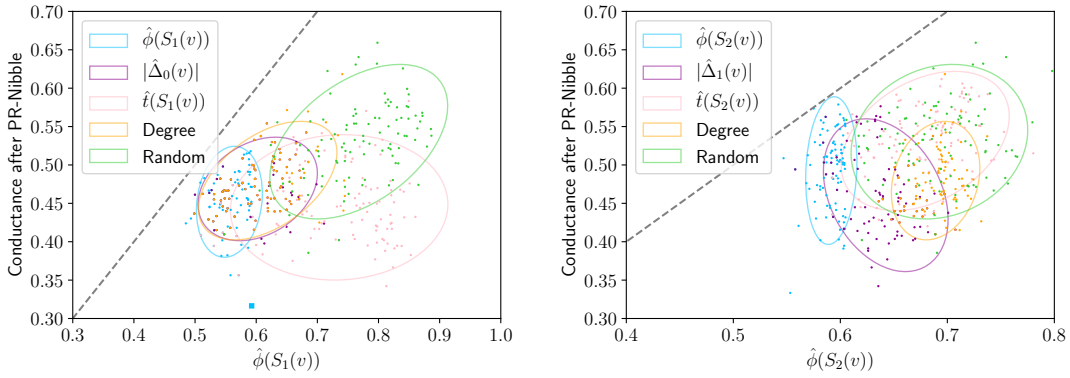
Another possibility to find communities is to use a set of nodes with the smallest conductance in $S_r(v)$ or the largest number of triangles in $S_r(v)$ as a seed set for the PageRank-Nibble or Multi-Walker Chain algorithm, as described in Section 3.1. We have implemented this for a LFR-3 graph with $\mu = 0.3$ and $p = 2^{14}$ registers and we have chosen 5 different seed sets consisting of 100 nodes:

1. Nodes with smallest conductance in $S_1(v)$ and $S_2(v)$,
2. Nodes with largest number of triangles of radius 0 and radius 1, $|\Delta_0(v)|$ and $|\Delta_1(v)|$,
3. Nodes with largest transitivity in $S_1(v)$ and $S_2(v)$,
4. Nodes with highest degree,
5. Randomly chosen nodes.

For the first three seed sets (conductance, number of triangles and transitivity) we have used the ball subgraphs of radius 1 and 2 and compared the conductance found by the PR-Nibble algorithm to respectively the estimated conductance in ball subgraphs of radius 1 and radius 2. For the PR-Nibble algorithm, we have used a maximum cut size of 200, a teleport probability of $\alpha = 0.85$ and we calculate the ϵ-approximate PageRank vector with $\epsilon = 10^{-8}$ [4]. The results are presented in Figure 5.12.

Example In order to understand what is depicted in Figure 5.12, we will give a small example. In Figure 5.12a, one larger blue dot can be found at the bottom of this figure. This blue dot corresponds to node i that is the center of a ball subgraph of radius 1 with low estimated conductance, since the blue seed sets consists of 100 nodes with lowest estimated conductance in their corresponding ball subgraphs of radius 1. The estimated conductance of the ball subgraph around node i is given on the $x-$axis and is equal to 0.5926.

Then, we ran the PR-Nibble algorithm with seed node i, which resulted in a subgraph, not necessarily a ball subgraph, with low conductance. The conductance after PR-Nibble, so the conductance of this new subgraph, is given on the $y-$axis and is equal to 0.3165.



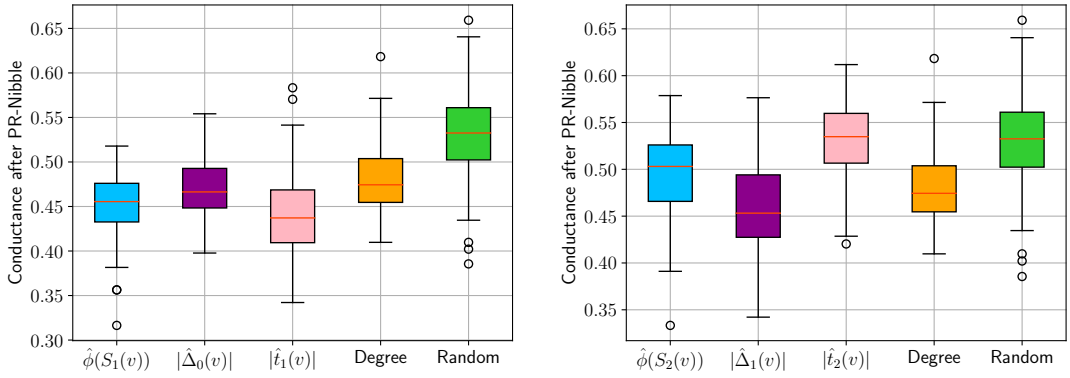
(a) Ball subgraph of radius 1

(b) Ball subgraph of radius 2

Figure 5.12: Conductance in ball subgraphs of radius 1 and 2 versus the set conductance found by using PR-Nibble with the center node of the ball as seed in a LFR-3 graph with $\mu = 0.3$, $p = 2^{18}$ registers. The curves are confidence ellipses with a confidence interval of 3σ.

In Figure 5.12 it can be seen that for every seed that is used, the resulting community has a smaller conductance. This can be seen since every dot is below the diagonal (dashed line), which means that for every dot, the $y-$value is smaller than the $x-$value. Especially in the ball subgraph of radius 1 (Figure 5.12a) the conductance of the ball subgraph is already low and after the PR-Nibble algorithm the resulting communities have the most improved conductance score. Choosing 100 random nodes as a seed set works significantly worse and it can be seen that there is not much difference between choosing the highest degree nodes or the nodes with the largest number of triangles as a seed set. This is again a confirmation of the fact that in these LFR-graphs, high-degree nodes are also the nodes with the most triangles around them (Figure 5.11b). Choosing the nodes with the highest transitivity also works well to find low conductance sets, but the initial conductance of these sets is on average much higher.

When using seed sets based on the ball subgraphs of radius 2 and comparing it to the conductance of these balls, the differences are less pronounced and different than in the ball subgraph of radius 1: as can be seen in the right picture, the nodes with the largest number of triangles now give lowest conductance sets.



(a) Ball subgraph of radius 1

(b) Ball subgraph of radius 2

Figure 5.13: Boxplots of the resulting conductance after PR-Nibble in every seed set

To quantify these statements, we made a boxplot of the resulting conductance after PR-Nibble for every seed set (Figure 5.13) which makes it easier to compare the conductance after running PR-Nibble with the different seed sets.

This confirms that for ball subgraphs of radius 1, the transitivity seed set works best and gives the lowest conductance sets after PR-Nibble. For the ball subgraphs of radius 2, as already said, the triangle seed set works best and the transitivity seed set does not work well at all. It can also be seen that the resulting conductance in Figure 5.13b is in general higher than in Figure 5.13a. This can be explained by the fact that the LFR graphs have relative small communities and therefore measures the ball subgraph of radius 2 do not work well as a seed sets.

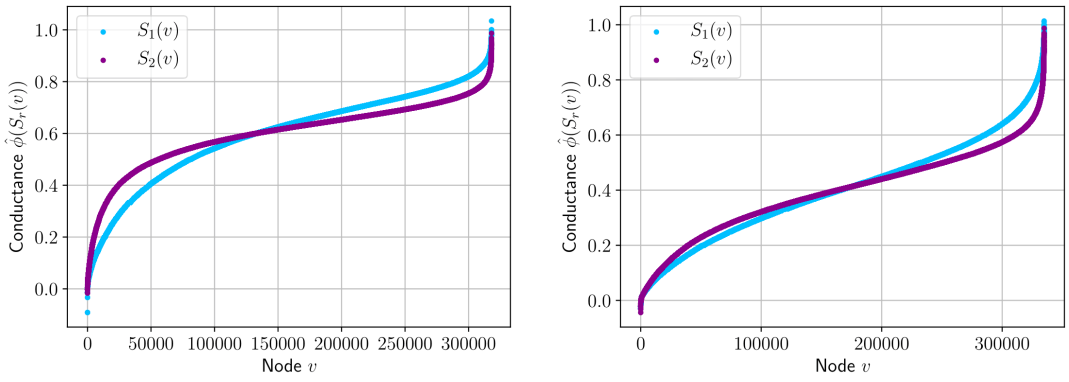
5.2 Numerical results for real life graphs

In order to test our algorithms in real-life graphs, we have used two large networks with a ground truth community structure: COM-DBLP and COM-AMAZON [26, 45]. COM-DBLP is the coauthorship network of the computer science library DBLP, where nodes represent the authors and edges exist if two authors published one or more papers together. The Amazon network was made by crawling the Amazon website through the *users who bought this product also bought...*-section. A node in this network is a product and there is an edge between nodes when two products are frequently bought together. Both these networks have communities consisting of 10 to 200 nodes. In Table 5.4, the properties of these graphs are summarised and more information about these networks is given in [45].

	$n = \|V\|$	$m = \|E\|$	$\|\Delta\|$
COM-DBLP	317k	1M	2M
COM-AMAZON	334k	925k	667k

Table 5.4: Properties of the used real life graphs

For the two graphs, we have used $p = 2^{13}$ registers in all plots that are shown below. We have first plotted the conductance and number of triangles of these three graphs.



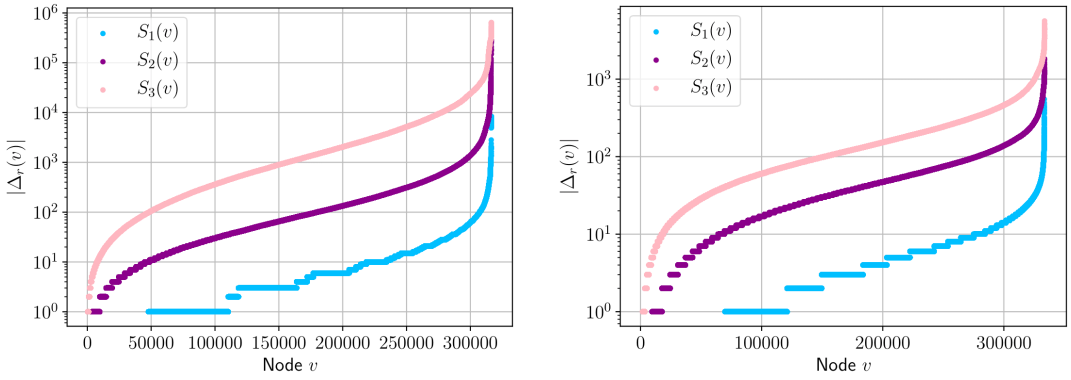
(a) com-DBLP **(b)** com-Amazon

Figure 5.14: Conductance in ball subgraphs $S_1(v)$ and $S_2(v)$ in real life graphs. The nodes are sorted by estimated conductance in ascending order.

There is a slight estimation error in Figures 5.14a and 5.14b, since the conductance can only be between 0 and 1. This is in line with the empirically found maximum error of the conductance estimator, as described in Table 5.2. When comparing Figure 5.14 to the plot given in Figure 5.1,

we notice that the range of the conductance in these real life graphs is similar to the range of a LFR graph with mixing parameter $\mu = 0.05$. This means that there are very well-defined communities with only a few links to the outside of the community in these real life graphs.

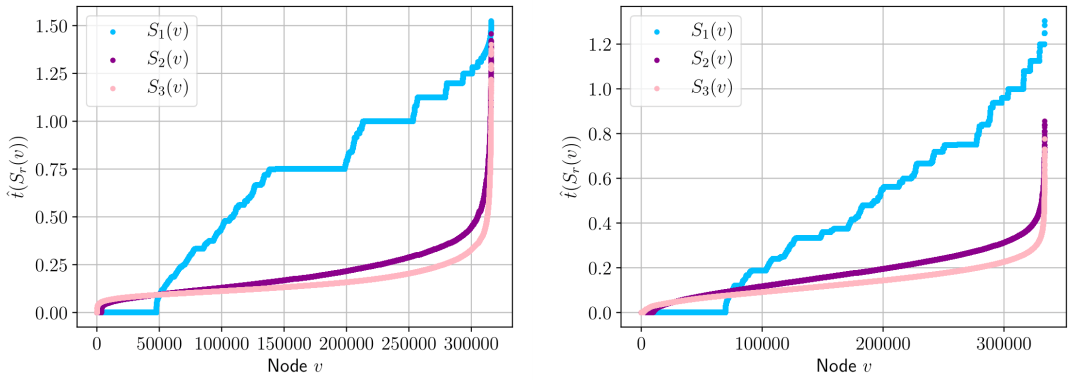
In Figure 5.15 the number of triangles in these three real-life graphs is plotted. There is a large difference between the number of triangles in the ball subgraphs $S_2(v)$ and $S_3(v)$ in all graphs, but there is also a large difference in the number of triangles in the DBLP (5.15a) and in the Amazon graph (5.15b): in the Amazon graph there are less triangles than in the DBLP graph in all ball subgraphs.



(a) com-DBLP **(b)** com-Amazon

Figure 5.15: Number of triangles in ball subgraphs $S_1(v)$, $S_2(v)$ and $S_3(v)$ in real life graphs. The nodes are sorted by estimated number of triangles in ascending order and the y-axis has a log-scale.

While the number of triangles differs much in those real life graphs, the transitivity in both graphs is more or less the same, as depicted in Figure 5.16.



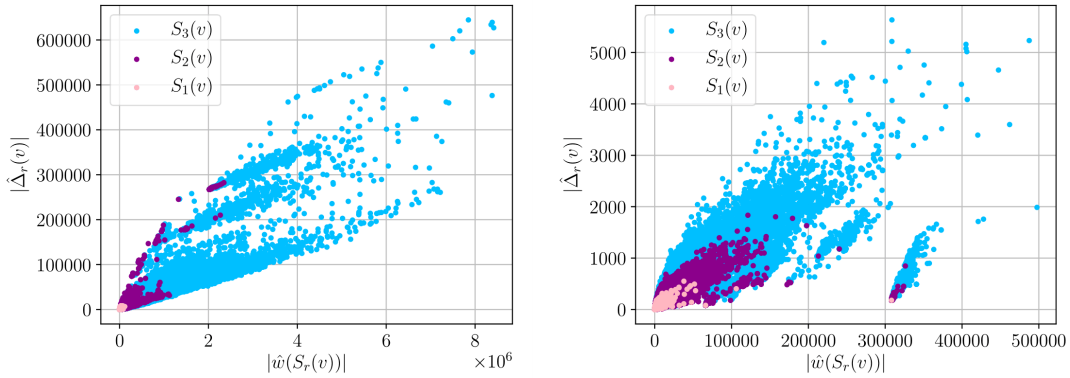
(a) com-DBLP **(b)** com-Amazon

Figure 5.16: Transitivity in ball subgraphs $S_1(v)$, $S_2(v)$ and $S_3(v)$ in real life graphs. The nodes are sorted by estimated transitivity in ascending order.

Just like the transitivity in the LFR-graphs, the transitivity in ball subgraph $S_1(v)$ behaves quite erratically while the transitivity in ball subgraphs with a larger radius is smoother. In the real life graphs, it can be seen that there is again an error in the estimation, since we reach a transitivity of 1.5 instead of a maximum of 1. This error is much larger than the error in conductance, but still

within the error bound. A difference with the LFR graphs with $\mu = 0.3$ is that the transitivity in these real life graphs has a larger range: instead of the transitivity ranging between 0.1 and 0.2 (Figure 5.8b) in a ball subgraph of radius 2, the real life graphs have a transitivity that is ranging from 0 to around 1 in ball subgraphs of radius 2 and 3, which means that even ball subgraphs of radius 2 or 3 are very clustered and therefore could be a community in contrary to the LFR graphs, where most communities are of the size of a ball subgraph of radius 1.

Since the error can become quite big in the transitivity, as can be seen in the above plot, it might again be a good idea to investigate this further by finding a relation between the number of wedges and triangles. However, as can be seen in Figure 5.17, the real life graphs do not behave the same like the LFR-graphs in this aspect. Where we observed a very clear relation between the number of wedges and the number of triangles in LFR graphs, this is not the case in the real life graphs. This makes it clear that when we want to find a clustered group of nodes, looking at the number of triangles around this node is not necessarily the best way and using the transitivity of a subgraph gives different results. Moreover, this does also mean that we cannot sort by degree in order to find clustered groups of nodes: high degree nodes have by definition a high wedge count, but as can be seen in Figure 5.17 this does not necessarily mean high triangle counts.



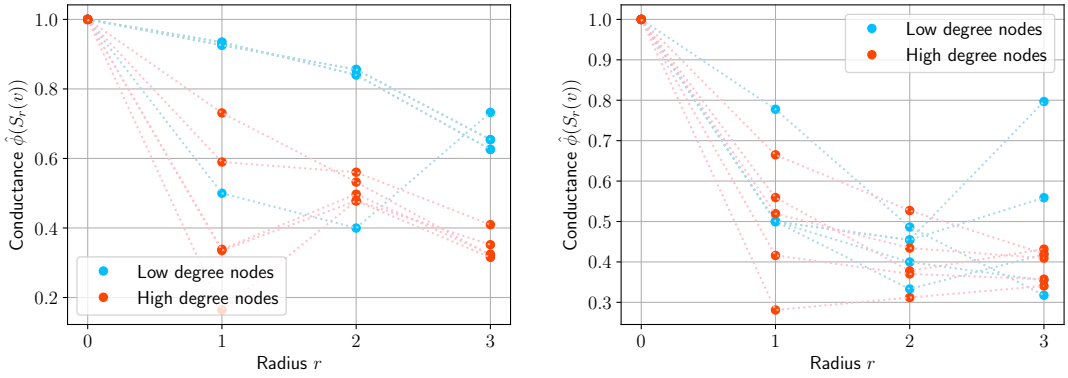
(a) com-DBLP **(b)** com-Amazon

Figure 5.17: Triangles versus wedges in ball subgraphs $S_1(v)$, $S_2(v)$ and $S_3(v)$ in real life graphs.

We have also plotted the conductance and the triangles over different radii of the ball subgraphs with the 5 highest- and 5 lowest degree nodes in Figures 5.18 and 5.19 to check the behaviour of the conductance and the number of triangles over these different radii.

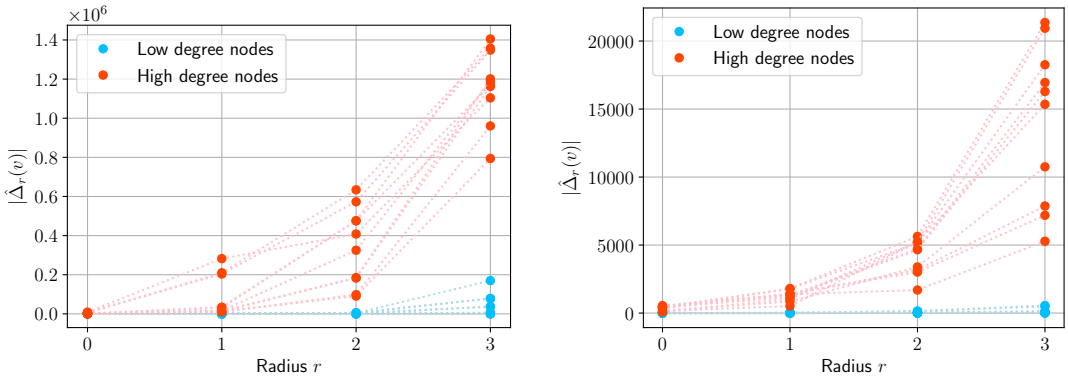
There is no defined split in the high degree and low degree nodes based on the conductance in the real life graphs, as can be seen in Figure 5.18. While in the DBLP graph, Figure 5.18a, we see a clear division starting in the ball subgraph $S_3(v)$, this is less clear in smaller radii. This shows that we cannot just sort by degree in order to find clustered groups of nodes based on conductance and that clustered nodes are not necessarily centered around a node with high degree.

For the triangles, however, as depicted in Figure 5.19, the split is very clear. High-degree nodes have much more triangles over all radii of the ball subgraphs, while low-degree nodes have a much lower number of triangles, even in ball subgraphs of radius 3.



(a) com-DBLP **(b)** com-Amazon

Figure 5.18: Conductance of the 5 highest and 5 lowest degree nodes in real life graphs.



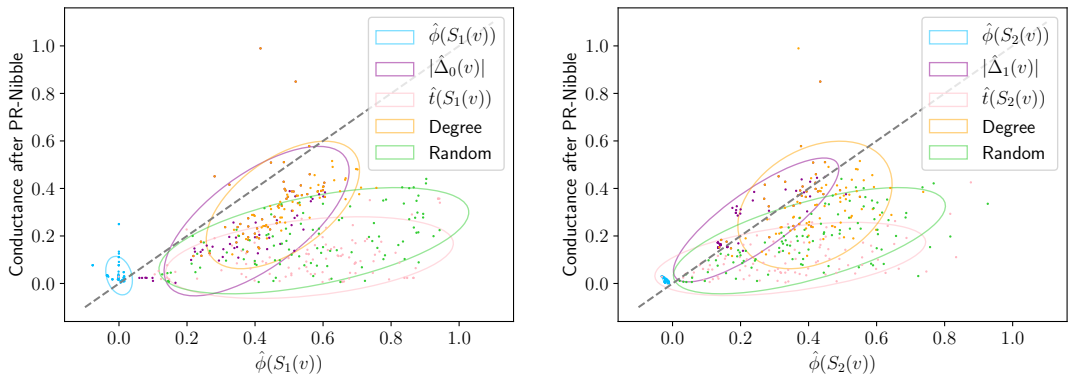
(a) com-DBLP **(b)** com-Amazon

Figure 5.19: Number of triangles of the 5 highest and 5 lowest degree nodes in real life graphs.

Seed sets for PageRank-Nibble

In Figures 5.20 and 5.22 we show the conductance of the different seed sets in $S_1(v)$ and $S_2(v)$ in comparison to the found conductance sets of PageRank-Nibble. The pictures show different results than we have found in LFR-graphs and the most striking difference is that using the 100 nodes with a smallest conductance ball subgraph does not work anymore, since there are a lot of nodes with no links to the the rest of the graph which makes the conductance of these nodes zero. However, since we have used $p = 2^{13}$ registers, these results are less precise than the results with PR-Nibble for LFR graphs.

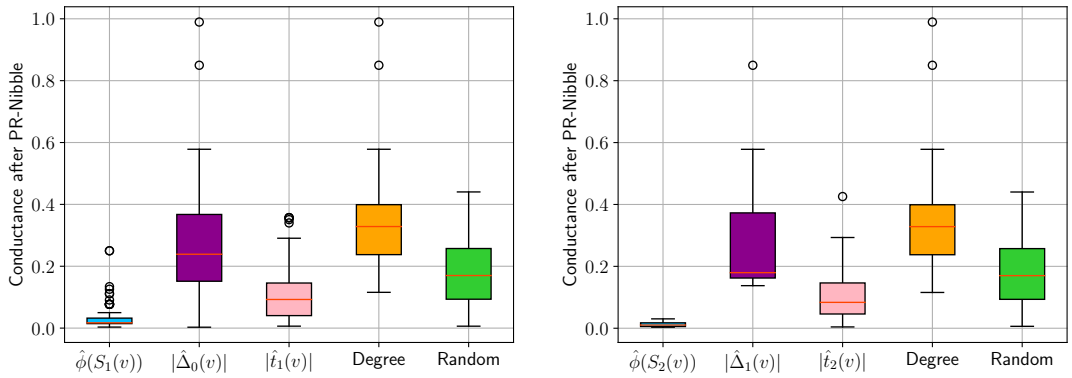
In Figure 5.21, it can be seen that for the Amazon graph, in both ball subgraphs the transitivity seed set gives the lowest resulting conductance after PR-Nibble and also the random seed set gives a low conductance after PR-Nibble.



(a) Ball subgraph of radius 1

(b) Ball subgraph of radius 2

Figure 5.20: Conductance in a ball subgraph of radius 1 and 2 versus the set conductance found by using PR-Nibble with the center node of the ball as seed in com-Amazon, $p = 2^{13}$ registers. The circles are confidence ellipses with a confidence interval of 3σ.

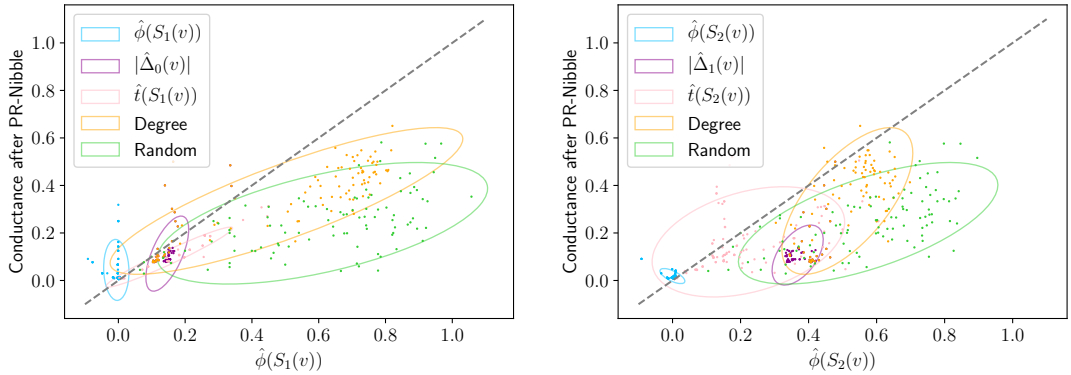


(a) Ball subgraph of radius 1

(b) Ball subgraph of radius 2

Figure 5.21: Boxplots of the resulting conductance after PR-Nibble in every seed set in the Amazon graph

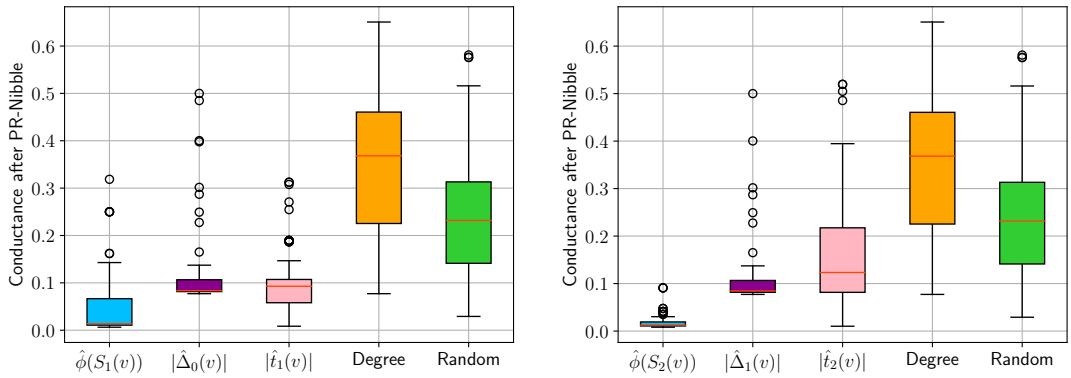
For the DBLP graph, Figure 5.22, the results are quite clear. In the ball subgraph of radius 1 we can see that the number of triangles of the transitivity as a seed set works best, since these ball subgraph already have low conductance and moreover they improve the most after running the PageRank-Nibble algorithm. For the ball subgraph of radius 2, one could say that using triangles as a seed set works best: the triangles have a very small variance and are therefore very clustered and after doing PageRank-Nibble. This is confirmed in Figure 5.23: the transitivity and triangle seed sets give a much lower conductance after PR-Nibble in comparison to the degree and random seed sets.



(a) Ball subgraph of radius 1

(b) Ball subgraph of radius 2

Figure 5.22: Conductance in a ball subgraph of radius 1 and 2 versus the set conductance found by using PR-Nibble with the center node of the ball as seed in com-DBLP, $p = 2^{13}$ registers. The circles are confidence ellipses with a confidence interval of 3σ.



(a) Ball subgraph of radius 1

(b) Ball subgraph of radius 2

Figure 5.23: Boxplots of the resulting conductance after PR-Nibble in every seed set in the DBLP graph

While for the Amazon graph, the transitivity and the random seed sets get the best results, the DBLP graph has the best results by using the number of triangles as a seed set. A reason for this can be that in the DBLP graph there are more triangles, which makes the entire graph more clustered. A difference between the real life graphs and the LFR-graphs is that in this real life graph, seed sets using measures with ball subgraphs of radius 2 also give good results which again indicates that the communities in this graph are larger.

6 Conclusion and Discussion

6.1 Conclusion

The aim of this research was to find clustered groups of nodes using the HyperBall algorithm. We started with a method to find the conductance in ball subgraphs $S_r(v)$ around node v with radius r. We introduced Theorem 2 and 3 which expressed the conductance of a ball subgraph in terms of quantities that can be estimated with an adapted version of the HyperBall algorithm by using directed versions of undirected edges.

We derived two kinds error bounds for the conductance estimate in ball subgraphs: Chebyshev and Vysochanskij-Petunin error bounds, where the latter can only be used when our estimate has a unimodal distribution. These error bounds turned out to be tight, as can be seen in Figure 5.2 and in Table 5.2, and the error in the estimate decreases fast when increasing the precision of the HyperBall algorithm (using more registers) or increasing the number of nodes and edges in a graph. Interestingly, in the error of the conductance in ball subgraphs of radius 1 there is a step-wise error which can be explained by the correlation between the number of directed and undirected edges.

In Chapter 4, we introduced an adaptation of the HyperBall algorithm to count the number of triangles or wedges in ball subgraph $S_r(v)$ and we again derived error bounds of these estimates by using the Chebyshev and VP error bounds. Since the estimated number of triangles and number of wedges is a direct output of the HyperBall algorithm, the error is smaller and also decreases fast when increasing the number of registers (Figure 5.6). However, since counting and enumerating the number of triangles is not a trivial task, it takes longer than the 'normal' HyperBall algorithm.

In Chapter 4, we have also studied the transitivity, which can be seen as the relation between the number of triangles and wedges in a graph. When this transitivity is close to 1, this means that a graph is very clustered and therefore the transitivity is a good measure for finding clustered ball subgraphs. We have found the error bounds for the transitivity in the same way as we did for conductance, depicted in Figure 5.8. Moreover, we found that there is a strong relation between the number of triangles and the number of wedges in LFR-graphs, which shows that it is not necessary to find the transitivity and that the number of triangles or the number of wedges are already a good indication of clustered groups of nodes. However, we found that this is a specific property of the LFR-graphs.

In Chapter 5 we have compared our results in generated LFR-graphs to real life graphs with defined communities. We used a graph of 300 thousand products (nodes) of Amazon product, with a link between two products when customers who bought product a also bought product b, and a graph from the DBLP computer science bibliography that represents authors as nodes and two authors have a link when they have published a paper together. These graphs do not behave the same as the generated LFR graphs: there is no clear relation between the number of triangles and wedges in a graph and there is also no clear split between high- and low-degree nodes in terms of conductance, which makes the transitivity and conductance measures even more important to find clusterings in these graphs. Moreover, in the real life graphs we see that the ball subgraph conductance of all

radii has lower conductance than the LFR-graphs, which means that there are more well-defined communities.

Lastly, we have used the estimates for conductance, transitivity and number of triangles as pre-selection for low-conductance sets that can be found by for example PageRank-Nibble or the Multi-Walker Chain algorithm. We have used seed sets of the 100 nodes with lowest conductance, highest transitivity and largest number of triangles and used these seed sets as starting seed for the PageRank-Nibble algorithm. The conductance of the resulting set that was found by PageRank-Nibble was better than the conductance of the ball subgraph for every seed in the seed set. Pre-selecting on the lowest conductance, highest transitivity or largest number of triangles gave a better result in the PageRank-Nibble algorithm than using a seed set based on degree or choosing 100 nodes at random. In the real life graphs, this result is not always as evident as in the LFR-graphs, but still it could be seen that the triangle and transitivity seed sets gave in general better minimal conductance sets after PR-Nibble. Moreover, when dealing with graphs with isolated nodes or graphs that are disconnected, using lowest conductance balls as seed-sets does not work. These ball subgraphs should be discarded in order to get useful results. In large networks, however, it is no trivial task to identify and discard these ball subgraphs. In general, we recommend using highest transitivity as a seed set for the PR-Nibble algorithm, since this seed set gave the most consistent and best results.

6.2 Discussion

We now discuss some interesting directions for further research. We have shown some results of graphs and realisations that were possible on a 'normal' computer, but to better assess the performance of the HyperBall algorithm, we need to further increase the size of the graphs that we are working with.

In the experiments on the LFR-graphs we came across some interesting results, as depicted in Figure 5.4: there is a correlation between the error in the estimated number of edges and the estimated number of directed edges in our algorithm. We have an intuition of why these strange patterns occur since this only happens when linear counting is used (lines 18 - 20 of Algorithm 2), but it might be interesting to explore this more and to research the linear counting algorithm that is used. Ertl [13] described an adaptation of the HyperBall algorithm where this linear counting is not used anymore, but we did not research this any further. Another option is to use this correlation between the edges and directed edges in order to make the estimations more precise: since the edgeball and the directed out-edgeball have overlap, it might be an idea to the same *hash markers* (Section 3.2.1) which could cancel out bias. The use of hashes has been explored further in [17], but is not included in this research.

The adaptation of the HyperBall algorithm to count triangles is a promising addition to the possibilities of the HyperBall algorithm. However, since the initialisation itself still needs to count the exact number of triangles in order for the HyperBall algorithm to work, this algorithm can only be as fast as the fastest exact counting algorithm for triangles in a graph. Instead of an exact counting algorithm, it might be possible to use an approximation of the number of triangles around a node as initialisation step in Algorithm 6 in order to make it faster, but this also means that it might become less precise.

In Chapter 5, we have chosen to use three kinds of LFR graphs that we could also count in an exact way in a reasonable amount of time. We have chosen a mixing parameter of $\mu = 0.3$ in most of these graphs since this mixing parameter gave graphs with well-defined communities that were also connected. When using smaller mixing parameters, the graphs were not connected anymore which made more it difficult to analyse. However, we are not sure whether $\mu = 0.3$ is the best representation of real life graphs. We also did not look into changing the parameters for the power law distributions (τ_1 and τ_2), which can also change the structure of the generated LFR-graphs to behave more like real life graphs.

For the real life graphs, we have used two graphs with a clear community structure. While both graphs have communities, there are still a lot of differences between those two graphs. This can depend on the size of the communities, the number of links between communities and the average degree of the graph. It would be interesting to explore more of these real graphs to see how these graph properties relate to ball conductance or transitivity, or to use different kinds of graphs such as directed, weighted or temporal graphs, since the HyperBall algorithm should work for these kinds of graphs. For temporal graphs, the method described in this research could help identify properties of temporal graphs very fast, since the introduced algorithm finds the conductance or the number of triangles in all ball subgraphs of the graph. However, for this to work, it is necessary to change the temporal graph into a 'normal' graph in a certain way. In [5] such a method was introduced, but it would be interesting to investigate this further.

We also used our clustering measures (conductance, number of triangles and transitivity) as seed sets for PageRank-Nibble. We chose to use PageRank-Nibble because we wanted to use the idea of starting with a good seed set. Moreover, the Multi Walker Chain model, as described in Section 3.1.1, was only an addition to the PageRank-Nibble so when it works on PageRank-Nibble, our idea was that it also would work for the Multi Walker Chain model. We did not check this for the Multi Walker Chain model, but it might be interesting to see whether the seed sets still work for the Multi Walker Chain model and whether this algorithm finds the low conductance sets even faster.

In the PageRank-Nibble algorithm, we did not investigate how the different parameters changed the resulting minimal conductance set. We have chosen to use a maximum community size of 200, which is quite large, and a teleport probability of 0.85 since this is standard for PageRank-vectors. However, it might be interesting to do further research about how these parameters change the minimal conductance set when using different kinds of seed sets. The outcomes of the PageRank-Nibble algorithm were also difficult to analyse, as can be seen in Figure 5.12. While we can see that some seed sets work better than others, it is difficult to choose the measures on which we want to base our comparison. Moreover, in the real life graphs (Figures 5.20 and 5.22) we see that there are nodes that are not connected to the rest of the graph and therefore have a conductance of zero which makes the lowest ball conductance as a seed set not useful. Therefore, an interesting idea for further research is to remove these disconnected nodes from the graph in order to find clusterings in large connected components.

Bibliography

[1] M. Al Hasan, *Methods and applications of network sampling*, in Optimization Challenges in Complex, Networked and Risky Systems, INFORMS, 2016, pp. 115–139.

[2] M. Al Hasan and V. S. Dave, *Triangle counting in large networks: a review*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 8 (2018), p. e1226.

[3] N. Alon, R. Yuster, and U. Zwick, *Finding and counting given length cycles*, Algorithmica, 17 (1997), pp. 209–223.

[4] R. Andersen, F. Chung, and K. Lang, *Local graph partitioning using pagerank vectors*, in 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), IEEE, 2006, pp. 475–486.

[5] A. Badie-Modiri, M. Karsai, and M. Kivelä, *Efficient limited-time reachability estimation in temporal networks*, Physical Review E, 101 (2020), p. 052303.

[6] V. Batagelj and M. Zaveršnik, *Generalized cores*, arXiv preprint cs/0202039, (2002).

[7] L. Becchetti, C. Castillo, D. Donato, R. Baeza-Yates, and S. Leonardi, *Link analysis for web spam detection*, ACM Transactions on the Web (TWEB), 2 (2008), pp. 1–42.

[8] Y. Bian, J. Ni, W. Cheng, and X. Zhang, *Many heads are better than one: Local community detection by the multi-walker chain*, in 2017 IEEE International Conference on Data Mining (ICDM), IEEE, 2017, pp. 21–30.

[9] P. Boldi, M. Rosa, and S. Vigna, *Hyperanf: Approximating the neighbourhood function of very large graphs on a budget*, in Proceedings of the 20th international conference on World wide web, 2011, pp. 625–634.

[10] P. Boldi and S. Vigna, *In-core computation of geometric centralities with hyperball: A hundred billion nodes and beyond*, in 2013 IEEE 13th International Conference on Data Mining Workshops, IEEE, 2013, pp. 621–628.

[11] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, in Proceedings of the nineteenth annual ACM symposium on Theory of computing, 1987, pp. 1–6.

[12] J. A. Dunne, R. J. Williams, and N. D. Martinez, *Food-web structure and network theory: the role of connectance and size*, Proceedings of the National Academy of Sciences, 99 (2002), pp. 12917–12922.

[13] O. Ertl, *New cardinality estimation algorithms for hyperloglog sketches*, arXiv preprint arXiv:1702.01284, (2017).

[14] R. Etemadi, J. Lu, and Y. H. Tsin, *Efficient estimation of triangles in very large graphs*, in Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, 2016, pp. 1251–1260.

[15] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, *Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm*, in Analysis of Algorithms 2007 (AofA07), 2007, pp. 127–146.

[16] D. F. Gleich and C. Seshadhri, *Vertex neighborhoods, low conductance cuts, and good seeds for local community methods*, in Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, 2012, pp. 597–605.

[17] L. K. Goldberg, *Sketch-based cardinality estimation algorithms*, B.S. thesis, Harvard College, 2018.

[18] J. A. Hartigan, P. M. Hartigan, et al., *The dip test of unimodality*, The annals of Statistics, 13 (1985), pp. 70–84.

[19] J. Huizinga, *Estimating graph properties with hyperloglog-type algorithms*, B.S. thesis, University of Twente, 2019.

[20] A. Itai and M. Rodeh, *Finding a minimum circuit in a graph*, SIAM Journal on Computing, 7 (1978), pp. 413–423.

[21] G. Jeh and J. Widom, *Scaling personalized web search*, in Proceedings of the 12th international conference on World Wide Web, Acm, 2003, pp. 271–279.

[22] M. Jha, C. Seshadhri, and A. Pinar, *A space-efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox*, ACM Transactions on Knowledge Discovery from Data (TKDD), 9 (2015), pp. 1–21.

[23] A. Lancichinetti, S. Fortunato, and F. Radicchi, *Benchmark graphs for testing community detection algorithms*, Physical review E, 78 (2008), p. 046110.

[24] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, United States Governm. Press Office Los Angeles, CA, 1950.

[25] M. Latapy, *Main-memory triangle computations for very large (sparse (power-law)) graphs*, Theoretical computer science, 407 (2008), pp. 458–473.

[26] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford large network dataset collection*. http://snap.stanford.edu/data, June 2014.

[27] F. D. Malliaros and M. Vazirgiannis, *Clustering and community detection in directed networks: A survey*, Physics Reports, 533 (2013), pp. 95–142.

[28] R. C. Merkle, *A digital signature based on a conventional encryption function*, in Conference on the theory and application of cryptographic techniques, Springer, 1987, pp. 369–378.

[29] M. Molloy and B. Reed, *A critical point for random graphs with a given degree sequence*, Random structures & algorithms, 6 (1995), pp. 161–180.

[30] K. Orman, V. Labatut, and H. Cherifi, *An empirical study of the relation between community structure and transitivity*, in Complex Networks, Springer, 2013, pp. 99–110.

[31] R. Pagh and C. E. Tsourakakis, *Colorful triangle counting and a mapreduce implementation*, Information Processing Letters, 112 (2012), pp. 277–281.

[32] M. Rahman and M. Al Hasan, *Approximate triangle counting algorithms on multi-cores*, in 2013 IEEE International Conference on Big Data, IEEE, 2013, pp. 127–133.

[33] M. Rosvall and C. T. Bergstrom, *Maps of random walks on complex networks reveal community structure*, Proceedings of the National Academy of Sciences, 105 (2008), pp. 1118–1123.

[34] S. E. Schaeffer, *Graph clustering*, Computer science review, 1 (2007), pp. 27–64.

[35] T. Schank, *Algorithmic aspects of triangle-based network analysis*, PhD dissertation, Universität Karlsruhe, 2007.

[36] T. Schank and D. Wagner, *Approximating clustering coefficient and transitivity.*, Journal of Graph Algorithms and Applications, 9 (2005), pp. 265–275.

[37] ——, *Finding, counting and listing all triangles in large graphs, an experimental study*, in International workshop on experimental and efficient algorithms, Springer, 2005, pp. 606–609.

[38] D. A. Spielman and S.-H. Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in Proceedings of the STOC, vol. 4, 2004.

[39] M. Stoer and F. Wagner, *A simple min-cut algorithm*, Journal of the ACM (JACM), 44 (1997), pp. 585–591.

[40] N. H. Tran, K. P. Choi, and L. Zhang, *Counting motifs in the human interactome*, Nature communications, 4 (2013), pp. 1–8.

[41] C. E. Tsourakakis, P. Drineas, E. Michelakis, I. Koutis, and C. Faloutsos, *Spectral counting of triangles via element-wise sparsification and triangle-based link recommendation*, Social Network Analysis and Mining, 1 (2011), pp. 75–81.

[42] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, *Doulion: counting triangles in massive graphs with a coin*, in Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, 2009, pp. 837–846.

[43] S. Wasserman, K. Faust, et al., *Social network analysis: Methods and applications*, vol. 8, Cambridge university press, 1994.

[44] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, *A linear-time probabilistic counting algorithm for database applications*, ACM Transactions on Database Systems (TODS), 15 (1990), pp. 208–229.

[45] J. Yang and J. Leskovec, *Defining and evaluating network communities based on ground-truth*, Knowledge and Information Systems, 42 (2015), pp. 181–213.

A | Glossary

$G = (V, E)$	An undirected graph with node set V and edge set E
$n = \|V\|$	Number of nodes in a graph
$m = \|E\|$	Number of edges in a graph
$\mathcal{N}(v)$	All nodes in incident to node v
$\{x, y\} \in E$	Undirected edge
$(x, y) \in E$	Directed edge
$\mathcal{B}_r(v)$	Ball consisting of nodes around node v with radius r
$\mathcal{E}_r(v)$	Ball consisting of undirected edges around node v with radius r
$\Delta_r(v)$	Ball consisting of triangles around node v with radius r
$\mathcal{E}_r^-(v)$	Ball consisting of directed out-edges around node v with radius r
$\mathcal{E}_r^+(v)$	Ball consisting of directed in-edges around node v with radius r
$\mathcal{E}_r^\pm(v)$	Ball consisting of directed in- and out-edges around node v with radius r
$S_r(v)$	Ball subgraph in G induced by the nodes in $\mathcal{B}_r(v)$
$p = 2^b$	Number of registers in a HyperLogLog counter
$\phi(S)$	Conductance of a subgraph S
$\delta(S)$	Edge boundary of a subgraph S
vol(S)	Volume of a subgraph S
$t(S)$	Transitivity of a subgraph S
$w(S)$	Set of wedges in a subgraph S
$C(S)$	Set of cycles in a subgraph S

Table A.1: Glossary of the notation used throughout this thesis