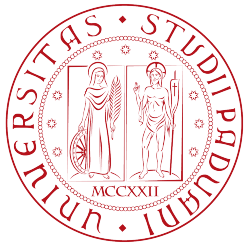

Università degli Studi di Padova
Master Thesis in Computer Engineering
Department of Information Engineering (DEI)
ING-INF/05

Design, Analysis and Engineering of Algorithms for Closeness Centrality

Candidate
Marco Basso

Supervisors
Prof. Andrea Pietracaprina
Prof. Geppino Pucci

Academic Year 2018-2019

Alla mia famiglia

Acknowledgments

I would like to thank my supervisors Prof. Andrea Pietracaprina and Prof. Geppino Pucci of the Department of Information Engineering at University of Padova for their professional guidance through each step of this research and for their patience in the careful review of this thesis. I would also like to express profound gratitude to my family who has always supported me throughout my years of study.

Abstract

The identification of central vertices in large networks is fundamental for many real world applications. Among all the centrality measures defined to address this task, the closeness centrality is one of the oldest and most popular. Its exact computation, even if polynomial in time, is not practically feasible for large graphs. Recent works developed sample based estimation techniques that reduce the computational effort in exchange for a controlled loss of the solution accuracy. The aim of this work is to devise and analyze novel efficient approaches for the estimation of the closeness centrality of the vertices of a large graph and to compare them with the state-of-the-art techniques. In this study we mainly introduce two novel sample based methods, one deterministic and one progressive randomized. The qualities of the approaches are analyzed from a theoretical point of view and by an extensive experimental analysis, based on road networks and social graphs, that assesses their quality with the relative errors of their estimates. The results show that the deterministic approach achieves good performances on road networks and the progressive randomize approach is superior in every aspect among all the compared methods. In conclusion, our novel approaches might be valuable alternatives for closeness centrality estimation on large graphs.

Sommario

L'identificazione dei vertici centrali nelle reti di grandi dimensioni è fondamentale in diverse applicazioni. Tra tutte le misure di centralità definite per affrontare questo compito, la *closeness centrality* è una delle più popolari. Il suo calcolo esatto, seppur ottenibile in un tempo polinomiale, è infattibile in pratica se applicato su grafi di grandi dimensioni. Nei lavori più recenti sono state sviluppate delle tecniche in grado di stimare tale misura, basandosi sul campionamento dei vertici del grafo, che riducono lo sforzo di calcolo in cambio di una perdita controllata dell'accuratezza della soluzione. L'obbiettivo di questo lavoro verte sul concepimento e sul'analisi di nuovi approcci efficienti per la stima della *closeness centrality* dei vertici di un grafo di grandi dimensioni e sul confronto con lo stato dell'arte. In questo studio introduciamo principalmente due nuovi metodi, uno deterministico e uno progressivo randomizzato. Le qualità degli approcci sono analizzate da un punto di vista teorico e da un'ampia analisi sperimentale, basata su reti stradali e grafi sociali, che ne valuta la qualità mediante gli relativi errori delle loro stime. I risultati mostrano che l'approccio deterministico raggiunge buone prestazioni sulle reti stradali e l'approccio progressivo randomizzato è superiore sotto ogni aspetto tra tutti i metodi comparati. In conclusione, i nostri nuovi approcci potrebbero essere valide alternative per la stima della *closeness centrality* su grafi di grandi dimensioni.

Contents

Chapter 1

Introduction

One fundamental task in graph analytics concerns the determination of the importance of the vertices of a graph. Over the years several measures, known as *centrality measures*, have been defined, which capture different graph structural properties in order to reveal the most important vertices of a graph. The common concept behind these measures consist in considering as more important the vertices that are more central on the graph. A vertex is considered central when it lies at the center of a star and centrality measures are based on different notion of star. There are a variety of applications where these measures play a crucial role, to give some examples, they are used to determine the influence of individuals in social graphs, or to detect crucial connections in transportation networks (e.g. road networks), or to determine the importance of web pages.

One of the oldest and most popular centrality measure is the *Closeness Centrality*, introduced in [Bav50] and defined, for each vertex of a connected undirected graph, as the inverse of the sum of distances between the vertex and all other vertices in the graph. Its exact computation reduces to the solution of the well known all-pairs shortest-paths problem whose time complexity is polynomial in the size of the input. With the advent of big data, the need to exploit large graphs structural information has increased. For such huge instances the computation of the all-pairs shortest-paths, although requiring polynomial time, becomes impractical introducing the necessity for alternative solutions. The problem has been addressed by reducing the computational effort in exchange for a controlled loss of the solution accuracy. The most relevant recent works on the estimation of the closeness centrality on huge graphs are those presented in [EW04] and [CCK15]. These state-of-the-art approaches are both based on the computation of a sample of vertices, whose closeness centrality is computed exactly, and, by harnessing the single-source shortest-paths computations of each sample vertex, are able to estimate the closeness centrality of every other vertex in the graph.

The focus of our research is to devise and analyze novel efficient approaches for the estimation of the closeness centrality of the vertices of a large graph, and to compare them with the aforementioned state-of-the-art techniques. We developed two novel sample based approaches that progressively refine the estimates of the closeness centrality for all the vertices of a graph by iteratively adding new vertices to the sample. The first is a deterministic approach based on the Farthest-First Traversal algorithm described in [Gon85] and the second is a progressive randomized approach based on [CCK15]. For each of them we introduce different variants whose qualities are analyzed from a theoretical point of view. We conducted an extensive experimental analysis to compare these novel approaches with each other and with the state-of-the-art RAND, [EW04] and [CCK15]. We based our experiments on road networks and social graphs, which feature different topological properties, and we used the relative errors of the closeness centrality estimates to assess the solution quality. The comparison outcome of the closeness centrality estimation showed that the deterministic approach achieves good performances on road networks, the progressive randomized approach is superior in every aspect and we observe good performances of RAND.

The work is structured as follows. Chapter 2 describes the basics required to understand our research. It presents fundamental graph theory and graph analytics notions. Then, it defines some important centrality measures and describes how the closeness centrality can be computed. Lastly, it illustrates some clustering algorithms that inspired our approaches. Chapter 3 describes the state-of-the-art approaches for the closeness centrality estimation presented in [EW04] and [CCK15]. For each approach we presents both the pseudocode and a theoretical analysis of its quality. Chapter 4 presents a selection of the most promising novel approaches we designed. We present them in the same order in which they were conceived and for each one we provide the pseudocode and an analysis on their operations. Chapter 5 describes in detail the extensive experimental analysis and reports a selection of the results. Finally, Chapter 6 summarizes our findings and discuss possible future developements of the work.

Chapter 2

Preliminaries

This chapter introduces the reader to the research topic by describing the basics needed to understand our work. Most of the concepts presented are described with more detail in [CLRS09]. The first section presents the fundamental graph theory notions in order to clarify the notation used throughout the work, furthermore describes in short how graphs could be represented in memory and two of the fundamental search algorithms used in our methods. In the second section we describe some basic tools used in graph analytics, some topological structures of complex networks and the introduction of the dimensionality of a graph used as input parameter in certain applications. The third section illustrates the most relevant centrality measures described in literature. The forth section focuses on the computation of the closeness centrality, core of our research. In the fifth section we introduce the approximation algorithms which are necessary to understand some of the previous works. Finally in the sixth section we describe some methods used to solve clustering problems which inspired our new algorithm designs.

2.1 Graph Theory Review

In graph theory a *graph* is a structure consisting of a set of objects and a set of pairwise relations among them. The objects are called *vertices* and the pairwise relations are called *edges*. Often, the terms nodes, for the objects, and arcs or links, for the pairwise relations, are used. Both vertices and edges could have *attributes* containing useful information related to them. Formally a graph $G = (V, E)$ is an ordered pair where V represents the set of vertices and $E \subseteq V \times V$ represents the set of edges. When a graph has self loops (i.e. an edge composed by a pair

of equal vertices) or multiple edges between the same pair of vertices it is called a *multi-graph*, otherwise, if none of such edges are present, it is called *simple graph*.

A graph is *undirected* (resp. *directed*) if edges are *unordered* (resp. *ordered*) pairs, and is *weighted* if each edge has a numerical attribute w representing a weight. It is possible to describe a weighted graph with the triplet $G = (V, E, w)$ where $w : E \to \mathbb{R}$ is a function that maps every edge into a real number.

A pair of vertices $v, u \in V$ are said to be *adjacent* if they have an edge $(v, u) \in E$ connecting them. The set of all the adjacent vertices with respect to a generic vertex v corresponds to the *neighborhood* (or *adjacency list*) of v, and its *degree* $\delta(v)$ corresponds to the number of vertices in such neighborhood. In case of a directed graph, it is possible to distinguish between the *in-degree* $\delta^-(v)$ and the *out-degree* $\delta^+(v)$. The former counts the neighbour vertices connected by the edges pointing to v, the latter counts the adjacent vertices whom edges point out from v. An edge $e \in E$ is *incident to a vertex* $v \in V$ if it contains such vertex while an edge e_i is said to be *incident to an edge* e_j if they have one vertex in common.

The edges in a graph define a rich set of possibilities for moving through the vertices. A sequence of vertices $v_1, v_2, \ldots, v_k$ with $(v_i, v_{i+1}) \in E$ is called a *path*. If such sequence does not contain repeated vertices is then called *simple path*. A simple path that starts and ends with the same vertex is called a *cycle*. In case of directed graphs, paths, simple paths and cycles are also directed, that is, the edges inside the sequence have all the same direction.

Two vertices are connected if there exists at least one path containing both of them, while they are unconnected or unreachable if no such path exists. A graph is *connected* when every pair of vertices is connected, otherwise it is *disconnected*. In the latter case the graph could be divided in *connected components* which are maximal connected subgraphs. Thus, every vertex and edge in an unconnected graph belongs to one connected component only.

A fundamental problem related to paths is the *single-source shortest-paths* (SSSP) problem, which consist in finding a shortest-path form a source vertex to all the other reachable vertices in the graph. The shortest-path between two vertices in an unweighted graph is the connecting path with the smallest number of edges while in a weighted graph corresponds to the connecting path where the sum of the edge weights is the lowest.

The *distance* $d(u, v)$ from u to v is the length of the shortest path form u to v, or ∞ if the two vertices are unreachable, that is, they belong to two distinct connected components.

The *Breadth-First Search* (BFS) is one of the simplest algorithms for searching graphs, it is the archetype for many other graph algorithms and it solves the SSSP problem on unweighted graphs. When it comes to weighted graphs, with positive

weights, the solution to the SSSP problem is achieved by the *Dijkstra's algorithm*. These two algorithms are at the basis of our research and are briefly described later on in this section.

2.1.1 Graph Representation

There are two standard ways to represent a graph $G = (V, E)$: as a collection of adjacency lists or as an adjacency matrix. Such representations applies to both directed and undirected graphs. The *adjacency-list* has a more compact structure and is used to represent *sparse* graphs, that is, graphs for which the edges number $|E|$ is much less than the squared vertices number $|V|^2$. The *adjacency-matrix* representation instead is used in case of *dense* graphs, that is, graphs for which $|E|$ is close to $|V|^2$, or when it is necessary to quickly know if there is an edge connecting two given vertices.

The adjacency-list representation of a graph $G = (V, E)$ consist of an array of $|V|$ lists, one for each vertex in the graph. For each vertex $u \in V$, the adjacency list contains all the adjacent vertices to u in G.

The adjacency-list representation requires an amount of memory to store the graph that is $\Theta(|V| + |E|)$, for both directed and undirected graphs.

The adjacency-matrix representation of a graph $G = (V, E)$ consist of a $|V| \times |V|$ matrix $A = (a_{ij})$, where all the vertices are numbered $1, 2, \ldots, |V|$, such that

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \ , \\ 0 & \text{otherwise} \ . \end{cases}$$

The adjacency-matrix representation requires $\Theta(|V|^2)$ memory to store the graph.

We assume that every algorithm described throughout the course of this work considers the adjacency-list representation of its input graph.

2.1.2 Breadth-First Search Algorithm

Given a graph $G = (V, E)$ and a source vertex s, the Breadth-First Search algorithm discovers the vertices reachable from s by systematically exploring the edges of G. It computes the distance (smallest number of edges) form s to each reachable vertex and it also produces the *BFS tree* with root s that contains all reachable vertices. For any vertex v the path from v to s in the BFS tree corresponds to a shortest-path in G from s to v, that is, a path containing the smallest number of edges.

The algorithm, whose pseudocode is reported below as Algorithm 1, expands the frontier between discovered and undiscovered vertices in order to discover all the vertices at distance k form s before discovering any vertices at distance $k + 1$.

This means that the algorithm expands its frontier first in breadth, from which it takes its name.

The algorithm keeps track of progress by coloring each vertex white or black through its execution and by adding discovered vertices to a first-in-first-out queue. All vertices start out white and may later become black. During the search, a vertex is discovered the first time it is encountered, it becomes black and it is added to the queue. The black vertices still in the queue that have at least one adjacent white vertex represent the frontier between discovered and undiscovered vertices.

Algorithm 1 Breadth-First Search algorithm

```
1:  procedure BFS(G, s)
2:      for all vertices u ∈ V − {s} do          ▷ initialize vertex attributes
3:          u.color ← WHITE
4:          u.distance ← ∞
5:          u.parent ← NIL
6:      end for
7:      s.color ← BLACK
8:      s.distance ← 0
9:      s.parent ← NIL
10:     Q ← ∅                                    ▷ Initialize the queue
11:     ENQUEUE(Q, s)                            ▷ add the source vertex to the queue
12:     while Q ≠ ∅ do
13:         u ← DEQUEUE(Q)
14:         for all vertices v adjacent to u do
15:             if v.color = WHITE then
16:                 v.color ← BLACK
17:                 v.distance ← u.distance + 1
18:                 v.parent ← u
19:                 ENQUEUE(Q, v)
20:             end if
21:         end for
22:     end while
23: end procedure
```

The Breadth-First Search builds a BFS tree, initially containing only its root, which is the source vertex s. Every time a white vertex v is discovered by scanning the adjacent vertices of an already discovered vertex u extracted from the queue, the vertex v and the edge (u, v) are added to the tree. Vertex u is the *predecessor* or *parent* of v in the BFS tree. Each vertex $v \neq s$ is discovered once during the

search thus it has exactly one parent.

The algorithms attaches three attributes to each vertex in order to store the current color, the predecessor vertex in the BFS tree and the distance from the source vertex. As described in Algorithm 1 all vertices are white except the source vertex which is black, that is, it has already been discovered and added to the queue. While the queue is not empty the breadth-first search explores the adjacency list of the vertices extracted from the queue and for every unexplored vertex it founds, it changes its color to black and inserts it into the queue. Since the queue is first-in-first-out than the frontier is explored in breadth first.

Given a graph $G = (V, E)$ the time complexity of breadth-first search is $O(|V| + |E|)$ which is linear with respect to the sizes of the graph G and the space complexity is $O(|V| + |E|)$.

2.1.3 Dijkstra's Algorithm

Given a weighted graph $G = (V, E, w)$ where every edge e has a non-negative weight assigned by the function $w : E \to \mathbb{R}$ and a source vertex s the Dijkstra's algorithm solves the SSSP problem for both undirected and directed graphs.

The algorithm maintains an initially empty set S of vertices whose final shortest-path weights form the source s have already been determined and, for each vertex $u \in V$, it keeps its shortest-paths estimate that will be updated each time an adjacent vertex added to S. The algorithm repeatedly selects the vertex $u \in V - S$ which has the minimum shortest-path estimate, adds u to S, and relaxes all edges leaving u. The process of *relaxing* an edge (u, v) consist of testing whether the shortest path to v found so far can be improved by going through u and, if so, update the v distance and parent attributes.

The pseudocode of Dijkstra's algorithm is reported below as Algorithm 2. Its time complexity depends on the implementation of the min-priority queue. Each entity added to the queue represent a vertex u of the graph, its key corresponds to the shortest-path weight from u to the source vertex s estimated so far though the course of the algorithm, and its value is a record that contains other information on the vertex u (such as the index, the parent vertex, and the color). The first possible implementation is an array where each cell corresponds to a vertex numbered from 1 to $|V|$ and contains the distance from the source vertex estimated so far. In this scenario the operations of inserting and decreasing the keys (i.e. update the distance of a vertex) of the priority queue elements takes $O(1)$ time, and each extraction from the queue takes $O(|V|)$ time since is necessary to search through the entire array. Therefore the complexity of Dijkstra's algorithm is $O(|V|^2 + |E|) = O(|V|^2)$.

Assuming that the input graph of the algorithm is sufficiently sparse, that is,

the number of edges is $o(V^2/\log V)$, it is possible to improve the time complexity by implementing the min-priority queue with a binary min-heap structure. The time complexity to create the heap is $O(|V|)$, which is negligible with respect to the overall complexity. The operations of extracting and decreasing the keys inside the heap takes both $O(\log |V|)$ time. Since there are $|V|$ extract operations and at most $|E|$ ($2|E|$ in case of undirected graph) decrease key operations the overall complexity of the Dijkstra algorithm became $O((|V| + |E|) \log |V|)$, which is better than $O(|V|^2)$ when the graph is sparse.

A further improvement could be achieved by implementing the min-priority queue with a Fibonacci heap. This structure could perform a decrease key operation in $O(1)$ amortized time and the extract operation in $O(\log |V|)$ amortized time, which means that the overall complexity of Dijkstra's algorithm became $O(|V| \log |V| + |E|)$.

Algorithm 2 Dijkstra's Single-Source Shortest-Paths Algorithm

```
1:  procedure DIJKSTRA(G, w, s)
2:      for all vertices u ∈ V do
3:          u.distance ← ∞
4:          u.parent ← NIL
5:      end for
6:      s.distance ← 0
7:      S ← ∅
8:      Q ← V
9:      while Q ≠ ∅ do
10:         u ← EXTRACTMIN(Q)
11:         S ← S ∪ {u}
12:         for all vertices v adjacent to u do
13:             if v.distance > u.distance + w(u, v) then
14:                 v.distance ← u.distance + w(u, v)
15:                 v.parent ← u
16:             end if
17:         end for
18:     end while
19: end procedure
```

2.2 Relevant Topological Characteristics

Networks that arise in many real-world applications are conveniently represented as graphs and their analysis is often influenced by variety of distinctive features,

such as, subsets of highly connected vertices, treelike structures, and highly connected hubs.

The most common tools developed for graph analytics consider vertex degrees, distance statistics, and clusters of connected vertices as methods to describe graph structure and topology.

2.2.1 Some Topological Features

Vertex degrees could give insights about the graph structure by combining local information of each vertex. This is done by considering the distribution of the vertex degrees, for instance, when the distribution is concentrated around a value, that is, all the vertices have more or less similar vertex degree, the graph will have similar structures on a local basis that will reflect on a more regular structure overall. A good way to visualize this structural property is by a vertex degree ranked histogram.

Some important graph analytics measures come from the *distance statistics* which summarize structural properties from the distances between pair of vertices in the graph. Given a connected graph $G = (V, E)$ the *eccentricity* of $u \in V$ is the maximum distance among all the possible shortest paths starting form u, the *radius* of G is the minimum eccentricity among its vertices, and the *diameter* of G is the maximum distance among all the possible shortest paths in the graph.

Another distance statistic measure is the *path length* of a vertex, that is, the sample mean of the sum of distances of a vertex. Form this measure it is possible to define the *average path length*, which is the average over all the vertices of their path lengths, that represent in a single value how the vertices are spread in the graph. Small average path length values imply compact graphs where vertices are close to each other on average.

Lastly the *local clustering* and the *global clustering coefficients* are used to measure, respectively, how strongly connected are the vertices within the graph in a local and global point of view. The former calculates the clustering coefficient of each vertex by counting the number of edges between the vertex's neighbors, and then dividing by the maximum possible number of edges between the neighbors. The latter instead counts the number of triangles, which are complete subgraphs with exactly tree vertices, and divides by the number of distinct triples, which are subgraphs with exactly three vertices and two edges.

2.2.2 Types of Topologies

In real world applications graphs are used to represent many different kind of data. Large real world networks, also referred to as complex networks, may have different

local features but in general they share similar global characteristics. Researchers have designed models to represent real world networks by studying their structural components with the tools of graph analytics. From these studies, they defined different graph categories, such as *mesh-like* graphs, *small world* graphs and *scale-free* graphs. In real world applications, the first two types are not commonly found while the last two types arise frequently.

A mesh graph has a configuration of vertices and vertices that forms a regular tiling and it is embedded in some euclidean space $\mathbb{R}^n$. In these kind of graphs, both path length and vertex degree distributions are concentrated, that is, all its vertices have similar vertex degrees and similar sum of distances. Furthermore, the vertices' clustering coefficient are high since close vertices in the euclidean space are connected, that is, vertices neighbours are neighbours too. Road networks are a good example of mesh-like graphs that could be found in real world applications, they embed in a two-dimensional euclidean space.

In a small world graph two randomly chosen vertices are always at a distance that is proportional to the logarithm of the number of vertices in the graph, that is, vertices within the network are all close together (i.e. the small world phenomena). These graphs have high clustering coefficients and low average path lengths. Many real world networks share these properties such as social graphs, some web based graphs like Wikipedia, and gene graphs.

Scale free graphs vertex degrees distributions follow a power law, that is, the fraction of vertices of degree k is proportional to $k^{-\alpha}$ where $\alpha \in (2, 3)$. This means that there are a few vertices, called the hubs, that have a lot of neighbours while the majority of the other vertices have just a few incident edges that likely connect them to at least one of the hubs and a few other vertices. The local clustering coefficient also follows a power law, but this time the higher the vertex degree the lower the clustering coefficient is. Furthermore these graphs have a low average path length because vertices are close together thanks to the hubs. These graphs are very common when it comes to real world use cases and are present in different kind of applications such as internet topology, collaboration graphs, airline graphs, biological graphs, and social graphs.

2.2.3 Dimensionality

An interesting parameter to understand graphs topology is the concept of dimension. There are different definitions for it but in this section we will describe the ones that are more relevant to our purposes.

The first graph dimension that has been defined in literature is the *euclidean dimension*. Given a graph $G = (V, E)$, the euclidean dimension is the least integer n such that there exists a bijective embedding $f : V \to \mathbb{R}^n$ for which $|f(u) - f(v)| =$

1 if and only if $(u, v) \in E$ [EHT65]. For instance a complete graph of size $|V|$ will have a dimension $n = |V| - 1$.

Another parameter used to describe graph structure is the *doubling dimension*. This measure is defined for a general metric space. A *metric space* is an ordered pair (M, d) where M is a set and $d(\cdot)$ is a metric on M, that is, a function $d : M \times M \to \mathbb{R}$ such that for every $x, y, z \in M$ holds that $d(x, y) \geq 0$, $d(x, y) = 0$ if and only if $x = y$, $d(\cdot)$ is symmetric and it satisfies the triangle inequality. Given a weighted undirected graph $G = (V, E, w)$ it is possible to define a distance measure $d(u, v)$ as the shortest path from u to v. This distance satisfies all the properties of a metric, therefore the pair (V, d) is a metric space.

To define the doubling dimension for such a graph it is necessary to define the *ball of radius r around a vertex v* $B(v, r) = \{u \in V : d(u, v) \leq r\}$, that is, the set of vertices that are within a distance r from vertex $v \in V$. The doubling dimension of G is the minimum value D such that, for every $r > 0$ and for every $v \in V$, we have that every ball $B(v, 2r)$ is covered by at most 2^D balls of radius r.

2.3 Centrality Measures

Centrality measures aim at revealing vertex importance in a graph. There are many alternatives for vertex centrality in a graph which could be divided in three main categories: *geometric*, *path-based* and *spectral*. Even though centrality measures are quite different from one another, most of their definitions stem from the natural idea that a vertex at a center of a star is more important than the periphery. We will review a few examples from the various categories. However, since the scope of our research focuses on the closeness centrality, we refer to [BV14] for a more complete overview of the topic.

2.3.1 Geometric Measures

vertices importance for these measures is considered a function of distances, that is, geometric centrality is based on how many vertices are present at every distance from the selected vertex. Furthermore, some of the oldest measures defined in literature belong to this category.

Degree Centrality

Degree centrality is the simplest centrality measure which could be considered geometric because it counts the vertices at distance one, that is, its value is the degree (if the graph is undirected) or the in-degree (if the graph is directed) of a vertex. Degree centrality has some drawbacks that preclude its usage in some applications

(e.g. in a web page scenario it is easy to increase a page importance by adding links to that page and increase its in-degree), but it is a good baseline. Computing the degree centrality measure for every vertex in the graph is polynomial in time, more precisely, it is $\Theta(|E|)$ in case of a adjacency list representation and $\Theta(|V|^2)$ in case of adjacent matrix representation of the graph.

Closeness Centrality

This measure has been proposed in the late forties by Bavelas [Bav50]. Given a graph $G = (V, E)$ the *closeness centrality* of a vertex $u \in V$ is defined by

$$cc_u = \frac{1}{\sum_{v \in V} d(v, u)}. \tag{2.1}$$

The idea behind this measure is that vertices that are more central are closer to all the other vertices in the graph, that is, they have smaller distances from all the other vertices. Therefore, a small sum of distances implies an high importance value.

The closeness centrality is also used in its normalized version defined by

$$\bar{cc}_u = \frac{|V| - 1}{\sum_{v \in V} d(v, u)}. \tag{2.2}$$

This definitions are applicable only for connected graphs (strongly connected in case of undirected graphs) because the distance of unreachable vertices is infinity by definition and brings the closeness centrality value to be zero for every vertex in an unconnected graph.

Computing the closeness centrality reduces to the computation of, for every vertex, the sum of distances to all other vertices, which means solving the all-pairs shortest-paths problem.

The closeness centrality of every vertex could be computed in polynomial time in the size of the input graph.

Harmonic Centrality

The *harmonic centrality* is the most recent measure proposed to adapt the closeness centrality for unconnected graphs. Instead of computing the inverse of the sum of distances it computes the sum of the inverse of the distances. Given a graph $G = (V, E)$ the harmonic centrality for a vertex $u \in V$ is defined by

$$\sum_{v \in V, v \neq u} \frac{1}{d(v, u)} \tag{2.3}$$

When a pair of vertices are unreachable the distance is infinity and the contribution to the sum is zero. This centrality measure has been proven to behave similarly to the closeness centrality on connected graphs (except for some particular graph topologies that are not common in real applications) [Roc09]. The time complexity for its computation is the same as for the closeness centrality.

2.3.2 Path-based Measures

Path-based measures considers all shortest paths (or all paths) coming into a vertex to assign its importance value.

Betweenness Centrality

The idea behind the *betweenness centrality* is to measure how frequently a vertex is within the shortest path connecting two other vertices in a graph. Assuming σ_{vu} to be the number of shortest paths connecting vertex v to vertex u, and $\sigma_{uv}(x)$ to be the number of such paths that pass through vertex x, the betweenness centrality of x is defined by

$$\sum_{u,v \in V,\ u,v \neq x,\ \sigma_{uv} \neq 0} \frac{\sigma_{uv}(x)}{\sigma_{uv}} \tag{2.4}$$

The intuition behind this measure is that if a large fraction of shortest paths passes through x, then x is an important junction point of the graph, thus, it has an higher importance.

In case of a weighted graph, each path is multiplied by its distance.

The computational time to compute the betweenness centrality is polynomial with respect to the size of the input graph.

2.3.3 Spectral Measures

Spectral measures compute the left dominant eigenvector of some matrix derived from the graph. These measures differentiate from each other depending on how the matrix is derived. These are slightly more complex measures but it has been discovered that they could all be expressed as path-based measures (we refer to [BV14] for more details).

Left Dominant Eigenvector Centrality

The *left dominant eigenvector centrality* is the most obvious spectral measure that computes the left dominant egienvector of the adjacency matrix of the graph. The idea behind this measure consist in assigning importance to vertices based on the importance of their adjacent ones.

Given a graph $G = (V, E)$ and $\boldsymbol{A} = (a_{vu})$ its adjacency matrix where every vertex is numbered $1, 2, \cdots, |V|$, the dominant eigenvector centrality score s_u of vertex $u \in V$ is defined by

$$s_u = \frac{1}{\lambda} \sum_{v \in V,\ v \neq u} a_{vu} s_v \tag{2.5}$$

where the sum considers only the adjacent vertices to u for whom $a_{vu} > 0$ and λ is a constant. This equation could be rewritten in matrix form as

$$\boldsymbol{x}\boldsymbol{A} = \lambda \boldsymbol{x} \tag{2.6}$$

where $\boldsymbol{x}$ is the left dominant eigenvector with $|V|$ components and λ is its left dominant eigenvalue. The centrality measure for the i^{th} vertex is represented by the normalized left dominant eigenvector i^{th} component.

The eigenvalue computation could be made with the power iteration algorithm, which is an iterative method that converges to the solution and finds the dominant eigenvector and corresponding eigenvalue.

Page Rank

The *Page Rank* is the most famous centrality measure to compute vertices centrality in a directed graph because has been used to assign vertex importance in a web graph in order to rank web pages in the Google search results.

The idea behind this measure was to model a person who surfs the web, and assign higher importance to the pages which are more likely to be surfed.

Consider a random surfer, starting form a random vertex, that navigates the directed graph $G = (V, E)$. Suppose that at some point he is positioned at vertex $v \in V$. If v has no outgoing edges, then the surfer will move to a random vertex in V. Otherwise, the surfer will move to an adjacent vertex along any of the outgoing edges, with probability $\alpha \cdot 1/n_v$ where n_v represents the number of the outgoing edges form v, or to an arbitrary vertex in V, with probability $1 - \alpha$, for some $\alpha \in (0, 1)$.

The page rank for every vertex in the graph are the elements of the unique vector $\boldsymbol{p}$ satisfying the equation

$$\boldsymbol{p} = \alpha \boldsymbol{p} \|\boldsymbol{A}\|_1 + (1 - \alpha)\boldsymbol{v} \tag{2.7}$$

where α is the *damping factor* and $\boldsymbol{v}$ is a *preference vector* (which must be a distribution, that is, its elements must sum to one). The damping factor represents how likely it is that a random surfer will follow an outgoing edge or jump to a random vertex, while the preference vector assigns to each vertex in the graph a probability to be visited by a surfer jump, that is, the probability to jump on the i^{th} vertex is the value in the i^{th} element of the preference vector.

2.4 Closeness Centrality Computation

Most of the geometric centrality measures could be led back to the solution of the all-pairs shortest-paths problem such as the closeness centrality measure, focus of our research. In fact, the exact computation of the sum of distances $W(u) = \sum_{v \in V} d(u, v)$ for all the vertices $u \in V$, required to compute the closeness centrality, is achieved by computing all the distances among all the vertices pairs within the graph. Therefore it is necessary to find all the shortest paths between all the vertices pairs in the graph which lead us back to the all-pairs shortest-paths solution.

2.4.1 All-Pairs Shortest-Paths

Given a weighted graph $G = (V, E, w)$, where the function $w : E \to \mathbb{R}$ maps the edges to their weights, the all-pairs shortest-paths problem consist in finding for every pair of vertices $u, v \in V$ a shortest (least-weight) path between u and v, where the weight of a path corresponds to the sum of its edges weights.

There are different alternatives to solve the all-pairs shortest-paths problem. The brute force approach consist in running a SSSP algorithm $|V|$ times, using each of the vertices in V as the source. When the edges' weights of the input graph are non-negative it is possible to compute the SSSP using the Dijkstra's algorithm. If the input graph presents negative weights but does not present negative-weighted cycles the Bellman-Ford algorithm should be used instead. The former brute force solution leads to different time complexities based on the implementation of the min-priority queue of the Dijkstra's algorithm. Implementing the queue with an array the overall time complexity is $O(|V|^3 + |V||E|) = O(V^3)$, using a binary heap implementation yields $O(|V||E| \log |V|)$ time complexity and using a Fibonacci heap leads to a running time of $O(|V|^2 \log |V| + |V||E|)$. The latter brute force approach the time complexity is $O(|V|^2|E|)$ because each SSSP computation runs in $O(|V||E|)$ time.

The alternatives to the brute force approaches are the Floyd Warshall and Johnson-Dijkstra algorithms. Both the algorithms solve in polynomial time the all-pairs shortest-paths problem for input graphs that do not present negative-weighted cycles.

The Floyd Warshall algorithm applies on directed weighted graphs $G = (V, E)$ represented by their weighted adjacency-matrix $A = (a_{ij})$ where

$$a_{ij} = \begin{cases} 0 & \text{if } i = j \\ \text{the directed edge weight} w_{ij} & \text{if } i \neq j,\ (i, j) \in E \\ \infty & \text{if } i \neq j,\ (i, j) \notin E \end{cases} \tag{2.8}$$

It relies on the following observation. Assuming that the vertices of G are $V = \{1, 2, \ldots, n\}$, let us consider a subset $\{1, 2, \ldots, h\}$ of vertices for some h. For any pair of vertices $i, j \in V$, consider all the paths from i to j whose intermediate vertices (i.e. the vertices of the path excluding the first and the last in the sequence) are all drawn from $\{1, 2, \ldots, h\}$, and let p_{ij} be the shortest path from among them. The algorithm exploits a relationship between such path p_{ij} and the shortest paths from i to j with all the intermediate vertices in the set $\{1, 2, \ldots, h\}$. The relationship depends on whether or not h is an intermediate vertex of path p_{ij}.

- If h is not an intermediate vertex of path p_{ij}, then all its intermediate vertices are in the set $\{1, 2, \ldots, h-1\}$. Which means that a shortest path form i to j with intermediate vertices drawn from $\{1, 2, \ldots, h-1\}$ is also a shortest path from i to j with intermediate vertices drawn from $\{1, 2, \ldots, h\}$.

- If h is an intermediate vertex of path p_{ij}, we decompose p_{ij} into p_{ih} and p_{hj}. Since p_{ij} is a shortest path, then p_{ih} and p_{hj} are both shortest paths whose intermediate nodes are drawn from $\{1, 2, \ldots, h-1\}$

The algorithm computes the shortest paths by iteratively adding vertices to the initially empty set of intermediate vertices until $h = |V|$. The initial shortest paths $p_{ij}^{(0)}$ contains the a_{ij} values of the adjacency-matrix of the input graph because there are no intermediate vertices. For each pair of vertices i, j, every time a new node is added to the intermediate set $\{1, 2, \ldots, h-1\}$ the shortest path $p_{ij}^{(h)} = \min\{p_{ij}^{(h-1)}, p_{ih}^{(h-1)} + p_{hj}^{(h-1)}\}$. From the previous observation, keeping the minimum between the shortest path without and with h fore each intermediate until $h = |V|$, suffice to assure that $p_{ij}^{|V|}$ contains the shortest path from i to j within the input graph. The Floyd Warshall time complexity is $\Theta(|V|^3)$ because for each intermediate vertex in V it computes the minimum operation for every possible pair of vertices in V.

The Johnson-Dijkstra algorithm solves the all-pairs shortest-paths problem for sparse directed weighted graphs that do not have negative-weighted cycles. It uses a re-weighting technique which assure that a positive weight will be assigned to all edges and that the shortest paths among the nodes pairs remain unchanged. It then computes the shortest paths, using the positive re-weighted edges, for each vertex in V by running $|V|$ times the Dijkstra's algorithm with a Fibonacci heap implementation of the min-priority queue. The re-weighting is achieved by adding an external vertex $s \notin V$ to the graph that has only outgoing edges. each with weight 0, pointing to all the vertices in V. From this new graph, the algorithm computes the SSSP from s to all the other vertices using the Bellman-Ford algorithm, and assigns to each vertex $u \in V$ the shortest path weight $h(u)$

from s to u. The re-weighting consist of assigning to each edge $(u, v) \in E$ the new weight $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$. If the graph does not have negative weighted cycles than the new weight $\hat{w}(u, v)$ is positive for any edge $(u, v) \in E$ since, by the triangle inequality, it holds that $w(u, v)+h(u) \geq h(v)$. The Johnson-Dijkstra's algorithm computes the re-weighting in $O(|V||E|)$ time because of the Bellman-Ford algorithm complexity, and it computes the all-pairs shortest-paths in $O(|V|^2 \log |V| + |V||E|)$ time since it runs $|V|$times the Dijkstra's algorithm. Therefore the overall time complexity is $O(|V|^2 \log |V| + |V||E|)$.

Summarizing, the all-pairs shortest-paths problem could be solved in polynomial time in the size of the input graph.

2.4.2 Sum of Distances approximation

The exact computation of the sum of distances for every vertex in the graph is feasible when its input size is small enough. It became impractical when we deal with huge graph instances, therefore alternative computation techniques are needed.

In order to compute the sum of distances for every vertex it is necessary to calculate the distance between every pair of vertices within the graph, that is, solve the all-pairs shortest-paths problem. The alternative methods developed so far reduce the number of SSSP required to compute the sum of distances for every vertex in the graph by a smart sampling of the source vertices. They are able to estimate the sum of distances of every vertex in the graph with the distance information obtained from the SSSP of the sampled vertices. Every method has a different sampling technique that produce different guarantees for the estimation quality.

As already described earlier in the chapter, the closeness centrality of a vertex u is defined as the inverse of the sum of distances $cc_u = 1/W(u)$. If we are able to estimate the sum of distances with the value $\hat{W}(u)$ such that $(1 - \epsilon)W(u) \leq \hat{W}(u) \leq (1 + \epsilon)W(u)$, where $\epsilon > 0$ is the relative error, and we define the closeness centrality estimate as $\hat{cc}_u = 1/\hat{W}(u)$. Than a small relative error over the sum of distances estimate implies a small relative error on the closeness centrality estimate. In fact, it holds that $\hat{cc}_u \geq \frac{1}{(1+\epsilon)W(u)} = \frac{cc_v}{(1+\epsilon)}$ and $\hat{cc}_u \leq \frac{1}{(1-\epsilon)W(u)} = \frac{cc_v}{(1-\epsilon)}$. From these bounds, it is deduced that it exists a value ϵ' (with ϵ' that tends to ϵ when ϵ tends to 0) such that

$$(1 - \epsilon')cc_u \leq \hat{cc}_u \leq (1 + \epsilon') \tag{2.9}$$

2.5 Approximation Algorithms

Approximation algorithms have been designed to overcome the need of solving important intractable optimization problems by finding near-optimal solution in polynomial time. A combinatorial optimization problem is a computational problem $\Pi(I, S, \Phi, m)$ defined by a set of instances I, a set of solutions S, an objective function Φ and $m \in \{\min, \max\}$. For each instance $i \in I$ there is a subset $S_i \subseteq S$ of feasible solutions. Given $i \in I$ the problem requires to find an optimal solution, namely a feasible solution $s \in S_i$ which minimizes or maximizes the objective function $\Phi(s)$ according to m.

In the era of big data, even optimization problems that could be solved in polynomial time may be impractical due to the huge input sizes. Therefore approximation algorithms become useful to reduce the computational effort in exchange for a controlled loss in solution accuracy.

Suppose that the objective function returns a positive value for every feasible solution. An approximation algorithm A for a combinatorial optimization problem $\Pi(I, S, \Phi, m)$ has an *approximation ratio* of $\rho(n)$ if, for any instance $i \in I$ of size n, it returns a feasible solution $A(i) \in S_i$, such that

$$\frac{\Phi(A(i))}{\min_{s \in S_i} \Phi(s)} \leq \rho(n) \quad \text{if } m = \min \quad \text{or} \quad \frac{\max_{s \in S_i} \Phi(s)}{\Phi(A(i))} \leq \rho(n) \quad \text{if } m = \max \tag{2.10}$$

that is, it returns a feasible solution within a factor $\rho(n)$ with respect to the optimal solution of the optimization problem. The algorithm A is called a $\rho(n)$-*approximation algorithm*.

For many optimization problems, polynomial-time approximation algorithm with small constant approximation ratios are known, although there are problems whose best known approximation algorithm has a ratio that grows as a function of the input size n. Another interesting class of approximation algorithms are those designed to achieve increasingly better approximation ratios by using more and more computation time. That is, it is possible to trade the computational effort in exchange for the quality of the approximation. These are known as *approximation schemes*. An approximation schemes provides a $(1 + \epsilon)$-approximation algorithm for every value $\epsilon > 0$, which is given as part of the input.

Approximation schemes could be polynomial in time if their time complexity is polynomial in the size n of its input instance. Moreover they could be fully polynomial in time if their time complexity is polynomial in both $1/\epsilon$ and the size n of the input instance. Polynomial time approximation schemes may be exponential with respect to $1/\epsilon$ which implies an exponential growth in complexity as the approximation is more and more precise. Instead fully polynomial time

approximation schemes could trade better approximation with a polynomial loss in time complexity.

Both approximation algorithms and approximation schemes could be randomized, that is, for every input instance the execution depends on a number of random choices which may return different feasible solutions. Typically, the complexity and/or the approximation factor featured by an approximation algorithm are analyzed probabilistically.

2.6 Center-Based Clustering

Our novel closeness centrality estimation approaches have been inspired by some well known techniques used to solve center-based k-clustering problems. Given a metric space (M, d). A set $P \subseteq M$ of n points and an integer k such that $1 \leq k \leq n$. A k-clustering is a tuple $\mathcal{C} = (C_1, C_2, \ldots, C_k;\ c_1, c_2, \ldots, c_k)$ where $(C_1, C_2, \ldots, C_k)$ defines a partition of P and $c_1, c_2, \ldots, c_k$ are suitable selected centers for the clusters such that $c_i \in C_i$ for every $1 \leq i \leq k$. A k-clustering problem for (M, d) is an optimization problem whose instances are finite sets of points $P \subseteq M$. For each instance P, the problem requires to compute a k-clustering of P, which minimizes a suitable objective function $\Phi(\mathcal{C})$. The most relevant clustering problems are the *k-center*, *k-means* and *k-median* clustering whose objective functions are respectively

- $\Phi_{k\text{-center}}(\mathcal{C}) = \max_{i=1}^{k} \max_{x \in C_i} d(x, c_i)$;
- $\Phi_{k\text{-means}}(\mathcal{C}) = \sum_{i=1}^{k} \sum_{x \in C_i} (d(x, c_i))^2$;
- $\Phi_{k\text{-median}}(\mathcal{C}) = \sum_{i=1}^{k} \sum_{x \in C_i} d(x, c_i)$.

Our first approach takes inspiration from the *Farthest-First Traversal* (FFT) algorithm, proposed by Gonzales in [Gon85], which is a 2-approximation sequential algorithm that solves the k-center clustering problems. The solution to this problem is the k-clustering that minimizes the objective function defined above, that is, the maximum distance of an input point from the closest center. The algorithm takes in input a set P of n points from a metric space (M, d) and an integer $k > 1$ which determines the number of clusters. A point x is assigned to the cluster C_j if the distance $d(x, c_j)$ to its center c_j is the lowest among the distances $d(x, c_i)$ to the other cluster centers c_i for $1 \leq i \leq k, i \neq j$. The centers are selected and added to an initially empty set S in an iterative fashion. Except the first center c_1, which is arbitrarily selected from P, the center c_i at the i^{th} iteration is the farthest point from set $S = \{c_1, c_2, \ldots, c_{i-1}\}$, where the distance of a point x from S is defined as $d(x, S) = \min\{d(x, c) : c \in S\}$. This means that the new cluster center at iteration i is $c_i = \text{argmax}_{x \in P - S}\, d(x, S)$. It is possible

to assign the points in P to their corresponding clusters during or after the $k-1$ iterations. In the former case it is necessary to keep track of the closest center index for each point $x \in P$. Firstly all the points are assigned to the cluster centered at c_1. At every iteration i if the new selected center is closer to the point x than the previously selected closest center than the index is updated to i. In the latter case, once all the k centers have been defined, each point is assigned to the cluster of the closest center by looping through all of them once. In both cases the time complexity is $O(nk)$.

Other two algorithms from which we took inspiration are the k-means++ and k-median++. The former is used to initialize the centers (centroids) of the well-known Lloyd's algorithm [Llo57] for the k-means clustering problem. The latter is used to initialize the centers (medoids) of the Partitioning Around Medoids algorithm which is used to find sub optimal solutions for k-median clustering problems. Both the approaches are capable finding good approximations for their respective clustering problems. The procedure they use for the center selection is similar to the one adopted by FFT, that is, they use the same iterative procedure but the centers are randomly selected at each iteration based on probabilities values that depend on the distances of the points to the set S. The difference between the two probabilistic approaches consists in the computation of such probabilities. The first point is selected uniformly at random in both of the approaches. At iteration i the next center c_i is selected from $P - S$, with $S = \{c_1, c_2, \ldots, c_{i-1}\}$, where for each $x \in P-S$ the probability of being selected is $\frac{(d(x,S))^2}{\sum_{y \in P-S} (d(y,S))^2}$ for the k-means++ algorithm and $\frac{d(x,S)}{\sum_{y \in P-S} d(y,S)}$ for the k-median++ algorithm. As for the FFT algorithm, it is possible to keep track of the clusters within the iteration or compute them at the end by assigning the points to their closest center cluster.

Chapter 3

Traditional Approaches

In this chapter we describe in detail the two approaches already presented in literature. For each of them we describe, specify the algorithm and analyze the quality of estimations it provides. Both designs uses probabilistic sampling techniques and are able to compute the estimates after a full iteration over that sample. The reader may see in the next chapter that our approaches could all be implemented to progressively refine the estimates and could be stopped at any iteration and give a result.

3.1 Eppstein and Wang's Method

In [EW04] Eppstein and Wang propose the first method designed to reduce the computational effort required for the computation of the normalized closeness centrality in exchange for a controlled loss of accuracy. They apply a random sampling technique to approximate the inverse normalized closeness centrality of all vertices in a weighted connected graph $G = (V, E)$ to within an additive error of $\epsilon\Delta$ with high probability in time $O(\frac{\log|V|}{\epsilon^2}(|V|\log|V|+|E|))$, where $\epsilon > 0$ and Δ is the diameter of the graph. Furthermore they prove that their approach provides a near-linear time $(1+\epsilon)$-approximation to the centrality of all vertices for graphs that exhibit the small world phenomenon.

3.1.1 Algorithm

The algorithm they propose, called RAND, is a randomized approximation algorithm able to estimate the closeness centrality of every vertex in the graph. Given in input a weighted connected graph $G = (V, E)$, RAND selects uniformly at random k sample vertices and computes from each of them the SSSP to all the other

vertices in the graph. Considering $S = \{v_1, v_2, \ldots, v_k\}$ to be the set of the sampled vertices, the estimated normalized closeness centrality $\hat{c}_u$ of a vertex $u \in V$ is computed in terms of the average distance $\bar{d}(u, S) = \frac{1}{k}\sum_{i=1}^{k} d(v_i, u)$ to the sample vertices as follows

$$\hat{c}_u = \frac{|V| - 1}{|V|\frac{1}{k}\sum_{i=1}^{k} d(v_i, u)} = \frac{|V| - 1}{|V|\bar{d}(u, S)} \tag{3.1}$$

The time complexity depends on the number k of sample vertices and the SSSP algorithm used. Considering a sparse positive weighted connected graph, the algorithm runs in $O(k(|V| \log |V| + |E|)$ time using Dijkstra's algorithm.

We describe a variant of the RAND algorithm which, for every vertex, estimates the sum of the distances with all other vertices. Recalling the relation between these two measures it is clear that Eppstein and Wang's method consider the sum of distance estimate of a vertex u to be $\hat{W}(u) = \sum_{i=1}^{k} \frac{|V|}{k} d(v_i, u)$, that is, $|V|$ times the average distance of vertex u to the sample set S. The pseudocode of the algorithm is reported below as Algorithm 3.

Algorithm 3 Eppstein and Wang's RAND algorithm

```
1:  procedure RAND(G, k)
2:      // Initialize the estimates
3:      for all vertices u ∈ V do
4:          Ŵ(u) ← 0
5:      end for
6:      S ← ∅
7:      for i ← 1 to k do
8:          v_i ← SELECTRANDOMVERTEX(V − S)
9:          S ← S ∪ {v_i}
10:         compute SSSP distances d(v_i, u) from v_i to all other vertices u ∈ V
11:         Ŵ(v_i) ← Σ_u d(v_i, u)
12:         for all u ∈ V − S do
13:             Ŵ(u) ← Ŵ(u) + (|V|/k) d(v_i, u)
14:         end for
15:     end for
16:     return (u, Ŵ(u)) for u ∈ V
17: end procedure
```

3.1.2 Analysis

We reformulate the analysis of [EW04] focusing on the sum of distances estimate instead of the inverse normalized closeness centrality estimate.

We first prove that for any sample size k and for any vertex u holds that the expected value $\hat{W}(u)$ is equal to the sum of distances $W(u)$.

Theorem 3.1.1. $E[\hat{W}(u)] = W(u)$

Proof. Let us consider a connected graph $G = (V, E)$ and let $n = |V|$. Given that the probability of selecting a generic vertex v_i at random within k extractions without replacement is

$$\begin{aligned} &Pr\{\text{“extract } v_i \text{ within } k \text{ extractions”}\} = \\ &= 1 - Pr\{\text{“NOT extract } v_i \text{ within } k \text{ extractions”}\} = \\ &= 1 - \frac{n-1}{n} \cdot \frac{n-2}{n-1} \cdot \frac{n-3}{n-2} \cdot \dots \cdot \frac{n-k+1}{n-k+2} \cdot \frac{n-k}{n-k+1} = \\ &= 1 - \frac{n-k}{n} = \frac{k}{n} \end{aligned} \tag{3.2}$$

Then the expected value of the sum of distances $\hat{W}(u)$ is

$$\begin{aligned} E[\hat{W}(u)] = n\,\frac{1}{k} E\Big[\sum_{i=1}^{k} d(v_i, u)\Big] = n\,\frac{1}{k}\sum_{i=1}^{n} E[d(v_i, u)] = \\ = \frac{n}{k}\sum_{i=1}^{n} \frac{k}{n}\, d(v_i, u) = W(u) \end{aligned} \tag{3.3}$$

□

The analysis of the algorithm requires the use of the Hoeffding's probability bounds for sums on independent random variables [Hoe63].

Lemma 3.1.2 (Hoeffding [Hoe63]). *If $x_1, x_2, \dots, x_k$ are independent random variables, $a_i \le x_i \le b_i$, and $\mu = E[\sum \frac{x_i}{k}]$ is the expected mean, then for $\xi > 0$*

$$Pr\left\{\left|\frac{\sum_{i=1}^{k} x_i}{k} - \mu\right| \ge \xi\right\} \le 2 \cdot e^{-2k^2\xi^2 / \sum_{i=1}^{k} (b_i - a_i)^2}. \tag{3.4}$$

Theorem 3.1.3. *Let $G = (V, E)$ be a connected graph with diameter Δ. For any $\epsilon > 0$, by setting $k = O(\log |V|/\epsilon^2)$ we have that, with high probability, for each vertex $u \in V$ the estimator $\hat{W}(u)$ computed by* Rand *for the sum of distances from u is within an additive factor $\xi = \epsilon|V|\Delta$ of the exact value $W(u)$.*

Proof. We want to prove that the estimation error of our estimator $\hat{W}(u)$ is at most $\xi = \epsilon|V|\Delta$ for any vertex $u \in V$. We know from Theorem 3.1.1 that $E[\hat{W}(u)] = W(u)$. Therefore we can bound the probability that the absolute difference between the sum of distances estimates $\hat{W}(u)$ and the actual sum of distances

$W(u)$ is more than ξ by applying the Hoeffding's bound with $x_i = |V|d(v_i, u)$, $\mu = W(u)$, $a_i = 0$, and $b_i = |V|\Delta$.

$$\begin{aligned} Pr\Big\{\Big|\frac{1}{k}\sum_{i=1}^{k} x_i - \mu\Big| \geq \xi\Big\} = \\ Pr\Big\{\Big|\frac{1}{k}\sum_{i=1}^{k} |V|d(v_i,u) - W(u)\Big| \geq \xi\Big\} = \\ Pr\Big\{\big|\hat{W}(u) - W(u)\big| \geq \xi\Big\} \leq 2 \cdot e^{-2k^2\xi^2/\sum_{i=1}^{k}(b_i-a_i)^2} \\ \leq 2 \cdot e^{-2k^2\xi^2/k(|V|\Delta)^2} \\ = 2 \cdot e^{-\Omega(k\xi^2/|V|^2\Delta^2)} \end{aligned} \tag{3.5}$$

For $\xi = \epsilon|V|\Delta$, if we run the RAND algorithm with $k = \alpha\frac{\log|V|}{\epsilon^2}$ samples, such that $\alpha \geq 1$, with high probability the absolute error between the sum of distances estimator $\hat{W}(u)$ and the sum of distances $W(u)$ of every vertex u in the graph will be at most $\epsilon|V|\Delta$. □

From the Theorem 3.1.3 it is possible to bound with high probability also the relative error of the sum of distances estimator $\hat{W}(u)$ with respect to the sum of distances $W(u)$ for every vertex u in the graph.

Corollary 3.1.4. *Let $G = (V, E)$ be a connected graph with diameter Δ. Suppose that for every vertex $u \in V$ its average distance from the other vertices, namely $\bar{d}(u, V) = W(u)/|V|$, is such that $\Delta/\bar{d}(u, V) < t$, for some $t > 0$. Then, for any $\epsilon > 0$, by running the* RAND *algorithm with $k = O(\log|V|/\epsilon^2)$ we have that, with high probability, for each vertex $u \in V$ the relative error of $\hat{W}(u)$ with respect to $W(u)$ is less than t.*

Proof. Recalling the final inequality of the Theorem 3.1.3

$$Pr\Big\{\big|\hat{W}(u) - W(u)\big| \geq \xi\Big\} \leq 2 \cdot e^{-\Omega(k\xi^2/|V|^2\Delta^2)} \tag{3.6}$$

If we divide by $W(u)$ both sides we have

$$\begin{aligned} Pr\Big\{\frac{\big|\hat{W}(u) - W(u)\big|}{W(u)} \geq \epsilon\frac{|V|\Delta}{W(u)}\Big\} \leq 2 \cdot e^{-\Omega(k\epsilon^2)} \\ Pr\Big\{\frac{\big|\hat{W}(u) - W(u)\big|}{W(u)} \geq \epsilon\frac{\Delta}{\bar{d}(u,V)}\Big\} \leq 2 \cdot e^{-\Omega(k\epsilon^2)} \end{aligned} \tag{3.7}$$

If it holds that $\Delta/\bar{d}(u, V) \leq t$ for every vertex $u \in V$ than we conclude our proof. □

From this last proof it is clear that this approach guarantees small relative errors over the computation of the closeness centrality only for particular types of graphs, that is, when the ratio between the diameter and the average distance of every vertex is bounded by a small enough constant t. For instance, in a small world graph the diameter is relatively small with respect to the size of the input and the average distances are similar for each vertex, which means that in such graphs the RAND algorithm may perform well. We also expect good performances on road networks were the average distance is similar for each vertex and within a constant factor to the diameter. The time complexity required to have such guarantees will be $O(\frac{\log |V|}{\epsilon^2}(|V| \log |V| + |E|)$.

3.2 Chechik *et al.* 's Method

The work done by by Chechik *et al.* [CCK15], proposes sharper tools to estimate the average distance between points and the closeness centrality of vertices on very large data sets. In particular they present estimators with tight statistical guarantees whose computation is highly scalable.

They consider inputs that are either in the form of an undirected positive weighted connected graph or a set of points in a metric space. In case of graphs, the distances of the underlying metric correspond to the lengths of the shortest paths between the vertices. They focus on the estimation of the sum of distances $W(u) = \sum_{v \in V} d(u, v)$, where V could be a set of points in a metric space or a set of vertices of a graph. Given the definitions of average distance $W(u)/|V|$ and normalized closeness centrality $(|V| - 1)/W(u)$, a small relative error on the estimation of $W(u)$ imply a small relative error on both values.

The approximation quality of the sum of distances estimator $\hat{W}(u)$ is measured by the normalized root mean square error $\sqrt{E[(\hat{W}(u) - W(u))^2]}/\mu$ where μ is the mean of the estimator. When the estimator is unbiased, as in this case, such measure is the ratio between the variance σ and the mean μ of their estimator which is called Coefficient of Variation (CV) in [CCK15]. Chebyshev's inequality implies that the probability that their estimator is within a relative error of η from is mean is at least $1 - (CV)^2/\eta^2$. Hence a CV of ϵ implies that the estimator is within a relative error $\eta = c\epsilon$ from its mean with probability $\geq 1 - 1/c^2$.

Therefore their method computes estimates within a small Coefficient of variation ϵ and a small relative error with high probability, using $O(\log |V|/\epsilon^2)$ SSSP computations.

In the following sections we will concentrate on the algorithm whose inputs are in the form of graphs and we will refer to it as CCK algorithm.

3.2.1 Algorithm

The Cck algorithm is based on the computation of a single weighted sample of vertices that provides sum of distances estimates for all the vertices of the input graph with tight statistical guarantees. The computations are divided in *sampling* and *estimation* phases. The former receives in input the graph $G(V, E)$, an integer k and a base set S_0, and produces in output a weighted sample of vertices. The weighted sample and the graph are then passed as inputs to the estimation phase, which computes the estimates.

Algorithm 4 Cck Sampling algorithm

```
1: procedure SAMPLING(G = (V, E), k, S_0)
2:     // Compute sampling coefficients γ_u
3:     for all u ∈ V do
4:         γ_u ← 1/|V|
5:     end for
6:     for all v ∈ S_0 do
7:         compute SSSP distances d(v, u) from v to all other vertices u ∈ V
8:         W(v) ← Σ_u d(v, u)
9:         for all u ∈ V do
10:            γ_u ← max{γ_u, d(v,u)/W(v)}
11:        end for
12:    end for
13:    // Compute the sampling probabilities p_u
14:    for all u ∈ V do
15:        p_u ← min{1, kγ_u}
16:    end for
17:    // Compute the Poisson sample according to p_u
18:    S ← ∅
19:    for all u ∈ V do
20:        if RANDOMNUMBER(0, 1) < p_u then
21:            S ← S ∪ {(u, p_u)}
22:        end if
23:    end for
24:    return S
25: end procedure
```

The base set covers a key role on the statistical guarantees of the final estimates. As described in Algorithm 4, it is used to assign to each vertex u in the graph a coefficient γ_u that combined with the parameter k determines its sampling

probability $p_u \equiv \min\{1, k\gamma_u\}$. These probabilities are used both for the Poisson sampling and as weights of the selected vertices in the weighed sample. Such sample is indeed a collection $\{(v, p_v)\}$ of vertices and their respective sampling probability. The estimates are calculated by accumulating the contribution of each sampled vertex to the sum of distances of all the other vertices, as described in Algorithm 5. The contribution of a sampled vertex v to the estimate $\hat{W}(z)$ of vertex $z \in V$ is given by the distance $d(z, v)$ divided by the weight p_v of the sampled vertex. That is, given the set S of sampled vertices, returned from the sampling phase, the sum of distances estimate for a vertex $z \in V$ is calculated as follows

$$\hat{W}(z) = \sum_{v \in V} \hat{d}(z, v) \quad \text{where} \quad \hat{d}(z, v) = \begin{cases} \frac{d(z,v)}{p_v} & \text{if } v \in S \\ 0 & \text{if } v \notin S \end{cases} \tag{3.8}$$

Algorithm 5 Cck Estimation algorithm

```
1: procedure Estimation(G = (V, E), S = {(v, p_v)})
2:     // Where S is the sample returned by Sampling(G, k, S_0)
3:     for all z ∈ V do
4:         Ŵ(z) ← 0
5:     end for
6:     for all v ∈ S do
7:         compute SSSP distances d(v, z) from v to all other vertices z ∈ V
8:         Ŵ(v) ← Σ_{z∈V} d(v, z)
9:         for all z ∈ V − S do
10:            Ŵ(z) ← Ŵ(z) + d(v, z)/p_v
11:        end for
12:    end for
13:    return (z, Ŵ(z)) for z ∈ V
14: end procedure
```

In order to obtain a sample set S able to guarantee a small coefficient of variation and a small relative error for the estimate $\hat{W}(z)$, the sampling probability p_u of each vertex $u \in V$ should be (roughly) proportional to its distance $d(z, u)$ from z. This approach, referred to as *Probability Proportional to Size* (PPS) sampling [CCK15], favors the selection, during the sampling process, of the vertices that give the higher contribution to the sum of distances of vertex z. Furthermore, in order to guarantee small relative errors for all the sum of distances estimates, the rough proportionality of each PPS p_u must hold for every vertex z in the graph.

The exact computation of these universal PPS requires to calculate the all-

pair shortest-paths, which we are trying to avoid. Luckily an approximation of such universal PPS is good enough to guarantee small relative errors estimates. To build an approximate universal PPS it must hold for each vertex u that $\gamma_u \geq c\, d(u,z)/W(z)$ for some constant c and for every vertex z. In practice to obtain such coefficients it is sufficient to have a well positioned vertex inside the base set S_0, which informally is a vertex that lays in the region of the graph where the vertices are more dense and close together. Since half of the vertices in a graph happen to have this property, building the base set by extracting at random $O(\log |V|)$ vertices from the graph is enough to guarantee with high probability the presence of one such vertex in the set.

Assuring universal approximate PPS implies that the sum of distances $W(z)$ for every vertex z can be estimated unbiasedly within a small relative error of ϵ with high probability with a sample of size $O(\frac{\log |V|}{\epsilon^2})$.

In the following subsection we review the analysis carried out in [CCK15], which explains the rationale behind the crucial choices made by the algorithm, and we will formally define the concepts mentioned before.

3.2.2 Analysis

The goal of this section is to prove that, given an input graph $G = (V, E)$, Algorithms 4 and 5 provide sum of distance estimates $\hat{W}(u)$ for every vertex $u \in V$ within a small relative error ϵ with high probability using $O(\frac{\log |V|}{\epsilon^2})$ SSSP distance computation.

The authors of [CCK15] start with the following lemma

Lemma 3.2.1. *Suppose that S_0 contains a vertex v. Consider an arbitrary vertex $z \in V$ such that v is the $(q|V|)^{th}$ closest vertex to z. Then for all vertices $u \in V$,*

$$\gamma_u \geq \frac{1-q}{4} \cdot \frac{d(z,u)}{W(z)}. \tag{3.9}$$

Proof. From the specification of Algorithm 4, since the sampling coefficients γ_u are initialized to $\frac{1}{|V|}$ and are updated to $\gamma_u \leftarrow \max\left\{\gamma_u, \frac{d(u,v)}{W(v)}\right\}$ for $v \in S_0$, then they satisfy

$$\gamma_u \geq \max\left\{\frac{1}{|V|}, \frac{d(v,u)}{W(v)}\right\} \tag{3.10}$$

Let $Q = d(z,v)$. Consider the classification of the vertices $u \in V$ "close" vertices and "far" vertices according to their distance from z (Figure 3.1), that are respectively the sets

$$\begin{aligned} L &= \{u \in V : d(z,u) \leq 2Q\} \\ H &= \{u \in V : d(z,u) > 2Q\} \end{aligned} \tag{3.11}$$

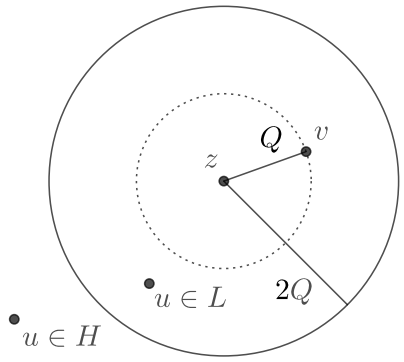

Figure 3.1: Visualization of the L and H sets with respect to the vertex z and its $q|V|^{th}$ closest vertex v. The set L of "close" vertices correspond to the ball of radius $2Q$ around z. The set H of "far" vertices instead correspond to all the vertices outside that ball.

Since $\gamma_u \geq \frac{1}{|V|}$, for $u \in L$ we have

$$\begin{aligned}\gamma_u \geq \frac{1}{|V|} \geq & \left(\frac{1-q}{2}\right)\left(\frac{2}{1-q}\right)\frac{1}{|V|} = \left(\frac{1-q}{2}\right)\left(\frac{2Q}{(1-q)Q}\right)\frac{1}{|V|} \\ \geq & \left(\frac{1-q}{2}\right)\frac{d(z,u)}{W(z)}\end{aligned} \tag{3.12}$$

To prove the last inequality of the above equation we observe that $d(z,u) \leq 2Q$, since $u \in L$, and we prove, in what follows, that $W(z) \geq (1-q)|V|Q$. Recall that v is the $(q|V|)^{th}$ vertex closest to z. Therefore there are $(1-q)|V|$ vertices more distant from z than v. Since we have defined $d(z,v) = Q$, it holds that $d(z,u) > Q$ for every vertex u more distant than v from z. We split the sum of distances from z in two sums, one considering the vertices that are within the distance Q and one for the others, thus we have that

$$\begin{aligned}W(z) = \sum_{u \in V} d(z,u) = & \sum_{u \in V : d(z,u) \leq Q} d(z,u) + \sum_{u \in V : d(z,u) > Q} d(z,u) \\ \geq & (q|V|-1)\ \min\{d(z,u) : d(z,u) \leq Q\} + (1-q)|V|Q \\ \geq & (q|V|-1)\, c_{min} + (1-q)|V|Q \geq (1-q)|V|Q\end{aligned} \tag{3.13}$$

where from the sum of the vertices within distance Q the distance form z to itself is not counted and $c_{min} > 0$ is the minimum distance of a vertex from z. Therefore $W(z) \geq (1-q)|V|Q$ which conclude the proof of the last inequality of Equation 3.12.

For all $u \in V$, we have that $d(v,u) \geq d(z,u) - Q$ by the triangle inequality. In addition, for the same reason it also holds that $d(v,u) \leq d(z,u) + Q$ for all $u \in V$. This imply that summing the distances from v to all the other vertices u we have the following

$$\begin{aligned}\sum_{u \in V} d(v,u) \leq & \sum_{u \in V} d(z,u) + |V|Q \\ W(v) \leq & W(z) + |V|Q\end{aligned} \tag{3.14}$$

Substituting into Equation 3.10 the first triangle inequality and this last inequality, for every vertex $u \in V$ we have that

$$\gamma_u \geq \frac{d(v,u)}{W(v)} \geq \frac{d(z,u) - Q}{W(z) + |V|Q} \tag{3.15}$$

In particular, for $u \in H$, which , by definition, is such that $d(z,u) > 2Q$, we have $d(z,u) - Q \geq \frac{1}{2}d(z,u)$. From $W(z) \geq (1-q)|V|Q$ we obtain

$$|V|Q \leq \frac{W(z)}{1-q} \tag{3.16}$$

hence,

$$W(z) + |V|Q \leq W(z)\Big(1 + \frac{1}{1-q}\Big) = W(z)\Big(\frac{2-q}{1-q}\Big) \tag{3.17}$$

Substituting this last inequality and $d(z,u) - Q \geq \frac{1}{2}d(z,u)$ in Equation 3.15 we get

$$\gamma_u \geq \frac{1}{2}\Big(\frac{1-q}{2-q}\Big)\frac{d(z,u)}{W(z)} \quad \text{for all } u \in H \tag{3.18}$$

From this last equation we have that $\gamma_u \geq \frac{1-q}{4}\frac{d(z,u)}{W(z)}$ for all $q \in [0,1]$ (which is the range of q by definition). By combining Equation 3.12 which holds for vertices in L and Equation 3.18 which holds for vertices in H we have

$$\gamma_u \geq \frac{1-q}{4}\frac{d(z,u)}{W(z)} \quad \text{for all } u \in V \text{ and for all } q \in [0,1] \tag{3.19}$$

which concludes the proof. □

Lemma 3.2.2. *Consider a set of sampling coefficients γ_u such that for a vertex z, for all u and for some $c > 0$, $\gamma_u \geq c\frac{d(z,u)}{W(z)}$. Let S be a sample obtained with probabilities $p_u = \min\{1, k\gamma_u\}$, and let $\hat{W}(z)$ be the inverse probability estimator as in Equation 3.8. Then*

$$Var[\hat{W}(z)] \leq \frac{W(z)^2}{ck} \tag{3.20}$$

Proof. Given the estimator of Equation 3.8 and the definition of variance

$$\begin{aligned}
Var[\hat{W}(z)] &= Var\Big[\sum_{u\in V}\hat{d}(z,u)\Big] = \sum_{u\in V} Var[\hat{d}(z,u)] \\
&= \sum_{u\in V}\mathrm{E}\Big[\Big(\hat{d}(z,u)-\mathrm{E}[\hat{d}(z,u)]\Big)^2\Big] \\
&= \sum_{u\in V}\mathrm{E}\Big[\Big(\hat{d}(z,u)-d(z,u)\Big)^2\Big] \\
&= \sum_{u\in V}\mathrm{E}\Big[\hat{d}(z,u)^2-2\hat{d}(z,u)\,d(z,u)+d(z,u)^2\Big] \\
&= \sum_{u\in V}\Big(\mathrm{E}[\hat{d}(z,u)^2]-2\mathrm{E}[\hat{d}(z,u)]\,d(z,u)+d(z,u)^2\Big) \\
&= \sum_{u\in V}\Big(\frac{d(z,u)^2}{p_u}-d(z,u)^2\Big) \\
&= \sum_{u\in V}\Big(\frac{1}{p_u}-1\Big)d(z,u)^2
\end{aligned} \tag{3.21}$$

Note that the vertices u for which $p_u = 1$ contribute 0 to the variance of the estimate of a vertex z. For the other vertices they use the lower bound $p_u \geq ck\frac{d(z,u)}{W(z)}$ that holds when the sampling coefficients γ_u are approximate PPS.

$$\begin{aligned}
\sum_{u\in V}\Big(\frac{1}{p_u}-1\Big)d(z,u)^2 &= \sum_{u\in V:p_u<1}\Big(\frac{1}{p_u}-1\Big)d(z,u)^2 \\
&\leq \sum_{u\in V:p_u<1}\frac{1}{p_u}d(z,u)^2 \\
&\leq \sum_{u\in V:p_u<1}\frac{W(z)}{ck\,d(z,u)}d(z,u)^2 = \frac{W(z)}{ck}\sum_{u\in V:p_u<1}d(z,u) \\
&\leq \frac{W(z)}{ck}W(z) = \frac{W(z)^2}{ck}
\end{aligned} \tag{3.22}$$

which conclude the proof. □

Lemma 3.2.1 demonstrates that the coefficients γ_u for each $u \in V$ are lower bounded by $\frac{1-q}{4}\cdot\frac{d(z,u)}{W(z)}$ for $q \in [0,1]$ and for every vertex $z \in V$. Lemma 3.2.2 demonstrates that if we have γ_u coefficients that satisfy the approximate Probability Proportional to Size definition for a vertex z, it is possible to upper bound the variance of the sum of distances of that vertex. In what follows, the notion of *well positioned vertex* is introduced and it is proved that if the base set contains one such vertex then indeed the γ_u coefficients satisfy the PPS definition. Moreover,

it will be also proved that under these circumstances, the estimate of the sum of distances for every vertex has a small relative error.

Let us define the notion of *well positioned* vertex. Let the *median distance* of a vertex u, denoted by $m(u)$, be the distance between u and the $\lceil 1 + |V|/2 \rceil$ closest vertex to $u \in V$. Furthermore she defines $\text{MinMed} = \min_{u \in V} m(u)$ to be the minimum median distance of any vertex $u \in V$.

Definition 3.1. *A vertex is well positioned if $m(u) \leq 2\,\text{MinMed}$, that is, the median distance $m(u)$ of u is within a factor 2 of the minimum median distance.*

We now show that most of the vertices are well positioned.

Lemma 3.2.3. *Let v be such that its $m(v) = \text{MinMed}$. Then all $\lceil 1 + |V|/2 \rceil$ vertices in V that are closest to v are well positioned.*

Proof. Let u be one of the $\lceil 1 + |V|/2 \rceil$ vertices closest to v. Then $d(u, v) \leq \text{MinMed}$ and a ball of radius $2\,\text{MinMed}$ around u contains all the $\lceil 1 + |V|/2 \rceil$ vertices closest to v. Therefore, $m(u) \leq 2\,\text{MinMed}$ and by Definition 3.1, u is a well positioned vertex. □

The well positioned vertices are interesting because of the following property

Lemma 3.2.4. *If v is a well positioned vertex, then for every vertex z we have that $d(z, v) \leq 3m(z)$.*

Proof. For every two vertices v and z we have that $d(v, z) \leq m(v) + m(z)$. In fact, there must be at least a vertex x that is both within distance $m(v)$ from v and within distance $m(z)$ from z. In other words, there must be at least one intersection between the $\lceil 1+|V|/2 \rceil$ closest vertices to v and the $\lceil 1+|V|/2 \rceil$ closest vertices to z because $\lceil 1 + |V|/2 \rceil + \lceil 1 + |V|/2 \rceil > |V|$. From this observation and the triangle inequality we have that

$$d(v, z) \leq d(v, x) + d(z, x) \leq m(v) + m(z) \tag{3.23}$$

If we choose v to be a well positioned vertex, that is $m(v) \leq 2\,\text{MinMed} \leq 2m(z)$ we have that $d(z, v) \leq 3m(z)$ which proves the lemma. □

Lemma 3.2.5. *Suppose that the base set S_0 contains a well positioned vertex v. Then for all vertices u,*

$$\gamma_u \geq \frac{1}{19} \max_{z \in V} \frac{d(z, u)}{W(z)} \tag{3.24}$$

Proof. Given an arbitrary vertex z and a parameter α, we partition the vertices in two sets L and H, similarly to the proof of Lemma 3.2.1, as follows

$$\begin{aligned} L &= \{u \in V : d(z,u) \leq \alpha\, m(z)\} \\ H &= \{u \in V : d(z,u) > \alpha\, m(z)\} \end{aligned} \quad \text{for } \alpha > 0. \tag{3.25}$$

By the definition of median distance there are $\lceil |V|/2 - 1 \rceil$ vertices that are distant form z more than $m(z)$, therefore we have

$$W(z) \geq \left(\left\lceil \frac{|V|}{2} - 1 \right\rceil \right) m(z) \geq m(z) \frac{|V|}{3} \tag{3.26}$$

where the last inequality holds for graphs that have more than 8 vertices.

We obtain that for all $u \in L$

$$\frac{d(z,u)}{W(z)} \leq \frac{\alpha\, m(z)}{m(z)\frac{|V|}{3}} = \frac{3\alpha}{|V|} \tag{3.27}$$

Therefore,

$$\gamma_u \geq \frac{1}{|V|} = \frac{3\alpha}{|V|} \frac{1}{3\alpha} \geq \frac{1}{3\alpha} \frac{d(z,u)}{W(z)} \quad \text{for } u \in L \tag{3.28}$$

Subsequently we consider $u \in H$. Since v is well positioned then Lemma 3.2.4 holds and we have that $d(z,v) \leq 3m(z)$. For the triangle inequality and since $u \in H$ implies that $d(z,u) > \alpha\, m(z)$ we have

$$\begin{aligned} d(v,u) &\geq d(z,u) - d(z,v) \geq d(z,u) - 3m(z) \\ &\geq d(z,u) - \frac{3}{\alpha} d(z,u) \\ &\geq \left(1 - \frac{3}{\alpha}\right) d(z,u) \end{aligned} \tag{3.29}$$

Since v is well positioned and we show above that $W(z) \geq m(z)\frac{|V|}{3}$, we have

$$\begin{aligned} W(v) &\leq W(z) + |V| d(z,v) \leq W(z) + 3|V| m(z) \\ &\leq W(z) + 9W(z) = 10W(z) \end{aligned} \tag{3.30}$$

Therefore

$$\gamma_u \geq \frac{d(v,u)}{W(u)} \geq \frac{\left(1 - \frac{3}{\alpha}\right) d(z,u)}{10W(z)} = \frac{1}{10}\left(1 - \frac{3}{\alpha}\right) \frac{d(z,u)}{W(z)} \quad \text{for } u \in H \tag{3.31}$$

Depending on whether a vertex u belongs to the sets L or the set H we have found respectively two lower bounds on γ_u presented in the last inequalities of Equation 3.28 and 3.31 depending on the parameter α. These inequalities have

the same measure $\frac{d(z,u)}{W(z)}$ multiplied by a different coefficient, respectively $c_1 = \frac{1}{3\alpha}$ and $c_2 = \frac{1}{10}\big(1 - \frac{3}{\alpha}\big)$. We want to find $\max_{\alpha>0} \min\{c_1, c_2\}$. Since the former is a monotonically decreasing function and the latter is a monotonically increasing function for $\alpha > 0$, in order to find the coefficient that maximizes the minimum value of the two coefficients, we should find α such that $c_1 = c_2$, that is, $\alpha = 19/3$. Substituting it into the two coefficients we have

$$\gamma_u \geq \frac{1}{19} \frac{d(z,u)}{W(z)} \quad \text{for } u \in V \tag{3.32}$$

therefore considering all $z \in V$ we have

$$\gamma_u \geq \frac{1}{19} \max_{z \in V} \frac{d(z,u)}{W(z)} \quad \text{for } u \in V \tag{3.33}$$

which concludes the proof. □

The following corollary is an immediate consequence of Lemmas 3.2.2 and 3.2.5.

Corollary 3.2.6. *If S_0 contains a well positioned vertex, then for any vertex z, $Var[\hat{W}(z)] \leq 19\frac{W(z)^2}{k}$*

One consequence of Lemma 3.2.5 is that the sampling coefficients γ_u cannot grow too much even if the base set S_0 includes all vertices.

Corollary 3.2.7. *Let*

$$\bar{\gamma}_u \equiv \max_{z \in V} \frac{d(z,u)}{W(z)} \,. \tag{3.34}$$

Then

$$\sum_{u \in V} \bar{\gamma}_u = O(1) \,. \tag{3.35}$$

Proof. From the specification of Algorithm 4, since the sampling coefficients γ_u are initialized to $\frac{1}{|V|}$ and are updated to $\gamma_u \leftarrow \max\left\{\gamma_u, \frac{d(u,v)}{W(v)}\right\}$ for $v \in S_0$, it holds that

$$\gamma_u \leq \frac{1}{|V|} + \max_{v \in S_0} \left\{ \frac{d(v,u)}{W(v)} \right\} \tag{3.36}$$

If we sum all the γ_u's we have that

$$\begin{aligned}\sum_{u\in V}\gamma_u &\leq \sum_{u\in V}\left(\frac{1}{|V|}+\max_{v\in S_0}\left\{\frac{d(v,u)}{W(v)}\right\}\right)\\ &\leq \sum_{u\in V}\left(\frac{1}{|V|}+\sum_{v\in S_0}\frac{d(v,u)}{W(v)}\right)\\ &\leq \sum_{u\in V}\frac{1}{|V|}+\sum_{u\in V}\sum_{v\in S_0}\frac{d(v,u)}{W(v)}=1+\sum_{v\in S_0}\sum_{u\in V}\frac{d(v,u)}{W(v)}=\\ &=1+\sum_{v\in S_0}\frac{W(v)}{W(v)}=1+\sum_{v\in S_0}1=1+|S_0|\end{aligned} \tag{3.37}$$

therefore $\sum_{u\in V}\gamma_u \leq 1+|S_0|$.

Consider the case where S_0 consists of a single well positioned vertex, thus we have that $\sum_{u\in V}\gamma_u \leq 2$. By Lemma 3.2.5 we have $\gamma_u \geq \frac{1}{19}\max_z \frac{d(z,u)}{W(z)}$. Therefore $\sum_{u\in V}\bar{\gamma_u} \leq 19\sum_{u\in V}\gamma_u \leq 38 = O(1)$, which concludes the proof. □

The following lemma establishes a probability bound on the relative error of the estimate of the sum of distances for every vertex z. The proof can be found in [CCK15].

Lemma 3.2.8. *If the sampling coefficients are approximate PPS for a vertex z, that is, there is a constant c such that for all vertices $u \in V$, $\gamma_u \geq c\frac{d(z,u)}{W(z)}$, and we use $k = O\left(\frac{\log|V|}{\epsilon^2}\right)$, then*

$$Pr\left[\frac{|\hat{W}(z)-W(z)|}{W(z)} \geq \epsilon\right] = O\left(\frac{1}{poly(|V|)}\right) \tag{3.38}$$

Proof. For $\tau = W(z)/(ck)$, we have

$$p_v \geq \min\{1, d(z,v)/\tau\} = \min\{1, ck\, d(z,v)/W(z)\}\,. \tag{3.39}$$

The contribution of a vertex v to the estimate $\hat{W}(z)$ is as follows. If $d(z,v) \geq \tau$, then the contribution is exactly $d(z,v)$. Otherwise, the contribution X_v of a vertex v is $d(z,v)/p_v \leq \tau$ with probability p_v and 0 otherwise.

The contributions X_v of the vertices with distance $d(z,u) \leq \tau$ are thus independent random variables, each in the range $[0,\tau]$ with expected value $d(z,v)$. The proof is completed by applying the Chernoff-Hoeffding bound to bound the deviation of the expected value of the sum of these random variables. More details can be found in [CCK15]. □

From the proof of Lemma 3.2.8, it follows that if the sampling coefficients are approximate universal PPS and k is large enough, but still $O(\log |V|/\epsilon^2)$, then the probability that the relative error of the estimate of each sum of distances exceeds ϵ can be made less than $1/|V|^a$ with $a > 2$, hence, by applying the union bound, all estimates will feature relative error at most ϵ with high probability.

From Lemma 3.2.5 it follows that we obtain the universal PPS sampling coefficients if S_0 includes a well positioned vertex. We would like this to happen with high probability. In [CCK15] the following approaches are proposed to this aim.

(i) From Lemma 3.2.3 we know that most vertices are well positioned, therefore taking a random sample U of $O(\log |V|)$ vertices, and choosing the vertex $x = \arg\min_{u \in U} m(u)$ with minimum median distance, means that we are guaranteed with probability $1 - O(poly(|V|))$ that u is well positioned. The SSSP required to identify a well positioned vertex with this procedure are $O(\log |V|)$.

(ii) Alternatively, it is possible to ensure that S_0 contains a well positioned vertex with high probability by simply placing $O(\log |V|)$ uniformly selected vertices in S_0. This procedure still requires $O(\log |V|)$ SSSP computations.

In the second case, we can argue that with a polynomially small error for each vertex z, one of the $\left\lceil 1 + \frac{|V|}{2} \right\rceil$ closest vertices to z is in S_0. Therefore we can apply 3.2.1 with $q \leq 1/2$ to obtain that with high probability, the sampling probabilities are approximate PPS for all vertices and thus the estimates will have polynomially small relative errors.

Chapter 4

Progressive Approaches

In this chapter we present our novel approaches, which are able to produce sum of distances estimates for every vertex of the input graph by progressively refining the solution while the sample is being built iteratively. In the first section, we describe a deterministic approach based on the Farthest-First Traversal (FFT) algorithm presented by Gonzalez in [Gon85]. In the second section, we present two probabilistic versions based on the same idea behind the FFT. Finally, in the last section, we present a method that combines the ideas behind the Cc algorithm and our approaches.

4.1 Farthest-First Traversal Methods

We propose a deterministic sample-based closeness centrality estimation algorithm that progressively refines its estimates by adding vertices to the sample set in the same fashion as the FFT algorithm, described in [Gon85], selects its clusters centers. As already explained in Section 2.6, the FFT is a 2-approximation sequential algorithm that solves the k-center clustering problem in a metric space (M, d) for a set of points $P \subseteq M$. The problem consist in finding the k-clustering $\mathcal{C} = (C_1, C_2, \ldots, C_k; c_1, c_2, \ldots, c_n)$ that minimizes, the maximum distance of a point from its reference center, that is, $\mathcal{C}$ minimizes the objective function

$$\Phi_{k\text{-center}}(\mathcal{C}) = \max_{i=1}^{k} \max_{x \in C_i} d(x, c_i)\,.$$

For a point x and a set S, define $d(x, S) = \min\{d(x, s) : s \in S$. The FFT algorithm selects the centers iteratively as follows. Let $S_j = \{c_, \ldots, c_j\}$ be the set of centers selected in the first j iterations. The next center c_{j+1} to be selected is

the point $x \in P$ maximizing $d(x, C_j)$, that is,

$$c_{j+1} = \underset{x \in V}{\operatorname{argmax}} \min_{c_i \in S_j} d(x, c_i)\,. \tag{4.1}$$

Our approach is an adaptation of the Gonzalez's technique to be used on graphs $G = (V, E)$ for the progressive estimation of the sum of distances $W(u)$, and consequently the closeness centrality, for every vertex $u \in V$. We propose two variations of the same algorithm where the difference is given by how we define the argument of the objective function to be minimized.

The main idea is to use the sum of distances of the sampled vertices to determine an upper and a lower bound to every sum of distances of any vertex in the graph. As in Gonzalez's approach, the centers are selected with the objective of minimizing the maximum distance from the vertices to their corresponding center, in our approach we want to build a sample that guarantees that every other vertex is at least at distance $\epsilon\Delta$ from it, where Δ represents the diameter of the graph. Indeed, by selecting at each iteration the farthest vertex we assure to reduce the maximum distance of a vertex to the selected sample.

Theoretically, our approach achieves small relative errors for graphs that present a mesh-like structure, such as road networks, using $k = O(\epsilon^D)$ sample vertices, where D represent the doubling dimension of the graph.

4.1.1 Algorithms

The approach we designed is described in Algorithm 6. It receives in input a weighted connected undirected graph $G = (V, E)$, a parameter k, which determines the maximum size of the vertex sample, and a parameter ϵ that defines the target maximum relative error of the estimates.

After the initialization step, the first next vertex v_n is arbitrarily picked among all the vertices in V. At each iteration, the next vertex of the previous iteration becomes the current vertex v_c and is added to the sample set S. Then the algorithm computes the SSSP distances from the current vertex to all the other vertices in the graph, and, using the distances information acquired, it updates the argument a_u, the lower bound lb_u, the upper bound ub_u and the relative error re_u of each vertex $u \in V$. After the updates, it computes the maximum relative error re_{max} and the next vertex to be added to the sample. Lastly it checks if the termination criterion is true, and, if not, it starts with another iteration. The TermCriteria$(k, \epsilon, |S|, re_{max})$ is a function that returns true only when the sample size is larger than k or when the maximum relative error is lower than ϵ.

For each vertex u the argument a_u keeps the minimum NewArgument(u, v_c) among all the currently selected sample vertices. Such function defines the two

versions of our algorithm. When NewArgument$(u, v_c) \equiv d(u, v_c)$ we have the first version of our approach, which selects the vertices in the exact same way as Gonzalez' does. When NewArgument$(u, v_c) \equiv d(u, v_c)/W(v_c)$ we have the second version of our approach, where the sum of distances $W(v_c)$ of the current vertex comes into play as a normalizing factor to determine which vertex is going to be selected next. In this second version, the effect we expect is to give more importance to the vertices that are farther from the more central vertices, thus producing a sample able to be in general closer to every vertex in the graph. In other words, at the i^{th} iteration, the first variation of the algorithm chooses the next sample vertex $v_n = \text{argmax}_{x \in V} \min_{c \in S_i} d(x, c)$ and the second variation of the algorithm chooses the next sample vertex $v_n = \text{argmax}_{x \in V} \min_{c \in S_i} d(x, c)/W(c)$, where S_i represent the set of sampled vertices at iteration i.

The idea behind the second variant is that if two vertices u, v have the same distance from the current sample set $d(u, S_j) = d(v, S_j)$, where the minimum distance from u to S_j is given by the vertex c_x and the minimum distance from v to S_j is given by the vertex c_y, if $W(c_x) > W(c_y)$ then c_y has an higher centrality, and thus increasing the normalized distance to the sample set of vertex v with respect to that of u.

The lower and upper bounds are defined based on the sum of distances estimates for the sampled vertices $c_1, \ldots, c_k$. The first idea could be to assign each vertex $u \in V$ to its closer sample vertex c_i the same way Gonzalez does in his algorithm, and use the sum of distances $W(c_i)$ and the distance $d(u, c_i)$ to determine the interval within which the estimate lies. These bounds can be simply obtained on the basis of the triangle inequality, stating that for each vertex v, $d(u, v) \leq d(u, c_i) + d(c_i, v)$, which clearly holds since the shortest path distances provide a metric space on the graph vertices. Thus $W(u) \leq W(c_i) + (|V| - 2)d(c_i, u)$, and $W(u) \geq W(c_i) - (|V| - 2)d(c_i, u)$, which gives us the interval endpoints. Observe that we multiply by $|V| - 2$ because the distance from u to itself is not counted and the the distance from u to c_i is already counted in $W(c_i)$ since the graph is undirected. Observe however that we are able to determine an interval for every vertex $u \in V$ with respect to every sampled vertex c_i, thus we may improve on the original idea by keeping the minimum upper bound and the maximum lower bound with respect to every c_i and for each vertex $u \in V$, which can only reduce the interval size and improve the solution. The final sum of distances estimates may be given by the arithmetic mean of the lower and upper bounds of each vertex $u \in V$.

Using the above approach, we are also able to compute an overestimate of the relative errors using the lower and upper bounds of each vertex $u \in V$ as described in Algorithm 6.

Algorithm 6 Farthest-First Traversal Method

procedure FFT($G = (V, E)$, k, ϵ)
  // Initialize arguments a_u, relative errors re_u,
  // lower bounds lb_u and upper bounds ub_u
  for all $u \in V$ **do**
    $a_u \leftarrow +\infty$; $re_u \leftarrow +\infty$; $lb_u \leftarrow 0$; $ub_u \leftarrow +\infty$;
  end for
  $S \leftarrow \emptyset$
  Assign the first vertex from V to the next vertex v_n
  repeat
    $v_c \leftarrow v_n$ // Set the next vertex v_n to be the current one
    $S \leftarrow S \cup \{v_c\}$
    Compute SSSP distances $d(v_c, u)$ from v_c to all other vertices $u \in V$
    $W(v_c) \leftarrow \sum_u d(v_c, u)$
    // Update argument, lower and upper bounds of v_c
    $a_{v_c} \leftarrow 0$; $lb_{v_c} \leftarrow W(v_c)$; $ub_{v_c} \leftarrow W(v_c)$;
    for all $u \in V - S$ **do**
      $a_u \leftarrow \min\{a_u, \text{NEWARGUMENT}(u, v_c)\}$
      $lb_u \leftarrow \min\{lb_u, W(v_c) - (|V| - 2)\, d(v_c, u)\}$
      $ub_u \leftarrow \max\{ub_u, W(v_c) + (|V| - 2)\, d(v_c, u)\}$
      $re_u \leftarrow (ub_u - lb_u)/(2 \max\{lb_u, d(v_c, u)\})$
    end for
    $re_{max} \leftarrow \max_{u \in V}\{re_u\}$ // Find maximum relative error re_{max}
    $v_n \leftarrow \text{argmax}_{u \in V - S}\{a_u\}$ // Compute next vertex v_n
  until TERMCRITERIA($k, \epsilon, |S|, re_{max}$)
  return $(\hat{W}(u) = (ub_u + lb_u)/2)$ for each $u \in V$
end procedure

In the next subsection we formalize the operations of the algorithm and we prove our claims on the small relative errors for mesh-like graphs.

4.1.2 Analysis

The proof of correctness requires some definitions, which we state in the following. We assume that each definition, unless otherwise mentioned, holds for a weighted undirected graph $G = (V, E)$, where for every $u, v \in V$, the distance $d(u, v)$ is given by the length of the shortest path in G between u and v.

Definition 4.1 (Diameter of a graph). *The diameter of graph G is the maximum distance between pairs of reachable vertices in G, that is,*

$$\Delta = \max\{d(u, v) : u, v \in V \wedge d(u, v) < \infty\}$$

Definition 4.2 (Ball of radius r around a vertex v). *The Ball of radius r around a vertex u is defined by the set of vertices that are within distance r from u. That is,*

$$B(u, r) = \{v \in V : d(v, u) \leq r\}$$

Definition 4.3 (Doubling dimension of a graph G). *The Doubling dimension of a graph G is the minimum value D such that for every $r > 0$ we have that every ball $B(u, 2r)$ is covered by at most 2^D balls of radius r.*

Lemma 4.1.1. *Given a weighted undirected connected graph $G = (V, E)$ with constant doubling dimension D and diameter Δ, for any constant $\epsilon \in (0, 1)$ we can cover G with at most $k = \lceil (2/\epsilon)^D \rceil$ balls of radius at most $\epsilon\Delta$*

Proof. Since G has diameter Δ, we observe that we can cover G with 1 ball of radius Δ. Given that G has constant doubling dimension D we can cover that one ball with (at most) 2^D balls of radius $\Delta/2$, and in turn we can cover each of these balls of radius $\Delta/2$ with at most 2^D balls of radius $\Delta/4$, that is, we can cover the entire graph with at most 2^{2D} balls of radius $\Delta/4$. Iterating this reasoning i times, we end up covering the graph with 2^{iD} balls of radius $\Delta/2^i$. As a consequence, if we want to cover G with balls of radius $\epsilon\Delta$ with $\epsilon \in (0, 1)$, we need to find $i \in \mathbb{Z}_0^+$ such that $\epsilon\Delta \leq \Delta/2^i$, that is, $i = \lceil -\log_2 \epsilon \rceil$. This means that we need at most $2^{\lceil -\log_2 \epsilon \rceil\, D}$ balls of radius at most $\epsilon\Delta$ to cover the entire graph. By definition of the ceiling function we know that $i < -\log_2 \epsilon + 11$, therefore it holds that

$$\begin{aligned} 2^{\lceil -\log_2 \epsilon \rceil\, D} &\leq 2^{(-\log_2 \epsilon + 1)\, D} \\ &\leq \left\lceil \left(\frac{2}{\epsilon}\right)^D \right\rceil \end{aligned} \tag{4.2}$$

and the thesis follows. □

Observe that Lemma 4.1.1 immediately implies that there exists a set S of k vertices $v_1, \ldots, v_k \in V$ such that for every $u \in V$ we have that $d(u, S) \leq \epsilon\Delta$, where $d(u, S) = \min\{d(u, v_i) : 1 \leq i \leq k\}$ is the distance to the closer center vertex from u.

Lemma 4.1.2. *Given a weighted undirected connected graph $G = (V, E)$ with constant doubling dimension D and diameter Δ. Selecting a set S of k vertices, where $k = \lceil (2/\epsilon)^D \rceil$, computed with the Farthest-First Traversal algorithm. Then,*

$$\max_{u \in V} d(u, S) \leq 2\epsilon\Delta$$

Proof. Given that the FFT algorithm start by selecting an arbitrary vertex v_1 and for $2 \leq i \leq k$ selects the i^{th} vertex v_i as the most distant vertex from $v_1, v_2, \ldots, v_{i-1}$. If we consider $z \in V$ the farthest vertex from the sample set S, that is $z = \text{argmax}_{u \in V}\, d(u, S)$, it is easy to see that the $k + 1$ vertices in $S \cup \{z\}$ are all at distance at least $d(z, S)$ from one another. In fact all the previously selected vertices v_i where chosen because their distances from the sample set $S_{i-1} = \{v_1, \ldots, v_{i-1}\}$ was greater than the distance of vertex z to S_{i-1}. Therefore, considering the k balls $B_1, B_2, \ldots, B_k$ of radius $\epsilon\Delta$ covering the graph G, which must exist because of Lemma 4.1.1, by the pigeonhole principle, at least two of the vertices in $S \cup \{z\}$, say x and y, must belong to the same ball B_i, hence, by the triangle inequality, their distance must be at most twice the radius of the ball. Thus,

$$d(z, S) \leq d(x, y) \leq 2\epsilon\Delta$$

which concludes our proof. □

Definition 4.4 (Mesh graph). *A mesh graph is a finite connected subgraph of the infinite n-dimensional lattice embedded in $\mathbb{Z}^n$, that is, the one in which the vertices are the points $(x_1, x_2, \ldots, x_n) \in \mathbb{Z}^n$ and each edge connects two vertices $u = (u_1, u_2, \ldots, u_n)$ and $v = (v_1, v_2, \ldots v_n)$ if and only if $\sum_{i=1}^{n} |u_i - v_i| = 1$.*

Lemma 4.1.3. *Given a mesh graph $G = (V, E)$ with diameter Δ. Then the sum of distances $W(u)$ of every vertex $u \in V$ is $\Theta(\Delta|V|)$.*

Proof. We will prove the Lemma 4.1.3 only for mesh graphs that could be embedded exactly in a polytope shape, for instance, in a 2-dimensional space such graph lie exactly in a rectangle (polytope in $\mathbb{Z}^2$) whose vertices are the 4 vertices of the mesh graph with the lower degree. We refer to such mesh graph as "rectangular" mesh graph. We start by considering a rectangular mesh graph $G = (V, E)$ embedded in $\mathbb{Z}^2$ such as the one shown in Figure 4.1. By observing that the rectangle that encloses the rectangular mesh graph has dimensions a_1 and a_2. It is clear

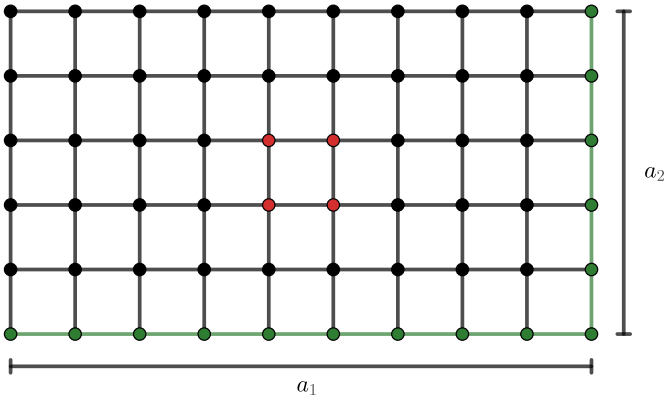

Figure 4.1: A mesh graph G=(V,E) embedded in $\mathbb{Z}^2$

that the its diameter is $\Delta = a_1 + a_2$ and the number of vertices are $|V| = a_1 \cdot a_2$ since by the definition of mesh graph each edge has unit weight.

The sum of distances of every vertex in a graph is trivially $O(\Delta|V|)$ by the triangle inequality. Therefore, to show that the sum of distances of all the vertices in a rectangular mesh graph are $\Theta(\Delta|V|)$ it suffice to prove that the most central vertex, that is, the one with the minimum sum of distances value, is $\Omega(\Delta|V|)$.

In case of a rectangular mesh graph such as in Figure 4.1, we can state that for the sum of distances $W(u)$ of the most central vertex u (which could be any of the highlighted vertices in red in Figure 4.1) holds the following

$$W(u) \geq 4 \sum_{i_1=0}^{\lfloor a_1/2 \rfloor} \sum_{i_2=0}^{\lfloor a_2/2 \rfloor} (i_2 + i_1) \tag{4.3}$$

Lets define $A_i = \lfloor a_i/2 \rfloor$, thus

$$\begin{aligned} W(u) &\geq 2^2 \sum_{i_1=0}^{A_1} \sum_{i_2=0}^{A_2} (i_2 + i_1) = 2^2 \sum_{i_1=0}^{A_1} \left(\sum_{i_2=0}^{A_2} i_2 + \sum_{i_2=0}^{A_2} i_1 \right) = \\ &= 2^2 \sum_{i_1=0}^{A_1} \left(A_2 \frac{(A_2+1)}{2} + A_2 i_1 \right) = 2^2 \left(\sum_{i_1=0}^{A_1} A_2 \frac{(A_2+1)}{2} + \sum_{i_1=0}^{A_1} A_2 i_1 \right) = \\ &= 2^2 \left(A_1 A_2 \frac{(A_2+1)}{2} + A_1 A_2 \frac{(A_1+1)}{2} \right) = \\ &= 2^2 A_1 A_2 \left(\frac{(A_1+1)}{2} + \frac{(A_2+1)}{2} \right) \end{aligned} \tag{4.4}$$

From the above equation, by substituting $A_i = \lfloor a_i/2 \rfloor$ ans by the definition of the floor function we have

$$\begin{aligned} W(u) &\geq 4\Big(\frac{a_1}{2}-1\Big)\Big(\frac{a_2}{2}-1\Big)\Big(\frac{a_1+a_2}{4}\Big) = \\ &= \Big(\frac{a_1+1}{2}-\frac{3}{2}\Big)\Big(\frac{a_2+1}{2}-\frac{3}{2}\Big)(a_1+a_2) \\ &= \Big(\frac{1}{4}(a_1+1)(a_2+1)-\frac{3}{4}(a_1+a_2+1)\Big)(a_1+a_2) \end{aligned} \tag{4.5}$$

By the definitions of a_1 and a_2, that is, $\Delta = a_1 + a_2$ and $|V| = (a_1+1)(a_2+1)$ we have that

$$W(u) \geq \Delta\Big(\frac{1}{4}|V| - \frac{3}{4}(\Delta+1)\Big) \tag{4.6}$$

We want to obtain that $W(u) \geq K\Delta|V|$, for some constant $K > 0$, therefore it must hold that

$$\begin{aligned} \Big(\frac{1}{4}|V| - \frac{3}{4}(\Delta+1)\Big) &\geq K|V| \\ \frac{\Delta+1}{|V|} &\leq \Big(\frac{1}{3} - \frac{4}{3}K\Big) \\ \frac{\Delta+1}{|V|} &< \frac{1}{3} \end{aligned} \tag{4.7}$$

Therefore, for bidimensional rectangular mesh graph for which holds the final inequality of the above equation we have that $W(u) = \Omega(\Delta|V|)$.

Let now generalize the proof for a rectangular mesh graph embedded in a generic n-dimensional infinite lattice, such that it is enclosed in a polytope in $\mathbb{Z}^n$ of dimensions $a_1, a_2, \ldots, a_n$. For such rectangular mesh graph we have that the diameter $\Delta = \sum_{i=1}^{n} a_i$ and the vertices number are $|V| = \prod_{i=1}^{n}(a_i+1)$. Let $A_i = \lfloor a_i/2 \rfloor$ for $1 \leq i \leq n$, thus, by the generalization of the results of Equation 4.4, for the most central vertex holds that

$$\begin{aligned} W(u) &\geq \sum_{i_1=1}^{A_1}\sum_{i_2=1}^{A_2}\cdots\sum_{i_n=1}^{A_n}(i_1+i_2+\cdots+i_n) = \\ &= 2^n\left(\sum_{i=1}^{n}\frac{(A_i+1)}{2}\right)\prod_{i=1}^{n}A_i \end{aligned} \tag{4.8}$$

By substituting $A_i = \lfloor a_i/2 \rfloor$ and by the definition of the floor function we have that

$$\begin{aligned} W(u) &\geq 2^n\left(\frac{1}{4}\sum_{i=1}^{n}a_i\right)\prod_{i=1}^{n}\Big(\frac{a_i}{2}-1\Big) = \\ &= 2^{n-2}\left(\sum_{i=1}^{n}a_i\right)\prod_{i=1}^{n}\Big(\frac{a_i+1}{2}-\frac{3}{2}\Big) \end{aligned} \tag{4.9}$$

this time the result of the multiplication gives a function that does not depend only on the diameter and the number of vertices, but also depend on a function $F(a_1, \ldots, a_n)$ of the dimensions of the polytope that encloses the graph. That is,

$$W(u) \geq 2^{n-2}\Delta\Big(K_1|V| - K_2F(a_1, \ldots, a_n)\Big) \tag{4.10}$$

where K_1 and K_2 are some constant greater than zero. Observe that $K_1|V| - K_2F(a_1, \ldots, a_n) \geq K|V|$, for $K > 0$ if

$$\frac{F(a_1, \ldots, a_n)}{|V|} \leq \frac{K_1}{K_2} - \frac{K}{K_2} < \frac{K_1}{K_2} \tag{4.11}$$

Therefore, if any of the combinations of the polytope dimensions satisfy this last inequality, we will have that $W(u) = \Omega(\Delta|V|)$. □

We can now state the following

Theorem 4.1.4. *Let $G = (V, E)$ be a weighted undirected connected mesh-like graph with constant doubling dimension D and diameter Δ. Using $k = O(\epsilon^{-D})$ samples computed with the Farthest-First Traversal approach, we can estimate the sum of distances $W(u)$ as defined in Algorithm 6 of every vertex $u \in V$ within a relative error of ϵ with k SSSP computations.*

Proof. By Lemma 4.1.2 we know that selecting a sample S with $k = \lceil(2/\xi)^D\rceil = O(\xi^{-D})$ samples in a FFT fashion assures us that for every vertex $u \in V$ it holds that $d(u, S) \leq \max_{z\in V} d(z, S) \leq 2\xi\Delta$. From our Algorithm 6 we know that by keeping track of the closest vertices $v_i \in S$ of every vertex $u \in V$ we can bound the sum of distances $W(u)$ as follows

$$W(v_i) - (|V| - 2)d(u, v_i) \leq W(u) \leq W(v_i) + (|V| - 2)d(u, v_i) \tag{4.12}$$

The estimate of our algorithm are the arithmetic mean of the lower and upper bounds, that is $\hat{W}(u) = W(v_i)$, where v_i is the closest vertex to u among the sample vertices. Therefore, we can rewrite Equation 4.12 as

$$\hat{W}(u) - W(u) \leq (|V| - 2)d(u, v_i) \quad \wedge \quad W(u) - \hat{W}(u) \leq (|V| - 2)d(u, v_i) \tag{4.13}$$

That is,

$$|\hat{W}(u) - W(u)| \leq (|V| - 2)d(u, v_i) \tag{4.14}$$

From Lemma 4.1.2, for a constant $c > 0$, we have that

$$|\hat{W}(u) - W(u)| \leq (|V| - 2)d(u, v_i) \leq 2(|V| - 2)\xi\Delta \tag{4.15}$$

If we divide both sides of the last inequality by $W(u)$ and we consider the mesh-like property of the graph given in Lemma 4.1.3, we have

$$\frac{|\hat{W}(u) - W(u)|}{W(u)} \leq 2\xi \frac{(|V| - 2)\Delta}{W(u)} \leq \frac{2\xi}{c} \tag{4.16}$$

By choosing $\epsilon = 2\xi/c$ our estimates assure assure small relative errors with $O(\epsilon^D)$ SSSP computations for mesh-like input graphs. □

It is not possible to make the same reasoning for the second version of the approach because the sample distance definition, that is $d(u, S_i) = \min_{v \in S_i} \frac{d(u,v)}{W(v)}$, involves also the sum of distances of the vertices in the sample S_i produced after i iterations of the algorithm. Therefore, the guarantees of Lemma 4.1.2 are not valid in this scenario, which means that there are no guarantees that, at each iteration i, for every pair of selected vertices $v, q \in S_i$ and for every vertex $u \in V - S_i$, the distance $d(v, q) \geq d(v, u)$.

Observe that, during the computation of the minimum distance $d(u, S_i)$ of a vertex $u \in V$ from the sample S_i at a generic iteration i, if we consider two vertices $v_h, v_l \in S_i$ such that $d(u, v_l) < d(u, v_h)$ and $W(v_l) < W(v_h)$, we could select as "closer" vertex v_h if the ratio $d(u, v_l)/d(u, v_h)$ is greater that the ratio $W(v_l)/W(v_h)$. Which means that the assignation of the vertices to their closes sample vertex does not behave as the first version. Furthermore, let consider two vertices $u, z \in V$ and two vertices $v_x, v_y \in S_i$. Let assume that $x = \text{argmin}_{1 \leq j \leq i}\, d(u, v_j)/W(v_j)$ and $y = \text{argmin}_{1 \leq j \leq i}\, d(z, v_j)/W(v_j)$, that is, v_x is the vertex that determine $d(u, S_i)$ and v_y is the vertex that determine $d(z, S_i)$. Let assume also that $d(u, v_x) > d(z, v_y)$ and $W(v_x) > W(v_y)$. In this scenario there is no guarantee that the farthest vertex u will be picked because if the ratio $d(u, v_x)/d(z, v_y)$ is lower than the ratio $W(v_x)/W(v_y)$, than vertex z will be selected in the sample.

Even if the coverage of the selected sample of the second version of the algorithm does not have the same guarantees as in the first version, we expect that by weighting with the sum of distances of the sample vertices we reduce the number of selected outlier vertices without affecting too much the coverage quality, thus the estimate relative errors.

4.2 k-means++ and k-median++ Methods

These are two probabilistic variations of the FFT algorithm, already presented in literature and used to seed the initial cluster centers of k-means and k-median clustering algorithms, respectively. While we were experimenting the FFT approach for the estimation of the closeness centrality in social graphs, as the reader may

see later in Chapter 5, the method performed poorly. We attributed the reason of these performances to the fact that social graphs have a high dimensionality, therefore, the sample created by FFT tended to be composed mainly of outlier vertices, i.e. vertices in the periphery of the graph. This composition did not allow good estimates for most of the more central vertices in the graph, that remained farther away from the selected sample.

In order to improve the estimate of the more central vertices it was necessary to take less peripheral vertices. One way to do it was to add a random component in the choice of the next selected vertex. Hence, instead of selecting at each iteration the farther vertex form the sample we thought to use the distance information to assign a selection probability proportional to the distance from the sample, which lead us to the same designs of the k-means++ and k-median++ algorithms as in [AV07].

4.2.1 Algorithms

The algorithms reflect the operations made by FFT, except for the selection of the sample vertices, which is probabilistic and based on the distances to the already selected sample set. The difference between this two probabilistic approaches is given by how they assign the probabilities to the vertices at each iteration. They receive in input a undirected weighted connected graph $G = (V, E)$, a parameter k which determines the maximum sample size, and a parameter ϵ used to determine the target maximum relative error of the estimates. After the initialization step, as in FFT, the first vertex is uniformly selected at random from V. At each iteration $2 \leq j \leq k$ we compute the SSSP form the current vertex v_c to all the other vertices $u \in V$. We update the argument a_u of each vertex by keeping the minimum distance from the sample set, the fork bounds and the relative errors as in FFT. The probability assigned to each vertex $u \in V$ is

$$p_u = \frac{d(u, S_j)}{\sum_{v \in V} d(v, S_j)}$$

for the k-median++ approach, while is

$$p_u = \frac{d(u, S_j)^2}{\sum_{v \in V} d(v, S_j)^2}$$

for the k-means++ approach, where S_j represents the sample set at iteration j.

In Algorithm 7 the minimum distance from a vertex $u \in V$ to the current sample set S_j is kept by the argument a_u. The termination criterion is the same as in FFT, that is, the algorithm stops its iterations when the size of the sample set reaches k or when the maximum relative error is below the target error ϵ.

Algorithm 7 k-median++ and k-means++ Method

procedure KMPP($G = (V, E)$, k, ϵ)
  `// Initialize arguments` a_u`, relative errors` re_u`,`
  `// lower bounds` lb_u `and upper bounds` ub_u
  for all $u \in V$ **do**
    $a_u \leftarrow +\infty$; $re_u \leftarrow +\infty$; $lb_u \leftarrow 0$; $ub_u \leftarrow +\infty$;
  end for
  $S \leftarrow \emptyset$
  Assign to v_n a vertex from V selected uniformly at random
  repeat
    $v_c \leftarrow v_n$ `// Set the next vertex` v_n `to be the current one`
    $S \leftarrow S \cup \{v_c\}$
    Compute SSSP distances $d(v_c, u)$ from v_c to all other vertices $u \in V$
    $W(v_c) \leftarrow \sum_u d(v_c, u)$
    `// Update argument, lower and upper bounds of` v_c
    $a_{v_c} \leftarrow 0$; $lb_{v_c} \leftarrow W(v_c)$; $ub_{v_c} \leftarrow W(v_c)$;
    for all $u \in V - S$ **do**
      $a_u \leftarrow \min\{a_u, d(v_c, u)\}$
      $lb_u \leftarrow \min\{lb_u, W(v_c) - (|V| - 2)\, d(v_c, u)\}$
      $ub_u \leftarrow \max\{ub_u, W(v_c) + (|V| - 2)\, d(v_c, u)\}$
      $re_u \leftarrow (ub_u - lb_u)/(2 \max\{lb_u, d(v_c, u)\})$
    end for
    $re_{max} \leftarrow \max_{u \in V}\{re_u\}$ `// Find maximum relative error` re_{max}
    Select next vertex v_n based on probability

$$p_u = \begin{cases} a_u / (\sum_{v \in V} a_v) & \text{if k-median++} \\ a_u^2 / (\sum_{v \in V} a_v^2) & \text{if k-means++} \end{cases}$$

  until TERMCRITERIA($k, \epsilon, |S|, re_{max}$)
  return $(\hat{W}(u) = (ub_u + lb_u)/2)$ for each $u \in V$
end procedure

4.2.2 Analysis

We do not provide a formal analysis of the sample size needed to achieve a given relative error for these two approaches but we provide the following intuitive explanation. The reasoning is the same made for the FFT. Given an undirected weighted connected graph with constant doubling dimension D and diameter Δ, we want to argue that the k sample vertices are spread in 2^{iD} balls of radius $\epsilon\Delta$ where $\epsilon = 1/2^i$. For the FFT we had the certainty that at every step the farthest vertex was selected to be part of the sample, therefore we could use the pigeonhole principle to argue that the sample set S of size $k = \lceil 2/\epsilon)^D \rceil$ whose vertices are mutually farther than $\max_{v \in V} d(v, S)$, together with the farthest vertex from the sample will cause two distant vertices to be within the same ball of radius $\epsilon\Delta$. This time we are assigning probabilities to the vertices thus it is not possible to argue the same as in FFT. But we can argue that, since we are assigning probabilities based on the distance to a vertex from the sample, we may end up covering the same balls of radius $\epsilon\Delta$ with a linear increase in the size of the sample and at the same time avoiding only picking outlier vertices.

4.3 Progressive CCK Methods

In this section we present two adaptations of Cck approach that instead of separating the sampling phase from the estimation phase, combine them together by progressively selecting new vertices to the sample while adding their contribution to the sum of distances estimates of every vertex of the input graph. Furthermore, instead of using just the information of the base set to build the sampling coefficients, we use the distances information due to every SSSP of the sampled vertices to update them.

We can argue that these two methods produce estimates with similar estimates guarantees with respect to Cc presented in Section 3.2. That is, for every vertex u of the input graph $G = (V, E)$, we can compute the sum of distances estimate $\hat{W}(u)$ within a small relative error ϵ with high probability using $O\left(\frac{\log |V|}{\epsilon^2}\right)$ SSSP regardless of the graph topology.

4.3.1 Algorithms

The two adaptations share the same operations except for the selection of the next vertices to be added to the sample, therefore we are going to treat them as variants of a single approach, which is described in Algorithm 8.

The algorithm receives in input a weighted undirected connected graph $G = (V, E)$, a parameter k, that determines the final sample size, and a base set S_0. It

Algorithm 8 Progressive Cc Algorithm

procedure ProgCcK($G = (V, E), k, S_0$)
  // Compute sampling coefficients γ_u
  for all $u \in V$ **do**
    $\gamma_u \leftarrow \frac{1}{|V|}$
  end for
  for all $v \in S_0$ **do**
    Compute SSSP distances $d(v, u)$ from v to all other vertices $u \in V$
    $W(v) \leftarrow \sum_u d(v, u)$
    for all $u \in V$ **do**
      $\gamma_u \leftarrow \max\left\{\gamma_u, \frac{d(v,u)}{W(v)}\right\}$
    end for
  end for
  // Initialize the estimates of each vertex z
  for all $z \in V$ **do**
    $\hat{W}(z) \leftarrow 0$
  end for
  $S \leftarrow \emptyset$
  $i \leftarrow 0$ // Counts the number of iterations
  repeat
    // Select next vertices based on every γ_u
    $Q \leftarrow$ VertexSelection$(V - S, \gamma_u)$
    for all $q \in Q$ **do**
      $S \leftarrow S \cup \{q\}$
      Compute SSSP distances $d(q, z)$ from q to all other vertices $z \in V$
      $\hat{W}(q) \leftarrow \sum_{z \in V} d(q, z)$
      for all $z \in V - S$ **do**
        $\hat{W}(z) \leftarrow \hat{W}(z) + d(q, z)/\gamma_q$
        $\gamma_z \leftarrow \max\left\{\gamma_z, \frac{d(q,z)}{\hat{W}(q)}\right\}$
      end for
    end for
    $i \leftarrow i + 1$
  until $|S| \geq k$
  for all $z \in V - S$ **do**
    $\hat{W}(z) \leftarrow \hat{W}(z)/i$
  end for
  return $(z, \hat{W}(z))$ for $z \in V$
end procedure

computes, as in Algorithm 4, the sampling coefficients γ_u using the base set S_0 by performing a SSSP from each vertex $v \in S_0$ to all the other vertices $u \in V$ such that $\gamma_u = \max_{v \in S_0} d(v,u)/W(v)$.

It then initializes the sum of distances estimates $\hat{W}(z)$ of each vertex $z \in V$, the sample set S and an iteration counter.

At each iteration the algorithm selects, based on the sampling coefficients γ_u, a set Q of vertices, that is computed differently depending on the adaptation. For each vertex $q \in Q$ we compute the following operations. Firstly we add q to the sample set S, then we compute the SSSP from q to all the other vertices $z \in V$. Moreover, using the distances information, we add to each sum of distances estimate $\hat{W}(z)$ the intermediate contribution $d(q,z)/\gamma_q$ of q which must be normalized by the total number of iterations. And lastly we update the sampling coefficients γ_z by keeping the maximum between γ_z and $d(q,z)/W(q)$, as if q were part of the base set S_0. Finally we increment the counter i and we end our iteration.

We stop iterating when the sample set size is greater or equal than k (it can be strictly greater only in one of the two adaptations).

The final contribution of each vertex $v \in S$ to the estimates $\hat{W}(u)$ for every $u \in V$ are given by $d(v,u)/(x\gamma_v)$ where x is the final number of iterations in Algorithm 8. The algorithm divides the intermediate contributions of each estimate by the number of the final iterations x after having computed the sample and not during the computation mainly for two reasons. Firstly, at each iteration i, it is always possible to compute the sum of distances estimates by dividing the accumulated values for each estimate by i. Secondly, in one of the two adaptations the final number of iterations is unkown a priori.

By making a comparison with Cck approach, the reader may notice that the contribution of a sample vertex q to the sum of distances estimate $\hat{W}(u)$ in Cc is given by $d(q,u)/p_q = d(q,u)/\min\{1, k\gamma_q\}$. In these progressive adaptations we replace the parameter k with the number of iterations x and we do not take the minimum. In the next section we will argue that, under certain conditions, these two adaptations may have similar estimates guarantees as Cck approach.

At every iteration, the sample vertex selection could be made by exactly extracting one vertex from $V - S$ or by Poisson sampling the vertices from $V - S$. The former case is presented in Algorithm 9, which is a possible implementation for the random extraction of one vertex based on the probabilities γ_u. In this adaptation, the final number of iterations x is going to be equal to the input parameter k of the algorithm.

The latter case is presented in Algorithm 10 which implements a Poisson sampling also based on the probabilities γ_u. In this case the number of iterations performed by the algorithm may be different from the input parameter k because

Algorithm 9 One sample selection

```
1: procedure VertexSelection(U, γ_u)
2:     Γ ← Σ_{u∈U} γ_u
3:     // Generate uniformly at random a number x ∈ [0, Γ]
4:     x ← RandomNumber(0, Γ)
5:     // Consider u_i ∈ U such that 1 ≤ i ≤ |U|
6:     i ← 0
7:     while x < Γ do
8:         x ← x + γ_{u_i}
9:         i ← i + 1
10:    end while
11:    return {u_i}
12: end procedure
```

at each step it selects $O(1)$ vertices to be added to the sample.

Algorithm 10 Poisson Sample selection procedure

```
1: procedure VertexSelection(U, γ_u)
2:     Q ← ∅
3:     for all u ∈ U do
4:         if RandomNumber(0, 1) < γ_u then
5:             Q ← Q ∪ {u}
6:         end if
7:     end for
8:     return Q
9: end procedure
```

As we may see in the following subsection, making a Poisson sampling based on the probabilities γ_u assures us to get $O(1)$ samples which does not affect the time complexity of the algorithm, but since is not exactly one we may end up with a sample of size bigger than k.

4.3.2 Analysis

The goal of the analysis is to show that, given an input graph $G = (V, E)$, both the progressive Cc approaches are able to produce the sum of distances estimates $\hat{W}(u)$ for every vertex $u \in V$ within a small relative error ϵ with high probability using $O(\frac{\log |V|}{\epsilon^2})$ SSSP distance computations. We mainly based this proof on the results obtained by Chechik *et al.* in [CCK15].

In Algorithm 8 the sampling coefficients are built the same way as in Cc before the sample construction. Therefore all the lemmas related to the sampling coefficients and the well positioned vertices, defined and proved in Subsection 3.2.2, holds for our two adaptations. Hence, the base set S_0 must have $O(\log |V|)$ vertices to obtain with high probability the sampling coefficients that approximate a universal Probability Proportional to Size. However, in these progressive approaches, the sampling coefficients are updated each time a new vertex is added to the sample set, hence they may increase their value depending to the iteration i and the sample vertices selected within such iterations. We refer to the sampling coefficient of each vertex $u \in V$ at iteration i as $\gamma_u^{(i)}$.

From the specification of Algorithm 8 it holds that $\gamma_u^{(i+1)} \geq \gamma_u^{(i)}$ for each $u \in V$. Before the iteration process, the sampling coefficients obtained from the base set computations are $\gamma_u^{(0)}$ and are used in the first VertexSelection of the algorithm during the first iteration. The final contribution of a sample vertex $v \in S$ to the sum of distances estimate $\hat{W}(u)$ is $d(v,u)/(x\gamma_u^{(i)})$ where x is the final number of iterations and i represent the iteration in which vertex v has been selected.

Let us focus on the Poisson sampling version of the progressive Cc adaptation. For each vertex $u \in V$, the probability of being selected as sample vertex is the following

$$p_u = 1 - \prod_{i=0}^{x-1}(1 - \gamma_u^{(i)}) \tag{4.17}$$

That is, 1 minus the probability of not being selected in any of the x iterations.

Let assume that we do not update the sampling coefficients at each iteration, therefore $p_u = 1 - (1 - \gamma_u^{(0)})^x$ for any $u \in V$. By the binomial expansion we have that

$$\begin{aligned} 1 - (1 - \gamma_u^{(0)}) &= 1 - \sum_{j=0}^{x} \binom{x}{j} 1^{x-j} \Big(- \gamma_u^{(0)} \Big)^j \\ &= x\gamma_u^{(0)} + \frac{x(x-1)}{2!} \Big(- \gamma_u^{(0)} \Big)^2 + \frac{x(x-1)(x-2)}{3!} \Big(- \gamma_u^{(0)} \Big)^3 + \dots \\ &\quad \cdots + \Big(- \gamma_u^{(0)} \Big)^x \end{aligned} \tag{4.18}$$

For sufficiently small values of $\gamma_u^{(0)}$ we can argue that the probability p_u is almost $x\gamma_u^{(0)}$, since the other therms of the equation can be neglected. If we assume that $\max_{u,v \in V} d(v,u)/W(v) \leq 1/x$ then the probabilities p_u are also equal to $\min\{1, x\gamma_u^{(0)}\}$. Therefore, each vertex u is selected in the sample with probability p_u and its contribution to the sum of distances of every other vertex $u \in V$ is $d(v,u)/x\gamma_u^{(0)}$. Hence we can apply Lemma 3.2.8 using $k = x$ and obtain the same

guarantees as in Cc.

The same reasoning could be made for any updated $\gamma_u^{(i)}$ throughout the iteration process. By updating the sampling coefficients we expect an improvement of the estimates relative errors since such update better refines the approximation of the universal Probability Proportional to Size.

Let us consider the one sample version of the progressive Cc adaptation. This time, for each vertex $u \in V$, the probability p_u of being selected involves the sum of the sampling coefficients at iteration i, which is $\Gamma^{(i)} = \sum_{u \in V - S^{(i)}} \gamma_u^{(i)}$, where $S^{(i)}$ is the set of the selected vertices after i iterations. Therefore,

$$p_u = 1 - \prod_{i=0}^{x-1} \left(1 - \frac{\gamma_u^{(i)}}{\Gamma^{(i)}}\right) \tag{4.19}$$

Let assume for now that we do not update the sampling coefficients, therefore $\gamma_u^{(i)} = \gamma_u(0)$ for every i and every $u \in V$.

Let us focus on the sum of the sampling coefficients $\Gamma^{(i)}$ for every iteration i. As in Corollary 3.2.7, if we consider $\bar{\gamma}_u = \max_{z \in V} \frac{d(z,u)}{W(z)}$, we can state that $\Gamma^{(i)} \leq \sum_{u \in V} \bar{\gamma}_u = \Gamma_{\max}$ for every iteration i. Furthermore, recalling the same corollary, $\Gamma_{\max}$ is $O(1)$, thus a constant.

We can argue that the minimum value that $\Gamma^{(i)}$ may have is during the final iteration of Algorithm 8 if all the vertices with the highest sampling coefficients have been selected. In this edge case we have that $\Gamma^{(i)} \geq \Gamma^{(i)} - \sum_{v \in S_{\max}} \gamma_v$, where $S_{\max}$ contains x vertices $v \in V$ that have the highest sampling coefficients $\gamma_v^{(0)}$ compared to the others. Therefore, given the number of iterations x, we can determine a $\Gamma_{\min} > 0$ if $x < |V|$.

Given the number of iterations $x < |V|$, we can determine a constant $\Gamma_{\min} > 0$ such that $\Gamma^{(i)} \geq \Gamma_{\min}$ for every iteration i. If we consider that during the run of Algorithm 8 we iteratively select x vertices with the highest sampling coefficients $\gamma_u^{(0)}$ among all the others, we have that $\Gamma^{(i)} \geq \Gamma^{(i)} - \sum_{v \in S_{\max}} \gamma_v$, where $S_{\max}$ contains such x vertices.

Since we are not considering the update of the sampling coefficients, we can now bound the probability p_u of every vertex $u \in V$ as follow

$$1 - \left(1 - \frac{\gamma_u^{(0)}}{\Gamma_{\min}}\right)^x \leq p_u \leq 1 - \left(1 - \frac{\gamma_u^{(0)}}{\Gamma_{\min}}\right)^x \tag{4.20}$$

By the binomial expansion and if we assume that $\gamma_u^{(0)}$ is sufficiently small we can argue that, for each node $u \in V$, we have

$$x \frac{\gamma_u^{(0)}}{\Gamma_{\max}} \leq p_u \leq x \frac{\gamma_u^{(0)}}{\Gamma_{\min}} \tag{4.21}$$

Therefore, the probability p_u is $\Theta(x\gamma_u^{(0)})$ for each node $u \in V$.

Even if the probabilities are not exactly the weight of the contribution of a vertex, we expect that Lemma 3.2.8 holds also for this method, and we argue that updating the sampling coefficients using the distances information of the SSSP of the sample vertices may only improve the approximation of the universal Probability Proportional to Size, thus improve the quality of the estimates.

Chapter 5

Experiments and Results

In this chapter we describe how the experiments have been conducted and we present our findings on the closeness estimation quality between the different approaches presented throughout the course of this work. In the first section we present the tools and the data sets we used. In the second section we describe how we have structured the runs of each approach and how the performance evaluation has been made. In the third section we present how we conducted the experiments. Lastly, in the final section, we show the results by comparing the different approaches.

5.1 Overview

The experiments we have conducted aim at assessing whether our novel methods, presented in Chapter 4, are competitive with the state-of-the-art algorithms, presented in Chapter 3. We based the comparison on two different graph types: road networks and social graphs. The former have a mesh-like structure while the latter are scale-free graphs that presents the small-world phenomenon.

We are going to present the results based on two road networks of the 9th DIMACS Challenge Dataset [DGJ05] and two social graphs of the SNAP Dataset [LK14], showed in Table 5.1. During our experiments we used two additional graphs, one from each data set, to test the sensitivity of Cck algorithm, which are shown in Table 5.2. We defined the base set size, based on a parameter, over two different graphs in order to keep it unbiased from the benchmark graphs, and achieve a fair final comparison with the other methods.

The methods we compare are shown in Table 5.3 and have been implemented sequentially in C++ using the primitives of the Boost Graph Library [SLL02] to store the input graphs in main memory and to compute the SSSP distances re-

Graph	Vertices	Edges	Weighted
USA-road-d.COL	435 666	1 057 066	yes
USA-road.d-FLA	1 070 376	2 712 798	yes
com-dblp.ungraph	317 080	1 049 866	no
com-youtube.ungraph	1 134 890	2 987 624	no

Table 5.1: Benchmark graphs used for the comparison

Graph	Vertices	Edges	Weighted
USA-road-d.BAY	321 270	800 172	yes
com-amazon.ungraph	334 863	925 872	no

Table 5.2: Benchmark graphs used to test the sensitivity of the base set size in Cck approach

quired. We have used the `boost::minstd_rand` random number generator of the Boost Library to implement the random selections involved within the probabilistic approaches implementations.

We also designed, implemented and tested other approaches and variations whose results are not reported in order to allow a lighter comparison of the most promising methods. Each omitted method performed similarly or poorly than the ones present in Table 5.3.

5.2 Experiment Runs and Evaluation Technique

Each implemented method receives in input a connected undirected graph G in the formats specified by either SNAP or DIMACS and a parameter k that determines the SSSP computations. Furthermore, the probabilistic methods have an additional input parameter used as the seed for the random number generator. A *run* of an algorithm is univocally determined by its input graph G and its parameter k. The probabilistic approaches may have multiple *executions* of the same run by changing the input seed, while the deterministic approaches have only one execution per run. For each execution of an algorithm we stored the sum of distances estimates of every vertex in the input graph and some other useful information, such as the input parameter k, the number of SSSP computations, the wall clock time of the entire execution and the wall clock time spent on computing the SSSP.

To be able to determine exactly the quality of our results we should have calculated the sum of the distances of each vertex for all the benchmark graph, that is, we should have computed the all-pairs shortest-paths of every benchmark graph. Since we did not have such computational power to do it in a feasible

Name	Description
CC	Cck approach.
RAND	Eppstein and Wang approach.
FFTv1	Farthest-First Traversal approach where NewArgument$(u, v_c) = d(u, v_c)$ for each $u \in V$.
FFTv2	Farthest-First Traversal appraoch where NewArgument$(u, v_c) = d(u, v_c)/W(v_c)$ for each $u \in V$.
KM++	k-median++ approach.
PCCos	Progressive Cck appraoch that selects one sample at each iteration.
PCCps	Progressive Cck approach that selects nodes with a Poisson sampling at each iteration.

Table 5.3: Methods

amount of time, we opted for alternative solutions. We chose to compute the exact sum of distances of small subsets on wisely selected vertices of the input graphs. We are going to refer to these subsets of vertices by the name of *ground sets*.

We used three different kinds of ground sets that could give insights on the solution quality form different perspectives. The first ground set we used is composed by 1 000 vertices uniformly selected at random from the input graph, which we refer to as the *random ground set*. The second and third ground sets are build by pooling respectively the most central and least central vertices based on the sum of distances estimates computed form all the runs we have made throughout the experimentation process on the same input graph. More precisely, for each graph, we selected the top 100 vertices with lower sum of distances estimate (higher centrality) from each of the run results and we unified them to form the *top ground set*. Similarly, we selected the bottom 500 vertices with lower sum of distances estimate for each of the run results, and we unified them to form the *bottom ground set*. We considered fewer vertices in the pooling process of the top vertices in order to keep the sizes of the top and the bottom ground sets similar. The random ground set gives an insight on the overall quality of the estimates, while the top and bottom ground sets respectively give insights on the quality of the estimates for the most central and least central vertices of the input graphs.

Given the estimates results of a run and one of the three ground set, we are able

to compute the relative errors for the corresponding ground set vertices. Instead of computing a measure to summarize the overall quality of the solution we decided to plot these relative errors in a normalized cumulative histogram, where on the x-axis we have the relative error and on the y-axis we count the percentage of vertices that have a relative error that is less or equal than the corresponding x-value. For illustrative purposes, the reader may find the results of a single execution in Figure 5.1. Each point corresponds to a bin of vertices whose percentage is determined by its y value and that have relative error less or equal than its x component value . From the figure it is clear that in this execution half of the ground sets vertices

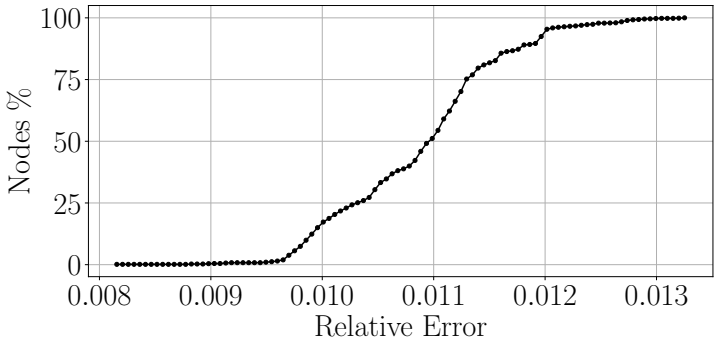

Figure 5.1: Example plot of a single experiment run results based on a ground set

have a relative error below 0.011 and the majority of the vertices have a relative error below 0.012.

5.3 Experimentation Workflow

When we first tested Cck approach, we observe an high variance in the relative error of the estimates among different runs with the same configuration of parameters. We thought that the reason could have been attributed to the base set size, which should be composed by $O(\log |V|)$ vertices. We tested its sensitivity by considering the base set size to be $\alpha \ln |V|$ and by executing different runs with fixed SSSP using different values of $\alpha \in \{0.33, 0.66, 1, 1.33, 1.66, 2\}$. We are not going to present the results to avoid showing unnecessary information, but $\alpha = 1.33$ has given the best performances in terms of variability and we have kept such value through the entire experimentation process. In general, the variability produced by each tested α showed similar results with no sensible reduction to the overall variance.

Another issue with Cck method has been its variability of the number of SSSP computed within the different executions of a run. In fact the sample set,

which determines the number of SSSP, is made out of a Poisson sampling, which produces a sample of a variable size that depends on the input parameter k and the random seed. This factor have shaped our experiment workflow as we are going to explain in a moment.

Another algorithm that has some variability on the sample size is the PCCps. But instead of the traditional Cck approach, its sample variability has been negligible. In fact given the input parameter k that determines the number of SSSP the experiments have shown that its final SSSP computations have exceeded such input parameter by at most one percent of its value.

The objective was to run CC and the other approaches using a comparable number of SSSP computations for every run. Given a benchmark graph, the experiment process starts by the executions of Cck approach with five different values for the parameter k, that is, $k \in \{100, 200, 300, 400, 500\}$. As already mentioned in the previous section, the probabilistic approaches requires multiple executions for the same run, each execution with a different seed. For each k and for every probabilistic approach, we chose to execute 15 times the same run changing the seed. Using the side information we stored, we have computed for Cck approach the average SSSP actual computations among the 15 executions for each of the five runs determined by k. We then used the five averages values of each run as input parameters for the five runs of the other methods. For instance, assuming that after executing 15 times CC with $k = 100$ we obtain that on average CC has used 130 SSSP computations, then all the other algorithms will compute their estimates using $k = 130$, that is, using 130 SSSP for one of their runs. Therefore, after having completed all the CC's executions we have used its 5 averages SSSP number as input parameters to execute the other methods, with the same procedure as in CC.

Another possible experiment procedure, at least for the comparison of the probabilistic approaches, might have been that, for each run of the probabilistic methods, instead of using the average of the SSSP of the corresponding run of CC, we could have made the 15 executions using as input parameter all the 15 values of SSSP computation made in the run of CC. We chose to use the averages instead of the exact values of SSSP of the run of CC to avoid introducing variance in algorithms that are able to compute exactly the specified number of SSSP in input.

For each run of a probabilistic approach we have 15 different sum of distances estimates for each vertex of the input graph, one for each execution, while for the deterministic approaches we stored just one estimate per vertex.

In order to apply the pooling technique to build the ground sets, in case of probabilistic approaches we based our selection on the averages of the estimates

among the 15 executions of each run. For the deterministic approaches, which are specifically the Farthest-First Traversal methods, we have found that the vertices with lower sum of distances estimate had all the same value, so we chose to select for the pooling of the top ground set 100 vertices uniformly at random from this portion of the most central vertices.

We have repeated these experiments for every benchmark graph. In the next section we are going to show the experiment results.

5.4 Results

In this section we are going to show a selection of the results that we obtained in order to focus the reader's attention to the most interesting insights. Therefore for each graph and for each tested algorithm, we are going to show only two of the five runs we made (one per each value of SSSP). We structure the results as follows, first we compare similar methods in order to determine the most promising ones and then we make a final comparison on these most promising approaches.

The first comparison is between the Farthest-First Traversal Methods and the k-median++ algorithm. Figures 5.2 and 5.3 plot the relative errors of the estimates returned by the algorithms on road networks and social graphs, respectively, on the random ground set. For each graph, the subplots are arranged in such a way that each row corresponds to a different algorithm and each column represents a run with a different number of SSSP computations. The two deterministic approaches, FFTv1 and FFTv2, perform similarly but the second version seems to give slightly better estimates in almost all the experiments. The KM++ does not improve the solution quality in general but it is able to assign good relative errors on almost all the vertices of the random ground sets in case of road networks. In fact the cumulative histogram curves of the relative errors increase rapidly and covers almost all the vertices way earlier than the deterministic approaches. Furthermore, we want to point out that the k-median++ algorithm, although less performing in approximating all the sum of distances on the random ground sets, has shown a slight improvement in the estimates of the top ground sets, while it has greatly worsened the estimates of the bottom ground sets. The deterministic approaches instead have behaved slightly worst in the top ground sets and slightly better in the bottom ground sets. Another interesting point that could be attributed to the k-median++ is its low variability on the solution quality, in fact all the executions of the runs are difficult to distinguish in all the plots. Although these considerations, from the results obtained by the experiments, we consider the FFTv2 to be the best among these three approaches.

The second comparison is between the two Progressive Cck approaches. Fig-

ures 5.4 and 5.5 plot the relative errors of the estimates returned by the algorithms on road network and social graphs, respectively, on the random ground set. The subplots are arranged as in the first comparison where we have algorithms in the rows and SSSP computations in the columns. It is clearly visible that the Progressive Cck approach that selects one sample at each iteration (PCCos) performs better with respect to its Poisson sampling (PCCps) counterpart for any graph type. The PCCos has way less variability on the different executions of the runs, and it gives estimates with smaller relative errors, Therefore it is the most promising method between the two.

The third comparison we present is between the state-of-the-art approaches (CC and RAND) and the most promising approaches we designed (PCCos and FFTv2). We present plots, one for each graph, where we show the performances on each of the three ground sets. Figures 5.6 and 5.7 show the plots relative to the road networks, while Figures 5.8 and 5.9 show the plots relative to the social graphs.

Focusing our attention on the FFTv2 method we can clearly see that its estimates relative errors are not competitive in social graphs. But on road network it could compete with CC approach which has more variability on its results with respect to the other two probabilistic approaches. The fact that the Farthest-First Traversal approach finds in a deterministic procedure its solution estimates gives a good competitive advantage on the comparison with Chechik. Furthermore, always focusing on the road networks results, the competitiveness of FFTv2 is kept also in the top ground sets. As already mentioned in this chapter, the Farthest-First Traversal approaches tend to assign to the top vertices the sum of distances of the most central sampled vertex, leading to an homogeneous estimates list of the most central vertices. Therefore the FFTv2 could be used to determine the most central vertices but it is not valuable to achieve a ranking of the most central vertices.

Looking at the RAND algorithm it is interesting to notice that it has great performances in both road and social graphs, even better than Cck approach. Indeed, the relative errors of its estimates present less variability between the runs and are slightly better than Cck approach. From the analysis we have made for the RAND approach we would have expected good results on both road and social graphs but their quality has been surprisingly better than what we expected.

The PCCos seems to be the most competitive together with the RAND approach. In case of road networks, it seems to have a slightly less variability on the relative errors of the estimates based on the top ground set vertices with respect to RAND. Meanwhile, even if the scale of the plots in Figures 5.8 and 5.9 do not consent to compare properly these two most promising approaches, it performed

slightly worse than RAND on these two social graphs. The key strength of PCCos with respect to CC is its less variability on the solutions relative errors, probably due to the absence of the Poisson sampling procedure.

Summarizing the findings of our results, the best methods for the closeness centrality estimation for the two types of graphs we analyzed are PCCos and RAND, followed by CC that is damaged by its more variability on its solutions relative errors. In case of road networks, The Farthest-First Traversal approaches are competitive and could be a valuable option for the closeness centrality estimation. We should say that Cck algorithm has an implementation that does not use the Poisson sampling that may reduce its variability. However, since the procedure requires a VarOpt sampling, which is a reservoir sampling [Vit85] with weighted probabilities, that introduces complexity to the algorithm, we did not tested this version, which could be the subject of future research.

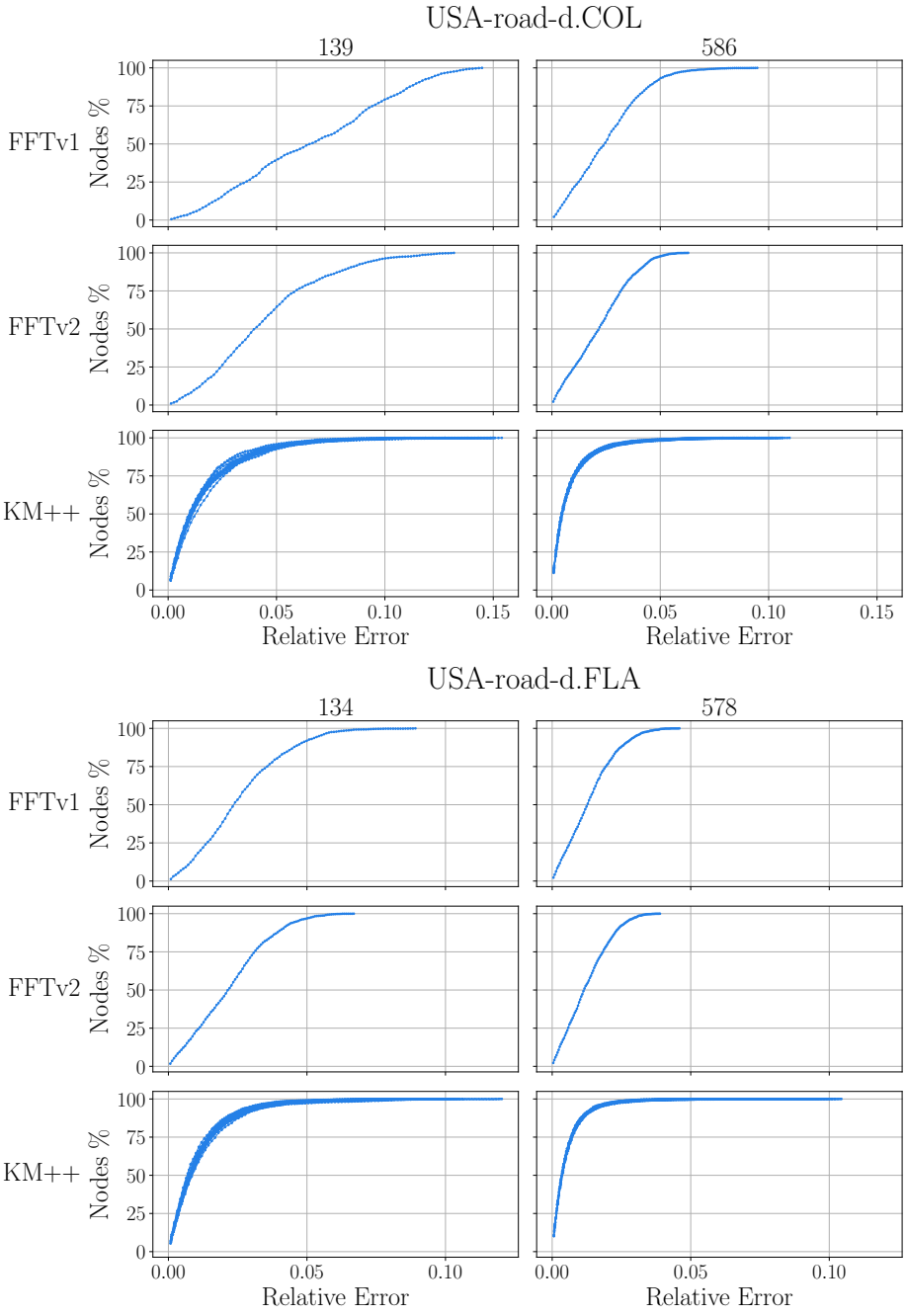

Figure 5.2: Comparison on road networks between Farthest-First Traversals (FFTv1, FFTv2) and the k-median++ (KM++) approaches. The numbers above each column represent the SSSP computations of a run. They are the average among the SSSP computations of each execution of the run of CC, respectively with $k = 100$ and $k = 400$ as input parameter.

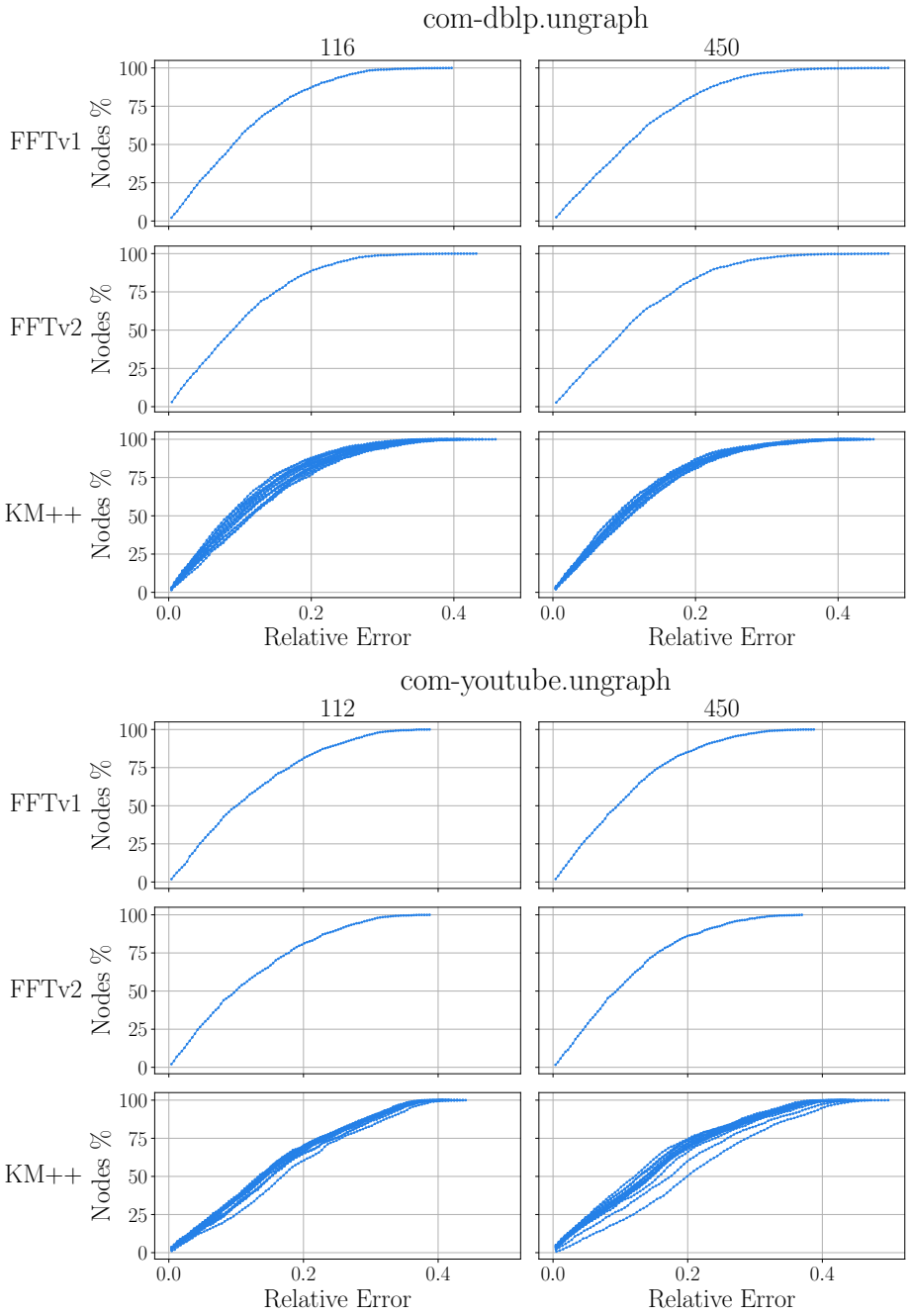

Figure 5.3: Comparison on social graphs between Farthest-First Traversals (FFTv1, FFTv2) and the k-median++ (KM++) approaches. The numbers above each column represent the SSSP computations of a run. They are the average among the SSSP computations of each execution of the run of CC, respectively with $k = 100$ and $k = 400$ as input parameter.

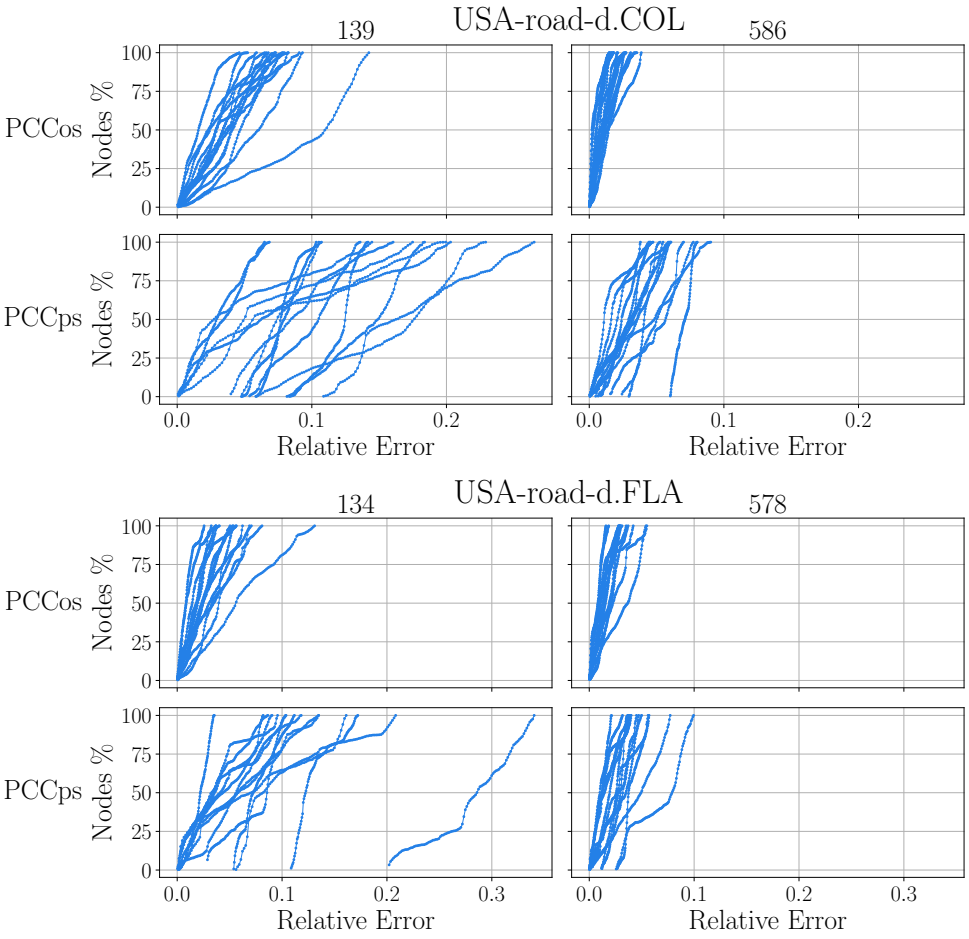

Figure 5.4: Comparison on road networks between the Progressive Cc (PCCos, PCCps) approaches. The numbers above each column represent the SSSP computations of a run. They are the average among the SSSP computations of each execution of the run of CC, respectively with $k = 100$ and $k = 400$ as input parameter.

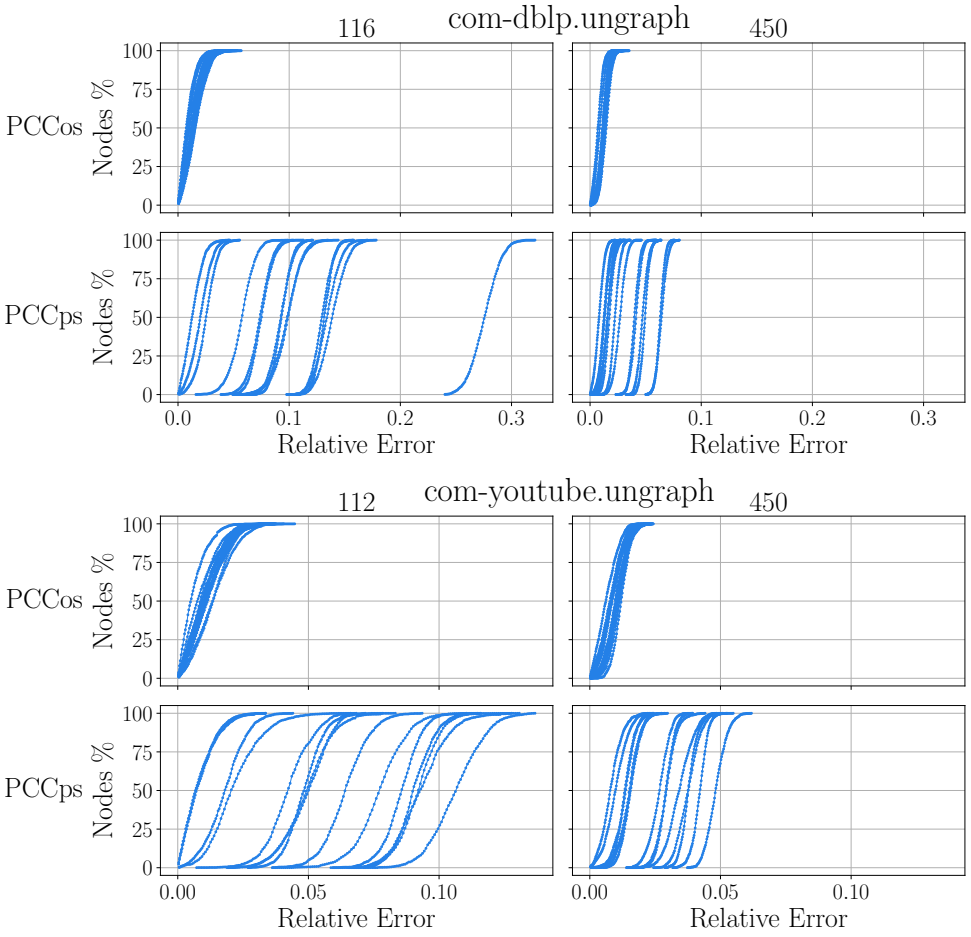

Figure 5.5: Comparison on social graphs between the Progressive Cc (PCCos, PCCps) approaches. The numbers above each column represent the SSSP computations of a run. They are the average among the SSSP computations of each execution of the run of CC, respectively with $k = 100$ and $k = 400$ as input parameter.

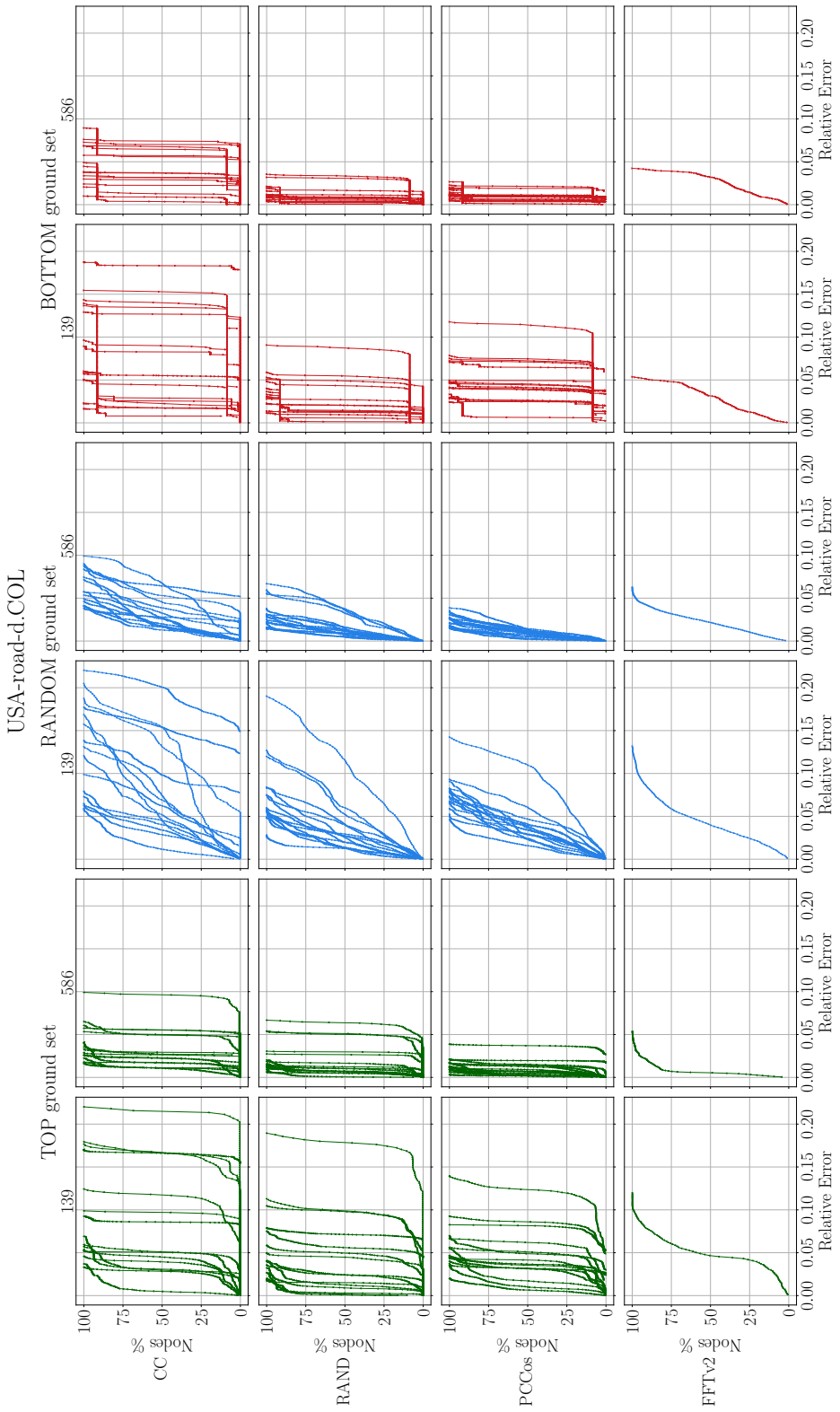

Figure 5.6: Best methods comparison on USA-road-d.COL road network. The numbers above each column represent the SSSP computations of a run. They are the average among the SSSP computations of each execution of the run of CC, respectively with $k = 100$ and $k = 400$ as input parameter.

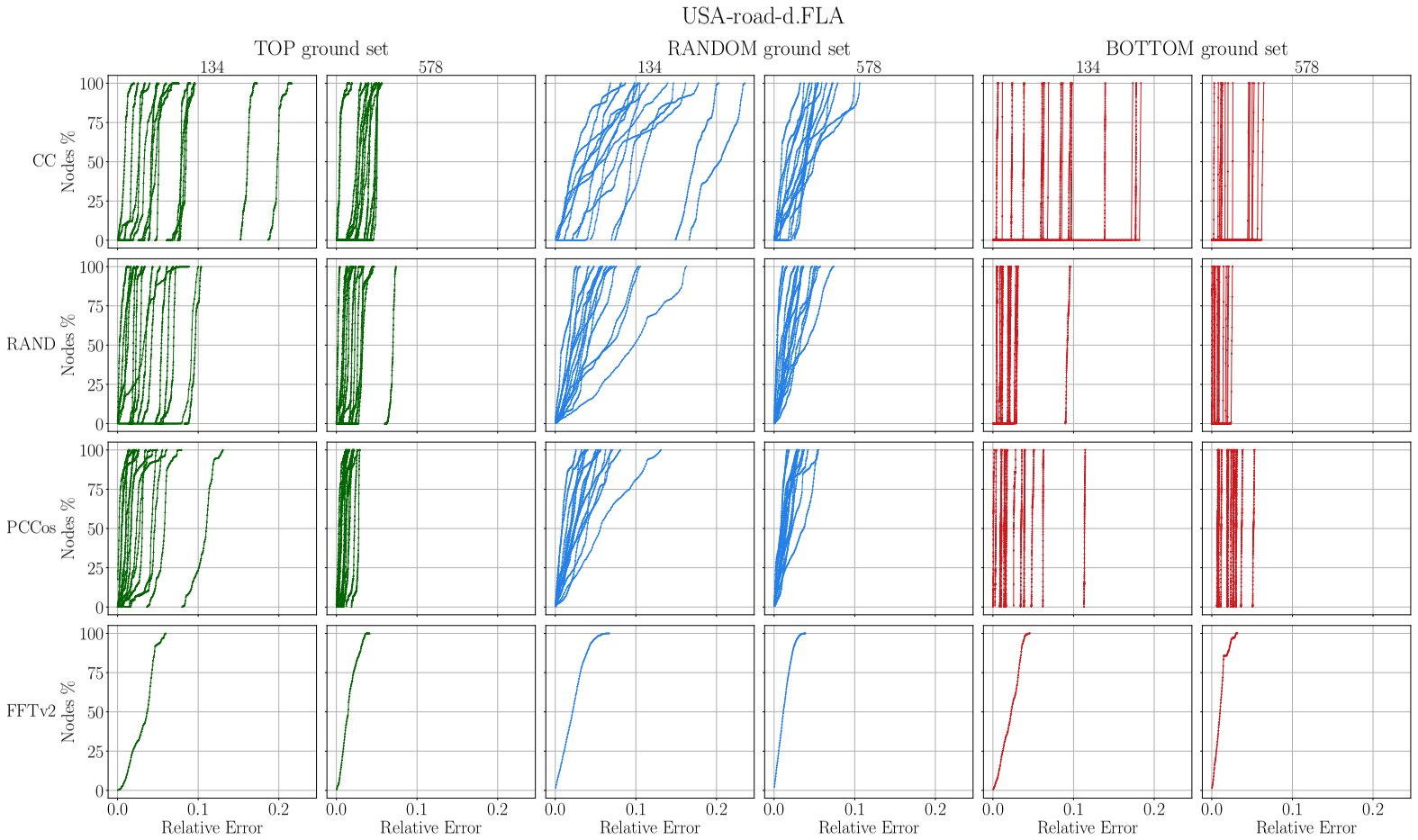

Figure 5.7: Best methods comparison on USA-road-d.FLA road network. The numbers above each column represent the SSSP computations of a run. They are the average among the SSSP computations of each execution of the run of CC, respectively with $k = 100$ and $k = 400$ as input parameter.

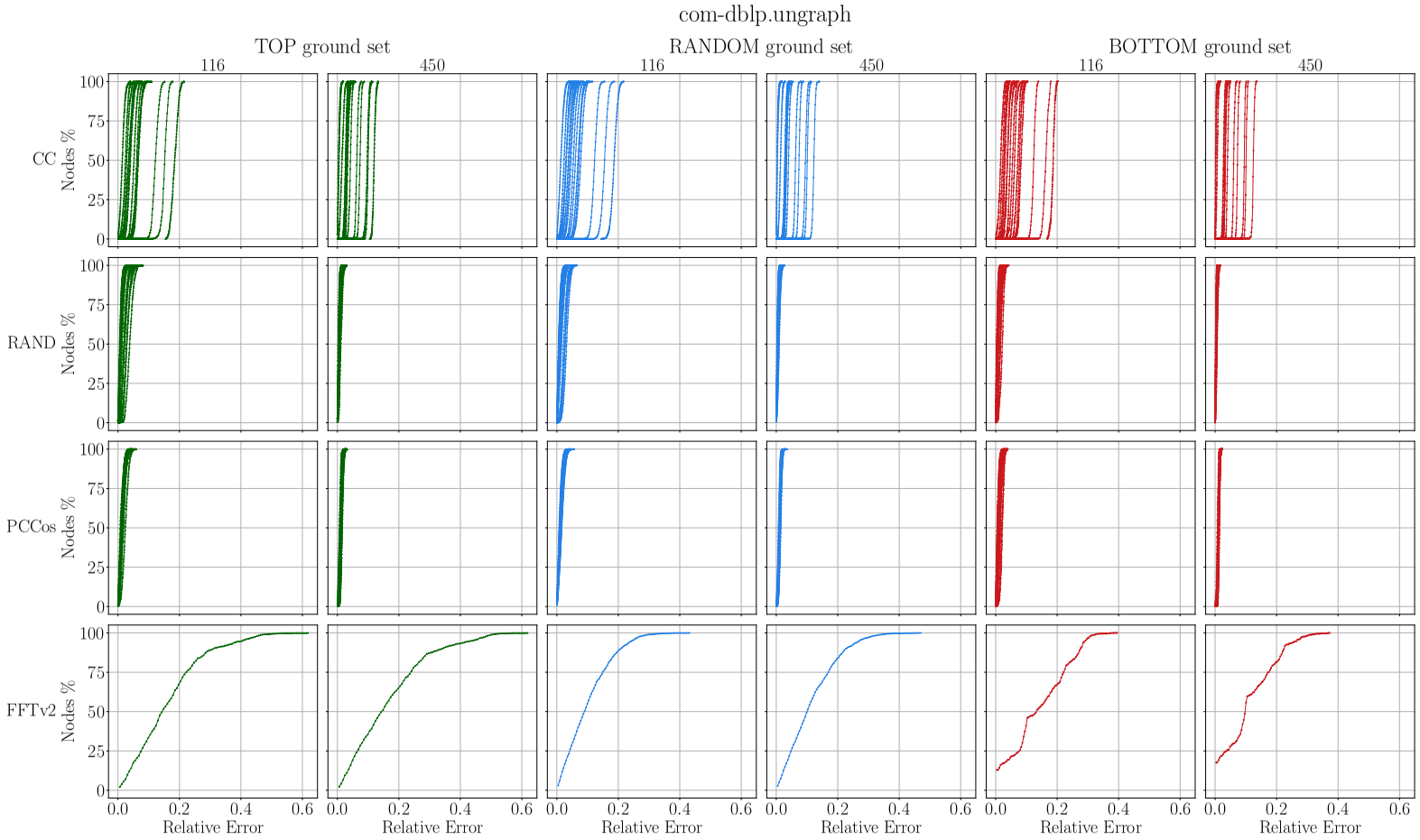

Figure 5.8: Best methods comparison on com-dblp.ungraph social graph. The numbers above each column represent the SSSP computations of a run. They are the average among the SSSP computations of each execution of the run of CC, respectively with $k = 100$ and $k = 400$ as input parameter.

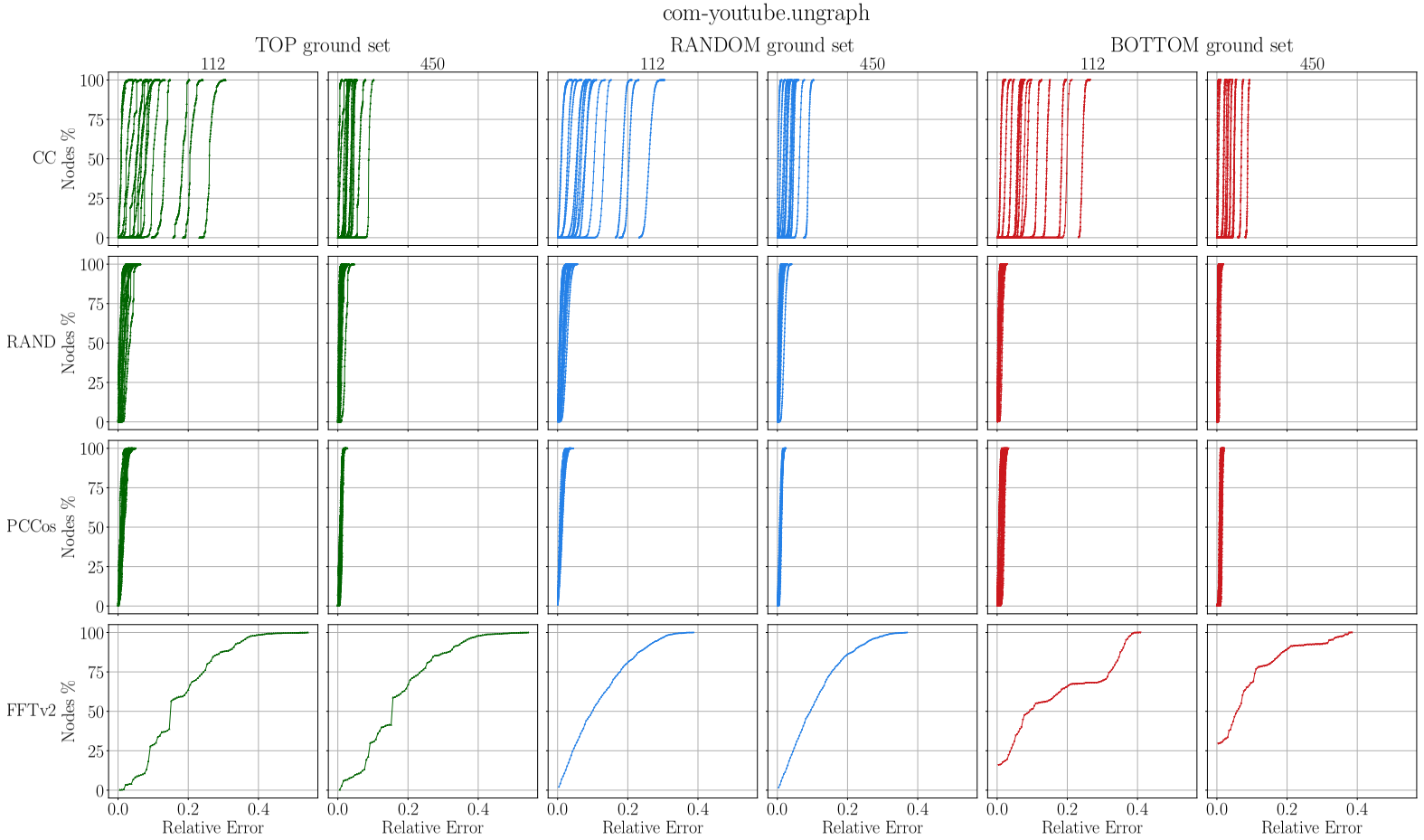

Figure 5.9: Best methods comparison on com-youtube.ungraph social graph. The numbers above each column represent the SSSP computations of a run. They are the average among the SSSP computations of each execution of the run of CC, respectively with $k = 100$ and $k = 400$ as input parameter.

Chapter 6

Conclusion

In this thesis we developed and analyzeed novel competitive approaches for the estimation of the closeness centrality in large graphs, and tested the performances of the resulting sequential implementations against that of state-of-the-art methods, using as benchmarks both road networks and social graphs. Based on the analysis and the experiments results carried out, we conclude that our novel approaches are competitive with the approaches described in [CCK15] and [EW04]. More specifically, the FFT based methods provide good estimates in the case of road networks while the progressive Cc approaches are competitive for both types of graphs.

We initiated our research by exploiting the effectiveness of sample selection of the FFT approach, because we saw a potential for the estimation of the closeness centrality given the guarantees based on the bounds over the distances of the vertices of a graph from this sample. We were aware that the approach was promising only for road networks whereas for higher dimensional graphs, such as social graphs, they would produce less competitive results. Our expectations were confirmed by the first experiments. We attributed the bad performances on graphs of high dimensionality to the selection procedure, that was selecting only peripheral nodes of the graph, hence producing poor estimates overall. To address this problem we then introduced a probabilistic component for the selection of the nodes, which led us to the k-median++ and k-means++ approaches. We expected to improve the performances on social graphs without diminishing the performances on road networks. Our line of reasoning was that the random selection of a vertex, based on the distances from the sample, would select with higher probability a peripheral vertex with higher probability, but, it would also include in the sample intermediate vertices with nonnegligible probability which could then yield better estimates on social graphs. This time the results invalidated our expectation, since, in fact, in fact there was no significant improvement on social graphs. In or-

der to combine the positive aspects of the methods in the literature and our novel approaches, we designed progressive versions of Cck approach. We expected to be able to improve the performance by iteratively adding vertices to the sample, which allows to refine the sampling coefficients and thus the approximation of the universal Probability Proportional to Size ideal strategy. The analysis we have conducted and the experimental results have shown that the two new methods implementing this idea are valuable alternatives to the already existing ones. In conclusion, based on our analysis, the approaches inspired by the clustering algorithms are competitive only in case of road networks, while we argued that the progressive methods based on the same concepts presented in [CCK15], under certain conditions on the sampling coefficients, have similar statistical guarantees of the approach on which they are based. The experimental outcome confirms the results of the analysis and show that the progressive methods based on Cck approach are the most effective novel approaches. PCCos has proved to improve the estimates' quality with respect to CC, showing lower relative errors variances over all the runs, and it has also been competitive to RAND (where sampling is performed uniformly and not proportionally to distances), which has surprisingly given the best closeness centrality estimates overall. In addition, we want to point out the competitiveness of the FFT approaches on road networks, which is a remarkable result and makes FFT a valid deterministic option for the closeness centrality estimation on this type of graphs.

Based on these conclusions, there are several possible directions for future research on the topic. Our experiments have been conducted using unweighted social graphs, which are scale-free graphs that feature the small-world phenomenon. One might experiment with these methods on weighted scale-free graphs, which may or may not feature the small-world phenomenon, so to understand if even on these types of graphs the algorithms' estimates quality comply with our conclusions for their unweighted counterparts. Also, our work based the comparison on the sequential implementation of these approaches. A possible challenging and interesting work may be to design and implement these methods in a big data framework such as Map Reduce/Spark or the new Google Cloud Dataflow. Another interesting aspect to explore is the comparison of these methods with Chechik's approach that uses the VarOpt sampling [CDK$^+$08] instead of the Poisson sampling, which removes the variability of the sample size, and may thus decrease the variability of the estimates.

Throughout this work we showed that there is still room for improvement in the quest for competitive solutions for closeness centrality estimation on large graphs. Furtyher research is needed on novel methods that may prove to be a promising alternative for the state-of-the-art approaches.

References

[AV07] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[Bav50] Alex Bavelas. Communication patterns in task-oriented groups. *Journal of the Acoustical Society of America*, 22:725–730, November 1950.

[BV14] Paolo Boldi and Sebastiano Vigna. Axioms for centrality. *Internet Mathematics*, 10:222–262, 2014.

[CCK15] Shiri Chechik, Edith Cohen, and Haim Kaplan. Average distance queries through weighted samples in graphs and metric spaces: High scalability with tight statistical guarantees. *CoRR*, abs/1503.08528, 2015.

[CDK+08] Edith Cohen, Nick G. Duffield, Haim Kaplan, Carsten Lund, and Mikkel Thorup. Variance optimal sampling based estimation of subset sums. *CoRR*, abs/0803.0473, 2008.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[DGJ05] Camil Demetrescu, Andrew Goldberg, and David Johnson. 9th dimacs implementation challenge - shortest paths. `http://users.diag.uniroma1.it/challenge9/download.shtml`, 2005.

[EHT65] Paul Erdös, Frank Harary, and William T. Tutte. On the dimension of a graph. *Mathematika*, 12(2):118–122, 1965.

[EW04] David Eppstein and Joseph Wang. Fast approximation of centrality. *Journal of Graph Algorithms and Applications*, 8:39–45, September 2004.

[Gon85] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293 – 306, 1985.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *J. American Statist. Assoc.*, 58:13 – 30, March 1963.

[LK14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[Llo57] Stuart P. Lloyd. Least squares quantitization in pcm. *IEEE Transactions on Information*, 1957.

[Roc09] Yannick Rochat. Closeness centrality extended to unconnected graphs : The harmonic centrality index. *ANSA*, August 2009.

[SLL02] Jeremy Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[Vit85] Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, March 1985.