Università degli Studi di Padova

Sede Amministrativa: Università degli Studi di Padova
Dipartimento di Ingegneria dell'Informazione

Corso di Dottorato di Ricerca in: Ingegneria dell'Informazione
Percorso di studio: Scienza e Tecnologia dell'Informazione
Ciclo : 29°

Clustering-based Algorithms for Big Data Computations

Coordinatore: Ch.mo Prof. Matteo Bertocco
Supervisore: Ch.mo Prof. Andrea Alberto Pietracaprina

Dottorando: Matteo Ceccarello

Abstract

In the age of big data, the amount of information that applications need to process often exceeds the computational capabilities of single machines. To cope with this deluge of data, new computational models have been defined. The MapReduce model allows the development of distributed algorithms targeted at large clusters, where each machine can only store a small fraction of the data. In the streaming model a single processor processes on-the-fly an incoming stream of data, using only limited memory. The specific characteristics of these models combined with the necessity of processing very large datasets rule out, in many cases, the adoption of known algorithmic strategies, prompting the development of new ones. In this context, clustering — the process of grouping together elements according to some proximity measure — is a valuable tool, which allows to build succinct summaries of the input data.

In this thesis we develop novel algorithms for some fundamental problems, where clustering is a key ingredient to cope with very large instances or is itself the ultimate target. First, we consider the problem of *approximating the diameter of an undirected graph*, a fundamental metric in graph analytics, for which the known exact algorithms are too costly to use for very large inputs. We develop a MapReduce algorithm for this problem which, for the important class of graphs of bounded doubling dimension, features a polylogarithmic approximation guarantee, uses linear memory and executes in a number of parallel rounds that can be made sublinear in the input graph's diameter. To the best of our knowledge, ours is the first parallel algorithm with these guarantees. Our algorithm leverages a novel clustering primitive to extract a concise summary of the input graph on which to compute the diameter approximation. We complement our theoretical analysis with an extensive experimental evaluation, finding that our algorithm features an approximation quality significantly better than the theoretical upper bound and high scalability.

Next, we consider the problem of *clustering uncertain graphs*, that is, graphs where each edge has a probability of existence, specified as part of the input. These graphs, whose applications range from biology to privacy in social networks, have an exponential number of possible deterministic realizations, which impose a big-data perspective. We develop the first algorithms for clustering uncertain graphs with provable approximation guarantees which aim at maximizing the probability that nodes be connected to the centers of their assigned clusters. A preliminary suite of experiments, provides evidence that the quality of the clusterings returned by our algorithms compare very favorably with respect to previous approaches with no theoretical guarantees.

Finally, we deal with the problem of *diversity maximization*, which is a fundamental primitive in big data analytics: given a set of points in a metric space we are asked to provide a small subset maximizing some notion of diversity. We provide

efficient streaming and MapReduce algorithms with approximation guarantees that can be made arbitrarily close to the ones of the best sequential algorithms available. The algorithms crucially rely on the use of a k-center clustering primitive to extract a succinct summary of the data and their analysis is expressed in terms of the doubling dimension of the input point set. Moreover, unlike previously known algorithms, ours feature an interesting tradeoff between approximation quality and memory requirements. Our theoretical findings are supported by the first experimental analysis of diversity maximization algorithms in streaming and MapReduce, which highlights the tradeoffs of our algorithms on both real-world and synthetic datasets. Moreover, our algorithms exhibit good scalability, and a significantly better performance than the approaches proposed in previous works.

Ringraziamenti

Grazie ai miei genitori Liliana e Ermanno, che mi hanno sempre incoraggiato a coltivare le mie passioni. Grazie a Martina, che porta nella mia vita la musica e la bellezza. Grazie ad Andrea e Geppino, per avermi guidato con competenza e passione. Grazie a Marco, Mattia, Giulia e tutti gli amici del coro. Grazie a Mattia e Vincenzo e tutti gli amici del laboratorio ACG.

Acknowledgements

I would like to thank Professor Eli Upfal of Brown University in Providence for all the work we have done together. Thanks to Priyanka, Abhijat, Jon, and Dani for making my stay in Providence so nice.

Contents

Chapter 1

Introduction

The last two decades have witnessed an impressive growth in the amount of digital data produced in an ample spectrum of application domains. Every digital process and social media exchange produces large amounts of data. Mobile devices, sensor networks, and internet-connected appliances are multiplying the amount of information that is being produced. Nowadays, estimates of the volume of data produced are in the order of zettabytes (10^{21} bytes), with a yearly increase rate of around 40% [FB13]. The effective exploitation of this wealth of data is of great value to science, business, government, and society in general, and it has been reckoned as one of the most important scientific challenges of the 21st century. A term has been coined to identify this phenomenon and the challenges it poses: *big data*. In [Lan01], big data is characterized in terms of three properties:

volume: The amount of data is unprecedented, and is increasing at a higher rate than our ability to process it;

variety: Data comes in a wide array of forms, both structured and unstructured, including text, video, audio and more;

velocity: Data is generated continuously at a high rate, and we are interested in extracting useful information almost in real time.

These three properties clearly illustrate how dealing with big data is requiring a sharp paradigm shift with respect to traditional computing. For example, in mainstream algorithmics we regard time complexities polynomial in the input size as feasible. However, when huge datasets have to be processed, even a quadratic running time becomes impractical. Thus, for computational problems whose state-of-the-art exact solutions require superlinear time, novel algorithmic strategies must be designed in order to tackle very large instances, and these strategies should give up the quest for exact solutions and aim at exposing suitable tradeoffs between accuracy of the returned solution and computational efficiency. In the big data realm,

space needs to be maintained under control. Algorithmic strategies that attain efficiency at the expense of a superlinear increase of the memory requirements may become impractical. Moreover, the analysis of large datasets must often face the presence of redundant or irrelevant data, which may increase the computational complexity and obfuscate relevant information, as well as the presence of some degree of uncertainty in the data, which may weaken the guarantees on the final results.

Motivated by the above scenario, novel computational frameworks have emerged in recent years, establishing themselves as standards both in academia and industry for big data processing. In particular, MapReduce [DG08; KSV10; Pie+12] is one of the most widely used model for batch processing huge datasets. To address the challenges posed by big data applications, it leverages the massive amount of parallelism offered by large clusters made of off-the-shelf medium-power components, endowed with large robust mass storage and transparent provision of fault-tolerance.

Along with new models of computation, some earlier ones have found new life in the big data era. The streaming model [HRR98] was originally introduced in the '90s to process data stored in slow tertiary storage, using only the limited main memory of a single machine. In this setting, for efficiency, one could only perform a small number of sequential reads of the data, keeping a small amount of information in main memory. Nowadays, with distributed filesystems [GGL03; Shv+10] enabling parallel access to large datasets, and with the consequent introduction of models such as MapReduce, the aforementioned motivating scenario for the streaming model is becoming less relevant. However, we mentioned how big data is also characterized by *velocity*: data is produced at a tremendous rate, and we want to process it nearly in real time. In this scenario, the streaming model finds a new application: inputs are provided as a continuous stream of data of possibly unbounded size, and storing the data for offline analysis may not be feasible or even desirable. In this case, data analysis must be performed by processing the data on the fly, and using only a limited amount of local memory.

This thesis focuses on the development of efficient algorithms for three relevant problems in the big data setting. Our goal is to design efficient algorithms for these problems, which are practical and feature provable guarantees on the performance and on the quality of the returned solutions, possibly exposing suitable tradeoffs with available resources. To assess the practicality of our approaches, for each problem we complement our theoretical findings with experimental evaluations.

The algorithms we propose share a common approach: the use of *clustering* as a tool. Clustering is the process of grouping elements of the input according to some criterion. In the big data context, clustering lends itself to be used for different purposes. On the one hand, one can be interested in considering each cluster separately, examining its elements as they share some common characteristic. On the other, one can use clustering as a way to build a summary of the input set, where each cluster is considered as a whole, representing all its elements. In this thesis we adopt both perspectives on clustering: for one of the problems we consider we are interested in clustering in itself, to relate elements belonging to the same cluster; for the other two problems we use clustering to provide summaries

of the input on which compute approximations. There are several definitions of clustering, based on the choice of the metric to group elements together. In this thesis, we concentrate on k-*center clustering*, which requires to group elements so as to minimize their distance from some set of *centers*, that is a suitably chosen subset of the input. In the analysis of our algorithms, we make use of the notion of *doubling dimension* of a metric space, which is the smallest D such that any ball of radius r can be covered by at most 2^D balls of radius $2r$.

The first problem we consider is the approximation of the diameter of large graphs, which is a fundamental primitive in graph analytics, with applications to a wide variety of networks, such as biological, communication, social, and web networks. For this problem, several exact algorithms are known. However, when the size of the graph becomes too large to fit into main memory, and computational frameworks such as MapReduce need to be used, exact algorithms are ruled out by their superlinear complexity. We develop a parallel algorithm for approximating the diameter of large weighted graphs, whose properties make it suitable to be implemented on distributed platforms such as MapReduce. In particular, our algorithm employs a clustering strategy to build a small sketch of the input graph on which to compute the diameter approximation. Our clustering strategy is of independent interest, and we prove that it provides a polylogarithmic approximation to the k-center problem. Moreover, we show that our algorithm attains a polylogarithmic approximation factor for the graph diameter. On spaces of bounded doubling dimension, ours is the first approximation algorithm that requires a number of MapReduce rounds sublinear in the diameter of the graph, using only linear space. Even on graphs of unknown doubling dimension, experimental evidence on datasets of up to billions of edges shows that our algorithm is fast and scalable, and attains very good approximation ratios. We implemented our algorithms on the Spark MapReduce engine, contributing the implementation to the public repository of Spark libraries.

The second problem we study is the clustering of *uncertain graphs*. These graphs are used to represent interacting entities where each interaction presents some degree of uncertainty. This uncertainty is encoded in the uncertain graph by assigning to each edge a probability of existence. For instance, in networks representing the interactions of proteins, the existence of an interaction is determined through a noisy experiment: the interaction is then represented as an edge with an existence probability reflecting the degree of confidence in the measure. Even for uncertain graphs of moderate size, the exponential number of their possible deterministic realizations frames the problem in the big data context. The problem of clustering uncertain graphs is hard for two independent reasons: clustering is NP-hard, and computing connection probabilities between arbitrary nodes is #P-complete. We develop clustering algorithms for uncertain graphs, aiming at minimizing a measure related to the connection probability of nodes belonging to the same cluster. We also show how to embed a progressive sampling strategy into our algorithm in order to efficiently and accurately estimate connection probabilities. To the best of our knowledge, our algorithms for clustering uncertain graphs are the first with provable guarantees on the quality of the approximation. We also present a preliminary suite of experiments, which shows that our algorithms compare favorably with previous works.

Last, we study *diversity maximization*, which is a fundamental problem in big data analytics: given a large set of points in a metric space, the goal is to find a small cardinality subset that maximizes some measure of diversity. Such a subset can be employed as a small representative summary of the large input point set. Diversity maximization has several applications, including aggregator websites, recommendation systems, web search, e-commerce, and facility location. We propose MapReduce and streaming algorithms for the diversity maximization problem which expose a tradeoff between the available resources and the quality of the approximation. Our algorithms are based on the (composable) core-set technique, in which the diversity is computed on a small subset of the (partitioned) input. In particular, we use k-center clustering to build our core-sets, leveraging its properties to ensure the accuracy of the result. With respect to previous works, that provided constant-approximations irrespective of the available computing resources, on metric spaces with bounded doubling dimension we obtain significantly better approximation factors. In MapReduce, our algorithms run in a constant number of rounds, whereas in streaming they require only one or two passes over the data. We support our theoretical findings with an extensive suite of experiments, showcasing the performance and accuracy of our algorithms. These experiments also provide evidence that our approach is effective also on point sets for which the doubling dimension is unknown. To the best of our knowledge, ours is the first work to study experimentally the performance of diversity maximization algorithms on MapReduce and streaming.

Part of this work was done while visiting Brown University (Providence, USA), as a *visiting research fellow*, from February to August 2016.

The rest of the thesis is structured as follows. In Chapter 2, we introduce some common notation, we describe the MapReduce and streaming models, and the notion of k-center clustering. In Chapter 3, we describe our diameter approximation algorithm. Our approach to the clustering of uncertain graphs is described in Chapter 4, while in Chapter 5 we deal with the problem of diversity maximization. Finally, in Chapter 6 we summarize our results and we outline a number of directions for future work. In each chapter we review previous work relevant to the results being presented. Part of the material of this thesis has been published in different venues, which are listed in section "Our publications" of the references.

Chapter 2

Preliminaries

In this chapter we introduce some core concepts that will be used throughout the thesis. Other preliminary material, specific to particular topics, is deferred to the relevant chapters.

2.1 Notation and basic concepts

Sets, distances, and spaces. In this dissertation, sets will be denoted by capital letters, such as C, P, and S. For a set S, the set of all its subsets is denoted as 2^S.

We denote with $\mathcal{M} = (M, \mathrm{dist})$ a general metric space, which is a set M paired with a function $\mathrm{dist} : M \times M \to \mathbb{R}_0^+$ such that for any $x, y \in M$

1. $\mathrm{dist}(x, y) \geqslant 0$
2. $\mathrm{dist}(x, y) = 0 \iff x = y$
3. $\mathrm{dist}(x, y) = \mathrm{dist}(y, x)$
4. $\mathrm{dist}(x, y) \leqslant \mathrm{dist}(x, z) + \mathrm{dist}(z, y), \quad \forall z \in M$ (triangle inequality)

We call such a function a *distance function*. For convenience, we also define the distance between a point and a set. Given a set S, a subset $P \subset S$, and a point $x \in S \setminus P$, the distance between x and P is $\min_{y \in P} \mathrm{dist}(x, y)$. As a shorthand, we will use the following notation

$$\mathrm{dist}(x, P) = \min_{y \in P} \mathrm{dist}(x, y) \tag{2.1}$$

Given a metric space $\mathcal{M} = (M, \mathrm{dist})$ and a point $x \in M$, the *ball of radius* r *centered at* x is the set of all points in M at distance at most r from x.

We define two more fundamental concepts we will use, namely the *radius* of a subset and the *farness* of a set. Given a metric space $\mathcal{M} = (M, \mathrm{dist})$, a set $S \subseteq M$, and a subset $C \subset S$, the radius of C with respect to S is defined as

$$r_C(S) = \max_{x \in S} \mathrm{dist}(x, C) \tag{2.2}$$

The *farness* of the subset C is defined as the minimum distance between any two elements

$$\rho_C = \min_{x,y \in C} \mathrm{dist}(x, y) \tag{2.3}$$

With $r_k^*(S)$ we denote the maximum radius of any subset of S with k elements. Similarly, we denote with ρ_k^* the maximum farness of any subset of k elements of S.

Doubling dimension. A fundamental concept we will use in this thesis is that of *doubling dimension*.

Definition 1 ([GKL03])**.** *The* doubling dimension *of a metric space $\mathcal{M}$ is the smallest* D *such that any ball of radius* r *is covered by at most* 2^D *balls of radius* $r/2$*.*

Note that the doubling dimension may not be a constant. Metric spaces with bounded doubling dimension are also called *doubling spaces*. Among others, this class of metric spaces includes the important family of ℓ_p-spaces. Consider for any fixed $d > 0$ the set $\mathbb{R}^d$, and for any $p > 0$ consider the function

$$\ell_p(x, y) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p} \qquad \forall x, y \in \mathbb{R}^d$$

The metric space $(\mathbb{R}^d, \ell_p)$, called ℓ_p-space, has doubling dimension $O(d)$ [GKL03]. Therefore we have that Euclidean spaces (which are ℓ_p-spaces with $p = 2$), have constant doubling dimension. Computing the doubling dimension of a metric $\mathcal{M} = (M, \mathrm{dist})$ is NP-hard, but a 2-approximation algorithm exists [GK13].

Given their properties, doubling spaces have been used in the literature for several purposes, including routing [KSW04; Abr+06; Sli07; KRX08], clustering [Tal04; FM10; ABS10], nearest neighbour search [KL04; RG06; BKL06], machine learning [BLL09; GKK14], and traveling salesman [Tal04; BGK12].

Probabilities. We will denote the probability of an event $\mathcal{X}$ with $\Pr[\mathcal{X}]$. The expectation of a random variable $\mathcal{X}$ is denoted as $\mathbb{E}[\mathcal{X}]$, and its variance with $\mathrm{var}[\mathcal{X}]$.

Graphs. In this thesis we will deal with undirected graphs, both weighted and unweighted. We denote an unweighted graph G with the pair $G = (V, E)$, where V is the set of nodes, and $E \subseteq \{\{u, v\} : u, v \in V\}$ is the set of edges. Moreover, we define $n = |V|$ and $m = |E|$. Weighted graphs are defined by a triple $G = (V, E, w)$, where w is the *weight function*, i.e. a function $w : V \times V \to \mathbb{R}^+$. In this thesis we will consider only strictly positive weight functions. Given a connected graph G and a pair of nodes $u, v \in V$, we define their distance as the length of the shortest (weighted) path connecting them, and we denote it with $\mathrm{dist}(u, v)$. The *diameter* of an unweighted connected graph $G = (V, E)$, denoted by $\Phi(G)$, is the maximum distance between any pair of nodes:

$$\Phi(G) = \max_{u,v \in V} \mathrm{dist}(u, v) \tag{2.4}$$

In the case of weighted graphs, we distinguish between the *weighted* diameter and the *unweighted* diameter. The weighted diameter of a graph $G = (V, E, w)$, denoted by $\Phi(G)$, is the maximum distance between any pair of nodes. The unweighted diameter, instead, is the diameter of the unweighted graph obtained from G by ignoring all edge weights, and is denoted by $\Psi(G)$.

For disconnected graphs, we define the (weighted) diameter to be the largest (weighted) diameter of the subgraphs induced by the connected components.

Note that for graphs we consider, that is graphs with positive edge weights, the pair (V, dist) of nodes together with shortest path distances is a metric space. Therefore, we can extend the definition of *ball of radius* r to graphs. Specifically, given a graph $G = (V, E)$ and a node $v \in V$, the ball of radius r centered in v is the set of nodes $u \in V$ such that $\mathrm{dist}(v, u) \leqslant r$. Similarly, we can extend the notion of *doubling dimension* to graphs.

Definition 2. *The* doubling dimension *of a graph* $G = (V, E)$ *is the smallest* D *such that any ball of radius* r *is covered by at most* 2^D *balls of radius* $r/2$.

The doubling dimension of a graph may not be a constant. However, several important classes of graphs have constant doubling dimension, including d-dimensional arrays, whose doubling dimension is $\Theta(d)$.

2.2 The MapReduce model

When dealing with large amounts of data, the crucial challenge is how to process all this data efficiently. Using a single machine is impractical or even impossible: the limited amount of memory available on a single machine is a show stopper for many algorithms when the input size grows beyond the memory limits. It is therefore necessary to resort to parallel and distributed algorithms. To ease the development of useful applications and relieve application programmers from the burden of managing low-level parallelism, several parallel processing frameworks have been developed by the industry and academia [DG08; Isa+07; Aki+15; Whi15; Zah+10]. Among these frameworks, MapReduce [DG08] is the most widespread. Created at Google to ease the development and deployment of distributed data processing applications, its open source incarnation Hadoop [Whi15] has seen a wide adoption in the industry. The main advantage of MapReduce as a computational platform is that it allows to process massive amounts of data using large clusters of commodity computers, instead of specialized and expensive hardware.

The high level idea of the MapReduce paradigm is defined in the following, while the next subsection we will describe the computational model adopted in this thesis. A MapReduce algorithm operates on multisets of key-value pairs, which are transformed by means of two functions, namely *map* and *reduce*. The `map` function takes as input a *single* key-value pair and produces in output a multiset of pairs on possibly different domains. The `reduce` function works on the outputs of the `map` function, taking as input all the pairs sharing the same key. The glue that connects the `map` and `reduce` function is the *shuffle* operation. Conceptually, the *shuffle* operation takes all the pairs produced by the `map` function, and groups together the ones with the same key, so that the `reduce` function can process each group. The ensemble of `map`, *shuffle*, and `reduce` functions defines a `round`, and

the output of a round can be the input of another. Indeed, MapReduce algorithms execute as a sequence of rounds, each transforming a multiset of key-value pairs into another. Observe that, given a pair, the $\mathtt{map}$ function operates independently of other pairs. Similarly, the $\mathtt{reduce}$ function can process a given a key and all its associated values independently of the others. This independence is what enables the parallel execution of the $\mathtt{map}$ and $\mathtt{reduce}$ functions on different inputs. In a typical MapReduce implementation there are several processors, each applying in parallel the $\mathtt{map}$ and $\mathtt{reduce}$ functions on different parts of the input, which is partitioned across the processors. The *shuffle* operation, then, exchanges data through the network, so as to bring all the values with the same key in the memory space of a single processor. Note that the details of the communication, which is performed by the *shuffle* operation, are hidden from the user, who only needs to specify how the $\mathtt{map}$ and $\mathtt{reduce}$ functions work on data. This data-centric view is what makes MapReduce oblivious to the execution platform.

2.2.1 The MapReduce computational model

While useful in practice, MapReduce was initially lacking a theoretical model to support the analysis of algorithms, hindering a systematic development of efficient algorithmic strategies. In subsequent years, several works have attempted to fill this gap [KSV10; GSZ11; Pie+12].

In this section, we review the MapReduce computational model presented by Pietracaprina et al. [Pie+12], which is the model we use in this thesis to analyze the complexity of our MapReduce algorithms. As noted in [Lat+11] and [Pie+12], a $\mathtt{reduce}$ function can embed the subsequent $\mathtt{map}$ function, simplifying the presentation. Therefore, in what follows we will assume that the $\mathtt{map}$ function is embedded in the preceding $\mathtt{reduce}$ function, thus adopting a slightly different terminology with respect to the original MapReduce paradigm. We assume that the first *reduce* function is the identity function, so that the first round effectively runs only the *map* function. In this thesis, we define a *reducer* as a processor with limited memory executing a given $\mathtt{reduce}$ function.

As we mentioned in the previous section, an algorithm specified in the MapReduce model executes as a sequence of *rounds*. Round r, with $r \geqslant 1$, takes as input a multiset of key-value pairs W_r where the keys are taken from a universe K_r. Then, by means of a *reduce* function ρ_r, W_r is transformed into two multisets: W_{r+1} which is the input to the next round (empty if r is the last round), and O_r which is the (possibly empty) output of round r. The output of the algorithm is $\bigcup_{r \geqslant 1} O_r$, where $\bigcup$ denotes the union of multisets. Note that K_{r+1}, the universe of keys the next round, can be different from K_r, and the same holds for the universe of values. The reduce function ρ_r defines the transformation operated at round r. It is applied independently to all the pairs in W_r sharing the same key, denoted as $W_{r,k} \subseteq W_r$, and is required to run in time polynomial in $|W_{r,k}|$.

The model is defined in terms of two parameters M_L and M_A, and is therefore called MR(M_L, M_A)[1]. An algorithm in the MR(M_L, M_A) model is called an

[1]The notation used in this thesis is slightly different than the one used in the original paper [Pie+12], with M_L used in place of m and M_A instead of M. This substitution is to avoid confusion when dealing with graphs on MapReduce, where m refers to the number of edges

MR-algorithm. The two parameters constrain the resources available to the reduce function: M_L is the amount of memory available for local computations, whereas M_A is the *aggregate* memory across all reducers. Let $M_{r,k}$ be the amount of memory needed to compute $\rho_r(W_{r,k})$ on the RAM model, including both the work space and the space taken by the input. The model imposes the following constraints

- $M_{r,k} \in O(M_L)$, for every $r \geqslant 1$ and $k \in U_r$
- $\sum_{k \in U_r} M_{r,k} \in O(M_A)$, for every $r \geqslant 1$
- $\sum_{r \geqslant 1} O_r \in O(M_A)$

Note that the size of the output is not constrained by the memory available for the local computation, while it is limited by the total aggregate memory available. The complexity of an MR-algorithm is expressed as the number of rounds executed in the worst case, and is a function of the input size n and of the parameters M_L and M_A.

Two fundamental primitives that are used in algorithms presented in this dissertation are sorting and prefix sum. The input for both primitives is a set of n key-value pairs (i, a_i), with $i \in [0, 1)$ and $a_i \in S$, where S is a suitable set. For sorting, given a total order over S, the output is a set of n key-value pairs (i, b_i), where $b_{i-1} < b_i$ and the b_i's form a permutation of the a_i's. For prefix sums, we consider a binary associative operation $\oplus$ over S: the output consists of a collection of n pairs (i, b_i) where $b_i = a_0 \oplus \cdots \oplus a_i$ for $0 \leqslant i < n$. We report a fundamental result from the original paper [Pie+12].

Theorem 1 ([Pie+12])**.** *The sorting and prefix sum primitives for inputs of size n can be performed in $O\left(\log_{M_L} n\right)$ round in MR(M_L, M_A) for any $M = \Omega(n)$.*

Observe that, when $M_L = O(n^\varepsilon)$, for some $\varepsilon \in (0, 1)$, the sorting and prefix sum primitives execute in a constant number of rounds.

2.2.2 MapReduce implementations and our experimental platform

There are several implementations of MapReduce, aside from the original one by Google [DG08]. The most widespread, in academia and industry alike is Hadoop [Whi15], which is the de-facto standard open source implementation. In order to run on clusters of commodity hardware in a scalable and fault tolerant way, Hadoop leverages the Hadoop Distributed Filesystem [Shv+10], which distributes and replicates data across machines. Each round reads its input and writes its output on this distributed filesystem, thus ensuring fault tolerance. On one hand this ensures that no data is lost during the processing, but on the other makes each round very costly. Moreover, each round includes a shuffle phase implemented as an expensive all-to-all communication. The combination of these two shortcomings prompted the development of alternative implementations.

Among the alternatives to Hadoop, Spark [Zah+10] is the one which gained the most momentum in the last few years. To improve performance, Spark caches the partitioned result of each round in main memory, from where the subsequent round can read it. By doing this, Spark avoids entirely the expensive round-trip to the distributed filesystem, considerably lowering the cost of each round. In order

to provide fault tolerance, Spark embeds in the result of each round its *lineage*, that is the sequence of transformations that were used to compute it, starting from the program's input safely stored on disk. If a reducer fails, then some partitions of the results computed so-far are lost. To recover, Spark can use the lineage of the lost partitions to recompute the missing data: the rationale is that the amortized cost of recomputing a fraction of the results every once in a while is less than the cost of accessing the distributed filesystem in each round. Given its superior performance, Spark is our platform of choice for implementing the MapReduce algorithms of this thesis.

Experimental platform. We deploy Spark version 1.6.1 on a in-house cluster of 16 machines, each equipped with 18Gb of main memory and a 3.07Ghz Intel i7 processor with 4 cores. The machines are connected with a 10Gbps Ethernet network. Moreover, each machine has 6Tb of storage, of which 3 are shared with the rest of the cluster through the Hadoop Distributed Filesystem, and 3 are available for local operation.

2.2.3 Experimental evaluation of the MapReduce model

In [CS15], we performed an experimental evaluation of the $\mathrm{MR}(M_L, M_A)$ model. The goal was to investigate the $\mathrm{MR}(M_L, M_A)$ model from a practical standpoint, exercising the tradeoff between available space and performance. We chose matrix multiplication as a case study since it is both computation and communication intensive. This allows to significantly load the system and to asses its performance under stress. Moreover, the possibility to decompose the problem into arbitrary-size subproblems makes matrix multiplication an ideal candidate to investigate the tradeoffs between available resources (local and aggregate memory) and performance. We implemented the sparse and dense matrix multiplication algorithms used in the reference paper [Pie+12] on the Hadoop platform[2], since it is the de-facto standard and the most adherent to the original specification.

This study gave us insights that have been useful for all our subsequent work. The main takeaway is that, while the performance is mostly determined by the number of rounds, one must pay attention also as to the volume of communication in a single round. In fact, it is true that each round in Hadoop involves an all-to-all communication, and therefore it is desirable to minimize the number of rounds. However, the volume of data shuffled in a single-round algorithm may be too high and congest the network. In such cases, splitting the same volume of communication over multiple rounds is beneficial and gives a better performance.

For a detailed exposition of these results, we refer to [CS15].

2.3 The Streaming model

Streaming [HRR98] is another popular model in the big data setting. Historically, this model was motivated by technological factors. Data was often stored on slow tertiary storage which provided only sequential access. In this case, algorithms

[2]The library, called M_3, is freely available at `http://crono.dei.unipd.it/m3/`.

were forced to perform only sequential passes over the data, and doing multiple passes was impractical. Nowadays, technological improvements mitigated this problem. However, several applications still require the development of streaming algorithms. In particular, in many cases there is a need to analyze in real-time large volumes of data as it is generated. For instance, one might be interested in analyzing the Twitter stream in real-time, or might want to compute statistics on a stream of financial transactions. In such cases, batch systems such as MapReduce are of little help, and we must resort to streaming algorithms.

In the streaming model there is a single processor with limited memory, which can perform only a limited number of sequential scans over the input. More formally, a *data stream* is a sequence $x_1, \ldots, x_i, \ldots, x_n$ of elements of a set X, which can only be read in increasing order of i. The streaming model is described by two parameters [HRR98]: the *number of passes* p and the *main memory space* s (also called *workspace* in the literature), that is the memory available to the processor. When developing a streaming algorithm, the goal is to use as little memory as possible, while minimizing the number of passes. Despite the heavy limitations imposed by the streaming model, several data sketching and statistics problems can be solved using $O(1)$ passes and limited space [FM83; AMS99; Gil+02; Fei+02; Cha+04; Mut05; Fla+08].

Oftentimes, however, there is a dichotomy between the number of passes p and the memory s. This happens when dealing with graphs, for instance. In the streaming model, a graph is represented as the sequence of its edges. With the restriction of being able to maintain in memory only a portion of size s is the edge set, even the simplest problems such as graph connectivity require $\Omega(n/s)$ passes over the data, with n being the size of the graph [HRR98; DFR09]. The tradeoff here is clear: the only way of performing fewer passes is to allow larger memories, with the extreme case that a one-pass algorithm can be obtained only if memory linear in the size of the graph is available.

2.4 k-center clustering

As we discussed in the introduction, this thesis focuses on the use of clustering as a tool to solve several big data problems. In this section, we define the clustering problem and review some fundamental related results. Given a set of points in a metric space, *clustering* is the process of grouping them into *clusters* according to some criterion. These clusters can then serve for different purposes: a cluster can be used as a proxy for its elements, summarizing them; or we may be interested in examining the constituents of each cluster. Indeed, clustering is applied in a variety of different contexts, like Information Retrieval [MRS08], computer vision [Yan+08], bioinformatics [NYP12], image hashing [MBE06], and classification [D+73; Pen+05; Dai+07].

Different clustering criteria define different instances of the clustering problem. Some clustering criteria group the elements according to their distance from some set of centers, optimizing for the minimum [Gon85], the sum [Cha+99], or the squared sum [Jai10] of distances. Other approaches forego the selection of a set of centers, for instance grouping nodes with respect to the probability of random walks [Don08]. Given a set S in a metric space, we define by k-*clustering* a partition

Algorithm 1: GMM(S, k)

Input: $S = \{v_1, \ldots, v_n\}$ the sets of elements to be clustered, k number of clusters to be found.
Output: The set of k cluster centers.

$C \leftarrow$ an arbitrary point $c_1 \in S$
for $i \leftarrow 2$ **to** k **do**
  $c_i \leftarrow \arg\max_{v \in S \setminus C} \mathrm{dist}(v, C)$
  $C \leftarrow C \cup c_i$
end
return C

of S in k subsets according to some measure. For the purposes of this thesis we concentrate on the k-*center* clustering problem [Gon85], defined below.

Definition 3. *Given a metric space $\mathcal{M} = (M, \mathrm{dist})$, a set of points $S \subseteq M$, and an integer k, the* k-center *problem requires to find a set $C \subset S$ of k points such that the* radius *of C with respect to S as defined in Equation (2.2)*

$$r_C(S) = \max_{p \in S} \mathrm{dist}(p, C)$$

is minimized.

The problem is NP-hard, and is actually NP-hard to approximate with an approximation factor better than 2 for arbitrary metric spaces [Gon85]. Restricting the input to be in an Euclidean space is of little help. Clustering on the Euclidean line can be done in polynomial time [Bru78]; however it is NP-hard to achieve an approximation factor of $2\cos(\pi/6) - \varepsilon$ on the Euclidean plane, and of $2 - \varepsilon$ on higher dimensional spaces [Gon85], for any $\varepsilon > 0$.

Recall that, given an undirected weighted graph $G = (V, E, w)$ and the shortest-paths distance function $\mathrm{dist} : V \times V \to \mathbb{R}_0^+$, the pair (V, dist) is a metric space, as we observed in Section 2.1. Therefore, for the shortest path distance and other distance functions with similar properties, we can define the k-center problem on graphs as follows.

Definition 4. *Given a weighted graph $G = (V, E, w)$, and a function $\mathrm{dist} : V \times V \to \mathbb{R}_0^+$ such that (S, dist) is a metric space, the k-center problem requires to find a set $C \subset V$ of k nodes such that $\max_{v \in V} \min_{c \in C} \mathrm{dist}(v, c)$ is minimized.*

2.4.1 The GMM algorithm

In his seminal work, Gonzalez presented a simple 2 approximation greedy algorithm for the k-center problem in general metric spaces [Gon85]. The algorithm, which we refer to as GMM (Greedy Minimum Maximum), is depicted in Algorithm 1 and works as follows. First, the set of centers C is initialized with an arbitrary point from the input set S. Then, iteratively, the algorithm adds to C the point in $C \setminus S$ which maximizes the distance from already selected points, breaking ties arbitrarily, until C contains k points. Then, we can build the clustering

by associating each point to the closest center $\in C$. There is an easy sequential implementation of the GMM algorithm which runs in $O(kn)$ time. The following proposition is proved in [Gon85].

Proposition 1. *Given a set S and an integer k, the GMM algorithms computes a set of cluster centers C such that*

$$r_k^*(S) \leqslant r_C(S) \leqslant 2r_k^*(S)$$

We now review the concept of *anticover* [Hal+99], which relates the radius and farness of a set of points.

Definition 5. *Given a set S, a subset $C \subset S$ is an* anticover *if $r_C(S) \leqslant \rho_C$.*

The set of centers found by the GMM algorithm is an anticover, as stated by the fact below. We will use this property of GMM in Chapter 5, when we will deal with the *diversity maximization* problem.

Fact 1. *The set of cluster centers found by the GMM algorithm when building a k-center clustering is an anticover.*

Proof. Let $C = \{c_1, \cdots, c_k\}$ be the centers of the clustering found over a set S by GMM, indexed in the order they were added to C by the algorithm. Let $d_k = \mathrm{dist}(c_k, C \setminus \{c_k\})$. By construction, we have that $\rho_C = d_k$, since the distance of any center to the ones added previously decreases as the center index increases. For the same reason, we have that $d_k \geqslant d_{k+1} = r_C(S)$, where d_{k+1} is the distance of the center that would be added if we were to build a $k+1$-clustering. Therefore, we have $r_C(S) \leqslant d_k = \rho_C$. Since $\mathrm{dist}(p, C) \leqslant r_C(S)$ for any $p \in P$, this suffices to prove that C is an anticover. □

2.4.2 MapReduce k-center algorithms.

Once datasets become large, the sequential GMM algorithm becomes ineffective, due to the memory limits of the RAM model. Because of this, in recent years much work has been devoted to the development of k-center algorithms in the distributed setting [EIM11; BEL13; Mal+15].

In particular, a two-round MapReduce algorithm for the k-center problem in metric spaces has been developed in [Mal+15], improving on previous works. The algorithm, as in [EIM11], assumes the existence of an oracle taking as input a pair of points and returning the distance between them, or that all the pairwise distances are given alongside with the input. The basic idea of the algorithm is simple and effective. First, the input S is partitioned in ℓ non overlapping subsets $S_1, \ldots, S_\ell$, where ℓ is the number of reducers, and S_i is assigned to the i-th reducer. Then, reducer i computes a set C_i of k points by running GMM on S_i. Finally, the algorithm collects $C = C_1 \cup \cdots \cup C_\ell$ in a single reducer, and runs GMM on this set to find the solution. By straightforwardly adapting the results in [Mal+15] to the $\mathrm{MR}(M_L, M_A)$ model, we can state the following proposition.

Proposition 2. *The above MR-algorithm is a 4-approximation algorithm, runs in two rounds with $M_A = \Theta(n)$ and $M_L = O\left(\max\{n^\varepsilon, kn^{1-\varepsilon}\}\right)$.*

Unfortunately, the assumption that there is an oracle for the distance function does not hold for important classes of inputs, such as graphs, where computing distances requires a non-constant number of rounds. Similarly, requiring the $\Theta\left(n^2\right)$ pairwise distances to be given in input is impractical for large values of n.

On MapReduce, a clustering algorithm for unweighted graphs has been presented by Miller et al. [MPX13]. The goal of the algorithm is to partition the input graph in clusters of small radius with only a small fraction of edges between clusters. Note that this algorithm does not optimize the k-center objective function. Nonetheless, its purpose is quite similar, therefore it is a viable candidate for clustering graphs on MapReduce. The algorithm features a single parameter β, which allows to control both the diameter of the clusters and the number of edges between them. The algorithm assigns to each node $u \in V$ a random time shift δ, taken from an exponential distribution with mean $1/\beta$. Then, it grows a cluster centered at u starting at time $\delta_{max} - \delta_u$, where δ_{max} is the maximum shift, unless by that time node u has been already covered by some other cluster. The authors show that in this fashion the graph is partitioned into clusters of maximum radius $O\left((\log n)/\beta\right)$, with high probability, while the expected number of edges between clusters is at most $O\left(\beta m\right)$.

2.4.3 Streaming k-center algorithms.

In Section 2.3, we reviewed the streaming model, where a single processor with limited memory operates on a sequential stream of data. Charikar et al. [Cha+04] developed the *doubling algorithm*, which is a k-center algorithm in the streaming model. Given a metric space $\mathcal{M} = (M, \mathrm{dist})$, a stream of points $S \subset M$, and an integer parameter k, the doubling algorithm works in phases as follows. In the initialization phase, the algorithm accepts the first $k+1$ points of the stream as singleton clusters, and sets a distance threshold d_1 to the minimum distance between any two points seen so far. At the start of phase i, the algorithm has a collection of $k+1$ clusters $C_1, C_2, \ldots, C_{k+1}$, each with a center c_j for $j \in [1, k+1]$, and a distance threshold d_i. The following invariants hold at the beginning of phase i

1. for each C_j and for any $p \in C_j$, we have $\mathrm{dist}(p, c_i) \leqslant 2d_i$
2. for each pair of clusters C_j and C_ℓ we have $\mathrm{dist}(c_j, c_\ell) \geqslant d_i$

Each phase then consists of two stages: the *merging stage* and the *update stage*. In the merging stage, the algorithm reduces the number of clusters in the following way. The algorithm set a new distance threshold $d_{i+1} = 2d_i$, and then, iteratively, it merges together the clusters whose centers are at distance $\leqslant d_{i+1}$, until the are no such clusters. Note that it may happen that for a given distance threshold there is no merge. In this case the algorithm doubles again the threshold, until there are no more than k clusters. Then, the update stage accepts new points from the stream. A new point p of the stream is placed in a cluster centered in any c such that $\mathrm{dist}(p, c) \leqslant 2d_{i+1}$. If there is no such cluster, then p is placed in a new singleton cluster. Note that this implementation of the update stage builds clusters complying with the invariants of phase $i+1$. The update stage continues until either there are no more points in the stream or the algorithm has built $k+1$

clusters. In the second case, phase i terminates and phase $i+1$ starts with a new merging stage.

The following result is proved in [Cha+04].

Proposition 3. *The above streaming algorithm is a 8-approximation algorithm for the k-center problem, runs in a single pass, and requires $O(k)$ memory.*

In Chapter 5, we will use the doubling algorithm as a basis to build efficient streaming algorithms for the problem of diversity maximization.

Chapter 3

Diameter approximation

A fundamental primitive for graph analytics is the computation of a graph's diameter. This primitive is computationally intensive, therefore resorting to parallelism is inevitable as the size of the graph grows. Moreover, computing exactly the diameter of large graphs is essentially as expensive as computing the All-Pairs Shortest Paths, which is unfeasible on very large graphs. Therefore, when dealing with large graphs, an approximation to the diameter is the best that can be hoped for. However, state of the art parallel strategies for diameter estimation are either space inefficient or incur long critical paths. These strategies are thus unfeasible for dealing with huge graphs, especially on platforms such as MapReduce, where only linear space in the size of the graph is allowed and long critical paths are penalized by the high intrinsic cost of each round.

In this chapter, we develop clustering-based parallel algorithms for diameter estimation on both weighted and unweighted undirected graphs, suitable for implementation on MapReduce-like platforms. The high level idea of our algorithms is the following. Given a (weighted) graph G, we derive a smaller *auxiliary graph*, whose size is tuned to fit into the local memory M_L of a single reducer. This auxiliary graph is built by clustering the input graph: each cluster will be represented by a single node in the auxiliary graph, and edges between nodes are weighted so as to summarize the shortest paths between the corresponding clusters. The approximation of the input graph's diameter is then obtained as a simple function of the exact diameter of the auxiliary graph, which can be computed efficiently because of its reduced size. We prove that the approximation ratio is bounded, and we verify experimentally that the actual approximation ratio is much better than what predicted by the theory.

Efficiently clustering a graph on MapReduce is tricky, because of the very limiting nature of the platform: only memory linear in the size of the graph is allowed, and distance computations are very expensive because of their inherent high round complexity. Therefore, a substantial part of this chapter is devoted to developing clustering algorithms that tackle the constraints of MapReduce (Section 3.2). After

that, we will see how to relate the diameter of the auxiliary graph to the one of the input graph in 3.3. We will also study how to implement efficiently this algorithm on MapReduce (Section 3.4), and how to optimize for the unweighted case (Section 3.5). We will then present an experimental evaluation of the MapReduce implementation of our algorithms in Section 3.7.

The results presented in this chapter were first presented at the Symposium on Parallelism in Algorithms and Architectures [Cec+15], for unweighted graphs, and at the International Parallel and Distributed Processing Symposium [Cec+16b] for weighted graphs.

3.1 Related work

3.1.1 Algorithms based on All-Pairs Shortest Paths

For general graphs with arbitrary weights, an approach for the exact diameter computation requires the solution of the All-Pairs Shortest Paths (APSP) problem. There are many classic algorithms to solve this problem. Floyd-Warshall's algorithm runs in $\Theta\left(n^3\right)$ time and requires $\Theta\left(n^2\right)$ space. On graphs with non-negative weights, running Dijkstra's algorithm (with Fibonacci heaps) from each vertex requires $O\left(mn + n^2 \log n\right)$ time, which is better than $O\left(n^3\right)$ for sparse graphs. Moreover, by keeping track of only the largest distance found at each invocation, the space requirement drops to $\Theta(n)$. On unweighted graphs, Dijkstra's algorithm can be replaced by a simple Breadth First visit, with the same space requirements and an overall running time of $O(mn)$. However, Dijkstra's algorithm and the BFS are difficult to parallelize. Indeed, one may run the sequential algorithm from each node on separate processors in parallel, however this approach is not applicable when the graph does not fit in the memory of a single processor. There are some works improving on the $O\left(mn + n^2 \log n\right)$ bound provided by Dijkstra's algorithm. In [Pet04], an APSP algorithm with running time of $O\left(mn + n^2 \log\log n\right)$ has been introduced.

Finally, another classic approach to solve the APSP problem is to repeatedly square the adjacency matrix of the graph. Using fast matrix multiplication algorithms, this approach can be implemented in $O\left(n^{2.3727} \log n\right)$ time [Wil12]. By applying a clever recursive decomposition of the problem, we can drop the repeated squaring, saving a logarithmic factor in the time complexity [AHU74, pp. 201-206]. The space requirement in any case is $O\left(n^2\right)$, which rules out matrix-based approaches in the context of large graphs.

All the above algorithms work on both directed and undirected graphs. For undirected graphs, a faster algorithm with running time $O(m\, n\, \alpha(m, n))$, where $\alpha(m, n)$ is the inverse-Ackermann function, is presented in [PR02].

The drawback of these approaches is that they are either space inefficient or difficult inherently sequential, making them not applicable on very large graphs.

3.1.2 Algorithms based on Single-Source Shortest Path

A very simple approximation algorithm, in both the weighted and unweighted case, consists in picking an arbitrary node of the graph, and finding the farthest

node from it by solving the Single-Source Shortest Paths problem (SSSP for short). It is easy to see that this is a 2-approximation for the diameter of the graph. This approach spurred a line of research in which a few SSSP instances are solved, starting from carefully selected nodes, to get good approximation of the diameter. Magnien et al. [MLH09] considered the case of unweighted graphs, and studied empirically a simple strategy which achieves very good approximation factors in practice. This algorithm, called 2-Sweep, performs a first BFS from an arbitrary vertex, and a second one from the farthest reachable node, returning the maximum distance found. The theoretical approximation factor achieved by this approach is still 2, like a single BFS, but experiments showed that on some real world graphs the actual value found is much closer to the optimum. Building on this idea, Crescenzi et al. developed the iFub algorithm for unweighted graphs [Cre+13], and the DiFub algorithm for weighted graphs [Cre+12], both of which compute the diameter of a given graph exactly. In the worst case, these algorithms require to solve the SSSP problem for every node. However, with an extensive experimental evaluation the authors showed that only a small number of BFSs is needed in practice.

The drawback affecting these approaches, which are very effective when the graph fits in the memory of a single machine, is the difficulty of parallelizing SSSP computations, especially on loosely coupled architectures such as MapReduce.

3.1.3 Algorithms based on the neighbourhood function

The neighbourhood function $N_G(h)$ of a graph G, also called the *hop plot* [FFF99], is the number of pairs of nodes that are within distance h, for every $h \geqslant 0$ [PGF02]. The neighbourhood function can be used for several purposes, including diameter computation [PGF02; BRV11], facility location [Gar+15], and centrality computation [BRV11]. Computing the neighbourhood function exactly requires to store, for each node and each h, the set of nodes reachable within distance h. This results in an overall $O(n^2)$ space requirement, which is impractical for large graphs.

Palmer et al. [PGF02] introduced the *Approximate Neighbourhood Function* algorithm (abbreviated ANF), where Flajolet-Martin probabilistic counters [FM83] are used to count the nodes reachable within distance h. Each node is associated with a counter, updated iteratively: in iteration i, the counter approximates the number of nodes reachable in i hops. At the end of iteration i, the information of all the counters is combined to obtain $N_G(i)$. Each counter requires only $O(\log n)$ bits, therefore lowering to $O(n \log n)$ the memory required to compute the neighbourhood function. This result has been further improved by Boldi et al. [BRV11] by replacing Flajolet-Martin counters with HyperLogLog counters [Fla+08]. HyperLogLog counters require only $O(\log \log n)$ bits to count the number of reachable nodes, so the overall memory requirement drops to $O(n \log \log n)$.

These neighbourhood function approximation algorithms were originally developed for shared memory machines. A MapReduce implementation of ANF, called HADI, has been presented by Kang et al. [Kan+11]. This MapReduce algorithm suffers mainly of two drawbacks: a) the memory required is slightly superlinear, and b) it requires a number of rounds linear in the diameter of the graph. Moreover, all three of ANF, HyperANF and HADI work only on unweighted graphs. In [BV13], the authors of HyperANF extended their approach to graphs with in-

teger weights on the *nodes*, which are arguably less common than edge-weighted graphs.

Cohen [Coh15] presented an overview of so-called all-distance sketches, which include the probabilistic counters used by the above neighbourhood function estimators. This unified overview is instrumental to the introduction of HIP estimators, which have half the variance of the approaches reviewed above when used to the neighbourhood function. Moreover, Cohen describes a way to compute all-distance sketches (and hence estimate the neighbourhood function) on weighted graphs. We note that, in both the weighted and unweighted case, the estimation of the neighbourhood function requires several single-source shortest paths computations, which makes these approaches unsuitable for the application in MapReduce.

3.1.4 Algorithms based on clustering

In this chapter's introduction, we mentioned how our approach is based on the construction of a small auxiliary graph by means of clustering. There are other approaches to approximate the diameter of a graph that are based on a similar approach.

In the external memory model, where performing even a single BFS has a very high I/O complexity, Meyer [Mey08] proposed an algorithm to approximate the diameter of unweighted graphs. The algorithm of [Mey08] seeks to derive, with a low I/O complexity, a graph that can fit into main memory, on which to compute an approximation to the diameter. The algorithm features a design parameter k, which can be employed to obtain a tradeoff between I/O complexity and approximation quality. The basic idea is the following. First, the algorithm selects nodes at random with probability $1/k$, marking the selected ones as *master nodes*, which are the representative centers of distinct clusters. Then, it grows clusters around master nodes "in parallel": iteratively, each master node tries to capture all unvisited neighbours of its current cluster. The clusters are non-overlapping, therefore if multiple masters try to capture the same node, only an arbitrary one succeeds. The cluster-growing phase terminates when all the nodes are assigned to a cluster, and is followed by a postprocessing needed to ensure that each node is at most $k-1$ hops away from the closest master. This is done by performing an Euler tour of an arbitrary spanning tree of G, and marking an additional master node every k nodes in the Euler tour. This postprocessing is inherently sequential and is what prevents this algorithm to be applied in MapReduce. After the graph has been clustered, the algorithm builds an auxiliary graph with a vertex for each cluster, and edges suitably weighted to approximately represent the distances between master nodes. By computing the diameter of this auxiliary graph, the algorithm provides a $O\left(\sqrt{k}\right)$ approximation to the input graph's diameter. The I/O complexity is $O(n \cdot \sqrt{\log k/(kB)} + k \cdot \mathrm{scan}(n) + \mathrm{sort}(n))$, where B is the number of elements that can be transferred within a single I/O operation, $\mathrm{scan}(n) = \Theta(n/B)$ is the number of I/O operations needed to scan an input of n elements, and $\mathrm{sort}(n) = \Theta(n/B \ \log_{n/B}(n/B))$ is the I/O complexity of sorting n elements. Therefore, setting a lower value of k to get a better approximation requires performing more I/O operations, intuitively because with fewer clusters each cluster requires to reach farther distances before the entire graph is covered. An extension of this

approach to weighted graphs in the external memory context has been proposed in [AMV12], which however is analyzed only experimentally and lacks provable approximation bounds.

Clustering is also used to approximate shortest paths between nodes, which can indirectly be used to approximate the diameter. In particular, in [Coh98] the notion of (β, W)-cover for a weighted graph G has been introduced. A (β, W)-cover is essentially a decomposition of the graph into overlapping clusters: each node is allowed to belong to $O\left(\beta n^{1/\beta} \log n\right)$ distinct clusters and for any two nodes at weighted distance at most W in the graph, there is a cluster containing both. A (β, W)-cover is obtained by growing clusters of decreasing radii from successive batches of centers. In [Coh00], the author presents a PRAM algorithm which uses (beta,W)-covers to approximate shortest-path distances. For sparse graphs with $m \in \Theta(n)$, this algorithm features $O\left(n^{\delta}\right)$ depth, for any fixed constant $\delta \in (0, 1)$, but incurs a polylogarithmic space blow-up due to the use of (β, W)-covers. The algorithms are rather involved and communication intensive, hence, while theoretically efficient, in practice they may run slowly when implemented on distributed-memory clusters of loosely-coupled servers, where communication overhead is typically high.

3.1.5 Δ-stepping

In a seminal work, Meyer and Sanders proposed a PRAM algorithm, called Δ-*stepping*, for the SSSP problem [MS03]. This algorithm exercises a tradeoff between work-efficiency and parallel time through a parameter Δ. The diameter approximation algorithm we present in this chapter will be partly based on ideas introduced in [MS03], therefore in this section we briefly describe the Δ-stepping algorithm. Moreover, recall that a single run of any SSSP yields a 2-approximation to the graph's diameter, hence the Δ-stepping algorithm can be regarded as a parallel diameter approximation algorithm.

Let $G = (V, E, w)$ be a weighted graph, and $s \in V$ be the source node for the SSSP computation. Furthermore, define $\mathrm{tent}(v)$ the *tentative distance* of v from s, that is an upper bound to $\mathrm{dist}(s, v)$. Tentative distances are improved with *edge relaxations* of the kind used in the classical Bellman-Ford's algorithm: for an edge $e = \{u, v\}$, the relaxation of e consists in setting $\mathrm{tent}(u) = \min\{\mathrm{tent}(u), \mathrm{tent}(v) + w(e)\}$ and $\mathrm{tent}(v) = \min\{\mathrm{tent}(v), \mathrm{tent}(u) + w(e)\}$.

The Δ-stepping algorithm maintains a one-dimensional array $\mathcal{B}$ of *buckets*, such that $B_i \in \mathcal{B} = \{v \in V : \mathrm{tent}(v) \neq \infty \wedge \mathrm{tent}(v) \in [i \cdot \Delta, (i+1) \cdot \Delta)\}$. In this setting, the parameter Δ defines the *bucket width*. Initially, s has $\mathrm{tent}(s) = 0$, whereas all the other nodes have tentative distance ∞. Therefore, initially B_0 contains only s with tentative distance 0. The algorithm then works in *phases*. In phase i, the algorithm removes all the nodes from bucket B_i and relaxes all the edges such that $w(e) \leqslant \Delta$ (these edges are called *light*). This operation may result in new nodes being inserted in bucket B_i. Furthermore, nodes previously deleted from bucket B_i are reinserted in B_i if their tentative distance is updated. This operation is repeated until bucket B_i is empty. At this point the algorithm proceeds to the relaxation of *heavy edges*, that is edges of weight strictly larger than Δ. Phase i terminates with this relaxation, and the algorithm moves on to bucket B_{i+1}.

The performance of the Δ-stepping algorithm is parameterized by a parameter ℓ_Δ, which in turn is tied to the weights of the input graph. Specifically, ℓ_Δ is one plus the maximum number of edges in a shortest path of length $\leqslant \Delta$.

The following result is proved in [MS03], and characterizes the complexity with two further parameters: n_Δ is the number of node pairs connected by a path of length $\leqslant \Delta$, and m_Δ is the number nodes that can be reached by extending any path of length $\leqslant \Delta$ with a light edge.

Theorem 2 ([MS03, Theorem 16])**.** *The Δ-stepping algorithm can be implemented to run on the CRCW-PRAM in time*

$$O\left(\frac{L}{\Delta}\ell_\Delta \log n\right)$$

with work

$$O\left(n + m + n_\Delta + m_\Delta + \frac{L}{\Delta}\ell_\Delta \log n\right)$$

Note that in the expressions of the parallel time and the work there are terms both increasing and decreasing with Δ, which makes difficult to choose the best value for Δ. Indeed, finding the right value of Δ for a given input graph is one of the most cumbersome parts of applying the Δ-stepping algorithm. We observe that the parallel time is $\Omega(\Psi(G))$, where $\Psi(G)$ is the unweighted diameter of G as introduced in Section 2.1. This has important implications when Δ-stepping is ported to MapReduce, since it translates in a number of rounds which is at least linear in the unweighted diameter.

In the original paper [MS03], the authors propose a preprocessing of the graph to speed up the execution. This preprocessing involves adding so-called *shortcuts*: the graph is augmented with new edges between nodes at distance $\leqslant \Delta$. While this preprocessing can be shown to turn the dependence of the running time on ℓ_Δ from multiplicative to additive, it also requires a potentially quadratic space blow-up. In the context of MapReduce, where memory is at premium and only linear space is allowed, this optimization may not be convenient.

3.2 Clustering algorithm

In this section we discuss a parallel algorithm for clustering weighted graphs, which, as we mentioned before, is a fundamental building block of our diameter approximation strategy. The challenge is to perform cluster growth by exploiting parallelism while, at the same time, limiting the weight of the edges considered in each growing step: the goal is to avoid increasing excessively the weighted radius of the clusters, which influences the quality of the subsequent diameter approximation. This must be done using a small number of rounds (i.e. the unweighted radius must be small) and only linear space in the size of the graph.

To address these issues, we grow clusters in stages, where in each stage a new randomly selected batch of cluster centers is added to the current clustering and the number of uncovered nodes halves with respect to the preceding stage. The idea behind such a strategy is to force more clusters to grow in poorly connected regions of the graph while keeping both the total number of clusters and the maximum cluster radius under control. Note that we cannot afford to grow a cluster boldly

by adding all nodes connected to its frontier at once, since some of these additions may entail heavy edges, resulting in an increase of the weighted cluster radius. To tackle this challenge, we use ideas akin to those employed in the Δ-stepping parallel SSSP algorithm proposed in [MS03] and which we briefly reviewed in Section 3.1.5. In particular, we limit a cluster's growth by imposing a threshold Δ on its radius. Unlike the Δ-stepping algorithm, however, this threshold is not fixed a priori but automatically tuned to a quasi-optimal value during the clustering process.

Before proceeding to the description of the subroutines used in the clustering and the algorithm itself, we will introduce some notation that will be used for the rest of the chapter. Let $G = (V, E, w)$ be a connected undirected weighted graph with n nodes, m edges, and a function w which assigns a positive real weight $w(e) \in \mathbb{R}^+$ to each edge $e \in E$. We make the reasonable assumption that the edge weights are polynomial in n, that is $w_{max}/w_{min} = O\left(n^h\right)$, for some $h > 0$, where w_{max} and w_{min} are, respectively, the maximum and minimum edge weight. As we defined in Chapter 2, the distance between two node $u, v \in V$ is denoted with $\mathrm{dist}(u, v)$ and is the weight of a minimum-weight path between u and v. Moreover, the diameter $\Phi(G)$ of the graph is the maximum distance between any two nodes. With the following definition we introduce the concept of τ-clustering in weighted graphs.

Definition 6. *For any positive integer* $\tau \leqslant n$*, a* τ-clustering *of* G *is a partition* $C = \{C_1, C_2 \ldots, C_\tau\}$ *of* V *into* τ *subsets called* clusters*. Each cluster* C_i *has a distinguished node* $c_i \in C_i$ *called* center*, and a* radius $r(C_i) = \max_{v \in C_i}\{\mathrm{dist}(c_i, v)\}$*. The* radius of a τ-clustering C *is* $r(C) = \max_{1 \leqslant i \leqslant \tau}\{r(C_i)\}$*. Finally, denote by* $R_G^{opt}(\tau)$ *the minimum among the radii of all* τ*-clusterings of* G*.*

The algorithm that we describe in the next sections maintains with each node $u \in V$ a *state* consisting of two variables (c_u, d_u): initially, c_u is undefined and $d_u = \infty$. Whenever u is assigned to a cluster, then c_u is set to the cluster center and d_u is set to an upper bound to $\mathrm{dist}(c_u, u)$. In particular, if u is a cluster center, then $c_u = u$ and $d_u = 0$.

3.2.1 Subroutines

We now present some subroutines that we will use to build our clustering algorithm.

Δ-growing step. Before presenting the algorithm, we introduce some technical details. Let Δ be a real-valued parameter which we use as a guess for the radius of the clustering. As in [MS03], we call an edge *light* if its weight is $\leqslant \Delta$, and *heavy* otherwise. The clustering algorithm repeatedly applies, in parallel, edge relaxations of the kind described in Section 3.1.5. More precisely, we define the following Δ-*growing step*: for each node u with $d_u < \Delta$ and for each light edge (u, v), in parallel, if $d_u + w(u, v) \leqslant \Delta$ and $d_v > d_u + w(u, v)$ then the status of v is updated by setting $d_v = d_u + w(u, v)$, and $c_v = c_u$. In case more than one node u can provide an update to the status of v, the algorithm performs the update that yields the smallest value of d_v and, secondarily, the one caused by the node u such that c_u has smallest index.

Algorithm 2: CONTRACT$(G = (V, E))$

```
V' ← ∅;
E' ← ∅;
for e = {u, v} ∈ E do
    if c_u = nil ∧ c_v = nil then
        V' ← V' ∪ {u, v};
        E' ← E' ∪ e
    end
    else if c_u ≠ nil ∧ c_v = nil then▷ The same applies for c_u = nil ∧ c_v ≠ nil
        V' ← V' ∪ {c_u, v};
        e' ← {c_u, v} with weight w(e);
        E' ← E ∪ e';
    else
        The edge has both endpoints in the cluster, ignore it so it does not appear in
         the output.
    end
end
for u, v ∈ V' such that are multiple edges between u and v do
    Remove from E' all edges {u, v} except the minimum weight one.
end
return G' = (V', E') ;
```

Graph contraction. Suppose that a sequence of Δ-growing steps is performed starting from some set of centers $X \subseteq V$. After the execution of these steps, we can contract the graph G as follows (Procedure CONTRACT(G), see Algorithm 2). For each center $c \in X$, all nodes $u \in V$ with $c_u = c$ are removed, except c itself. For each edge (u, v) the algorithm performs one of the following transformations:

- if both c_u and c_v are defined, the edge is removed;
- if both c_u and c_v are undefined, the edge is left unchanged;
- if c_u is defined and c_v is undefined, the edge is replaced by a new edge (c_u, v) of weight $w(u, v)$. Note that this may result in multiple edges between c_u and v. In this case, the algorithm retains the one with minimum weight.

3.2.2 Clustering algorithm

We now present algorithm CLUSTER(G, τ), which, given in input a graph G with n nodes and m edges, and an integer parameter τ, builds a clustering of G with $O(\tau \log^2 n)$ clusters. The algorithm, whose pseudocode is given in Algorithm 3, grows clusters progressively in a number of stages, until all the nodes of the graph are covered. In each stage, which corresponds to an iteration of the outer while loop, a sequence of Δ-growing steps is executed, each with the goal of including into the current clusters at least half of the nodes that are still uncovered but are reachable from some cluster through a path of weight at most Δ. The set X of centers of the current clusters includes clusters partially grown in previous stages (if any), which are now contracted and represented only by their centers (set C_i), and a set of $O(\tau \log n)$ centers randomly selected from the uncovered nodes. In each stage, the algorithm guesses geometrically increasing values of Δ, starting

Algorithm 3: CLUSTER(G, τ)

$\Delta \leftarrow \min\{w(u,v) \ : \ (u,v) \in E\}$
$\gamma \leftarrow 4 \ln 2$
$C_1 \leftarrow \emptyset$ ▷ current set of cluster centers

$G_1(V_1, E_1) \leftarrow G(V, E)$
$i \leftarrow 1$
while $|V_i \setminus C_i| \geqslant 8\tau \log n$ **do**
  Make $v \in V_i \setminus C_i$ a new center with probability $\frac{(\gamma \tau \log n)}{|V_i \setminus C_i|}$
  $X \leftarrow C_i \cup \{\text{newly selected centers}\}$
  foreach $u \in V_i$ **do**
    if $u \in X$ **then** $(c_u, d_u) \leftarrow (u, 0)$ **else** $(c_u, d_u) \leftarrow (\text{nil}, \infty)$
  end
  $V' \leftarrow \emptyset$ ▷ Set of nodes covered in this iteration

  while $|V'| < |V_i \setminus C_i|/2$ **do**
    repeat
      perform a Δ-growing step on $\bar{G}$
      $V' \leftarrow \{u \in V_i \setminus C_i \ : \ d_u \leqslant \Delta\}$
    until *(no state is updated) or* $(|V'| \geqslant |V_i \setminus C_i|/2)$
    if $|V'| < |V_i \setminus C_i|/2$ **then** $\Delta \leftarrow 2\Delta$
  end
  Assign each $u \in V'$ to the cluster centered at c_u
  $G_{i+1}(V_{i+1}, E_{i+1}) \leftarrow$ CONTRACT(G_i)
  $C_{i+1} \leftarrow X$
  $i \leftarrow i + 1$
end
Assign each $u \in V_i \setminus C_i$ to a new singleton cluster centered at u
▷ V_i is the final set of cluster centers

from a suitable initial value, until the coverage goal can be attained. When few nodes are left uncovered, these are added as singleton clusters and the algorithm terminates. Observe that at most $O(\log n)$ stages are executed and each contributes an additive factor Δ to the clustering radius. We will show below that the largest guess for Δ will be $O\left(R_G^{opt}(\tau)\right)$, with high probability.

Observe that when the algorithm terminates, each node $u \in V$ is assigned to a cluster centered at some node c_u. Let Δ_{end} denote the value of Δ at the end of the execution of CLUSTER(G, τ). In the following lemma, we show that with high probability Δ_{end} does not exceed $R_G^{opt}(\tau)$ by more than a constant factor.

Lemma 1. $\Delta_{end} = O\left(R_G^{opt}(\tau)\right)$, *with high probability.*

Proof. Consider an arbitrary iteration of the outer while loop and let $G_i = (V_i, E_i)$ be the (contracted) graph on which cluster growth is performed during the iteration. Let $C_i \subseteq V_i$ be the nodes of G_i representing clusters grown in previous iterations. Clearly, we have that $|V_i - C_i| \geqslant 8\tau \log n$. Refer to the nodes of $V_i - C_i$ as *uncovered nodes*. We now show that, with probability at least $1 - 1/n$, in G_i at least half of the uncovered nodes can be reached by the new centers selected in the iteration or

by nodes of C_i with paths of weight at most $2R_G^{opt}(\tau)$ traversing only uncovered nodes.

Let $\bar{C}$ be a τ-clustering of the whole graph with optimal radius $r(\bar{C}) = R_G^{opt}(\tau)$. To avoid confusion, we refer to its clusters as $\bar{C}$-*clusters*, while simply call *clusters* those grown by our algorithm. Consider the $\bar{C}$-clusters that include some uncovered node. Among these $\bar{C}$-clusters, those that contain less than $|V_i - C_i|/(2\tau)$ uncovered nodes account for a total of less than $\tau|V_i - C_i|/2\tau = |V_i - C_i|/2$ such nodes. We call *large* the $\bar{C}$-clusters that contain $|V_i - C_i|/(2\tau)$ or more uncovered nodes. Therefore, large $\bar{C}$-clusters account for more than $|V_i - C_i|/2$ uncovered nodes altogether. Let $\bar{C}_\ell$ be any such large $\bar{C}$-cluster. By the choice of γ in the probability for center selection, we have that with probability $\geqslant (1 - 1/n^2)$ at least one uncovered node $c' \in \bar{C}_\ell$ is selected as a new center. Since, for every uncovered node $v \in \bar{C}_\ell$, there is a path in G from c' to v through $\bar{c}_\ell$ of weight $w \leqslant 2R_G^{opt}(\tau)$, there must be a path in G_i from c' to v of weight at most w in G_i. Note that the suffix of this path starting from the last cluster center has weight $\leqslant w$ and traverses only unconvered nodes. The desired property follows by applying the union bound over all large $\bar{C}$-clusters.

Since clusters are grown from the newly selected centers as well as from the nodes of C_i, any value $\Delta \geqslant 2R_G^{opt}(\tau)$ guarantees that half of the nodes in $V_i - C_i$ are covered by clusters. Consequently, Δ can never be doubled beyond $4R_G^{opt}(\tau)$. The lemma follows by applying the union bound over all iterations. □

Recall that with ℓ_X we denote the maximum number of edges in any shortest path of length $\leqslant X$, for a given graph. Moreover, observe that $\ell_{cX} = O(\ell_X)$, for any constant $c > 0$. We can now state the main result for this section.

Theorem 3. *Let τ be a positive integer. With high probability, CLUSTER(G, τ) returns a clustering of G with $O(\tau \log^2 n)$ clusters of radius $O\left(R_G^{opt}(\tau) \log n\right)$, and performs $O\left(\ell_{R_G^{opt}(\tau)} \log n\right)$ Δ-growing steps, with $\Delta = O\left(R_G^{opt}(\tau)\right)$.*

Proof. By Chernoff's bound each iteration of the outer **while** loop selects $O(\tau \log n)$ new cluster centers with high probability. Hence, the bound on the number of clusters follows applying the union bound over the $O(\log n)$ iterations of this loop. As for the bound on the clustering radius, we observe that the aggregate number of iterations of the inner **while** loop is at most $\log n + \log(\Delta_{end}/w_{min})$. Moreover, note that $\Delta_{end} = O(\Phi(G)) = O(n \cdot w_{max})$. By the assumption of polynomiality of edge weights, we therefore have that $\log(\Delta_{end}/w_{min}) = O(\log(n \cdot w_{max}/w_{min})) = O(\log n)$. By virtue of Lemma 1 and the discussion above, the number of iterations of the inned **while** loop is thus $O(\log n)$. The bound follows by observing that each such iteration increases the radius of cluster by an additive term at most $\Delta_{end} = O\left(R_G^{opt}(\tau)\right)$.

Finally, observe that in every iteration of the inner **while** loop the number of Δ-growing steps executed is at most $\ell_\Delta \leqslant \ell_{\Delta_{end}}$. By the properties of edge relaxations, after $\ell_{\Delta_{end}}$ growing steps all nodes at distance less than Δ from some center of X have been reached by the closest center in X with a minimum-weight path, hence their state cannot be further updated. Therefore, the final number of Δ-growing steps will be $O(\ell_{\Delta_{end}} \log n) = O\left(\ell_{R_G^{opt}(\tau)} \log n\right)$, with high probability. □

Algorithm 4: CONTRACT2$(G = (V, E))$

```
V' ← ∅;
E' ← ∅;
for e = {u, v} ∈ E do
    if c_u = nil ∧ c_v = nil then
        V' ← V' ∪ {u, v};
        E' ← E' ∪ e
    else if c_u ≠ nil ∧ c_v = nil then ▷ The same applies for c_u = nil ∧ c_v ≠ nil
        V' ← V' ∪ {c_u, v};
        e' ← {u, v};
        if w(e) ⩽ 2R_G^CL(τ) then w(e') ← d_u + w(u, v) − 2R_G^CL(τ) ;
        else w(e') ← w(u, v) ;
        E' ← E ∪ e';
    else
        The edge has both endpoints in the cluster, ignore it so it does not appear in
         the output.
    end
end
for u, v ∈ V' such that are multiple edges between u and v do
    Remove from E' all edges {u, v} except the minimum weight one.
end
return G' = (V', E') ;
```

The algorithm we presented above can be extended to disconnected graphs. Let G be a graph with $h \geqslant 1$ connected components. It is easy to see that for any $\tau \geqslant h$, algorithm CLUSTER(G, τ) works correctly with the same guarantees stated in Theorem 3.

3.3 Diameter approximation algorithm

We now present an algorithm to estimate the diameter of a weighted graph as a function of the diameter of a suitable (much smaller) auxiliary graph derived from a clustering obtained through a refined version of the strategy devised in the previous section. As we will clarify in the analysis, we introduce this refinement to achieve a provable bound on the approximation guarantee, by ensuring that not too many clusters have the potential to reach small neighborhoods of the graph. Algorithm CLUSTER2(G, τ), whose pseudocode is given in Algorithm 5, builds the required clustering in two steps. First, it computes the radius $R_G^{CL}(\tau)$ of the clustering returned by CLUSTER(G, τ). Second, it executes $\log n$ iterations where it selects uncovered nodes as new cluster centers with probability doubling at each iteration. In the i-th iteration, both previous and new clusters are grown using $2R_G^{CL}(\tau)$-growing steps until *all* uncovered nodes at distance at most $2R_G^{CL}(\tau)$ from them are reached. At the end of the iteration, the graph is contracted using Procedure CONTRACT2, whose pseudocode is reported in Algorithm 4. This procedure is similar to Procedure CONTRACT used in CLUSTER, with the difference that each original edge (u, v) of weight $w(u, v) \leqslant 2R_G^{CL}(\tau)$ and such that c_u is defined and c_v is undefined, is replaced by a new edge (c_u, v) with rescaled weight $d_u + w(u, v) - 2R_G^{CL}(\tau)$. Note that this rescaling results in a positive weight, because when c_u is defined and c_v

Algorithm 5: CLUSTER2(G, τ)

Let $R_G^{CL}(\tau)$ be the radius of the clustering returned by CLUSTER(G, τ)
$C_1 \leftarrow \emptyset$ ▷ `current set of cluster centers`

$G_1(V_1, E_1) \leftarrow G(V, E)$
for $i \leftarrow 1$ **to** $\log n$ **do**
  Select $v \in V_i - C_i$ as a new center independently with probability $2^i/n$
  $X \leftarrow C_i \cup \{\text{newly selected centers}\}$
  foreach $u \in V_i$ **do**
    if $u \in X$ **then** $(c_u, d_u) \leftarrow (u, 0)$ **else** $(c_u, d_u) \leftarrow (\text{nil}, \infty)$
  end
  $\Delta \leftarrow 2R_G^{CL}(\tau)$
  repeat
    perform a Δ-growing step on $\bar{G}$
  until *no state is updated*
  $G_{i+1}(V_{i+1}, E_{i+1}) \leftarrow$ CONTRACT2(G_i)
  $C_{i+1} \leftarrow X$
end

is undefined, we have that $d_u + w(u, v) > 2R_G^{CL}(\tau)$, otherwise v would have been included in the cluster centered in c_u.

The following lemma analyzes the quality of the clustering returned by CLUSTER2(G, τ) and upper bounds the number of growing steps performed.

Lemma 2. *Let τ be a positive integer. With high probability,* CLUSTER2(G, τ) *computes an $O\left(\tau \log^4 n\right)$-clustering of radius $R_G^{CL2}(\tau) = O\left(R_G^{opt}(\tau) \log^2 n\right)$ by performing $O\left(\ell_{R_G^{opt}(\tau)\log n} \log n\right)$ Δ-growing steps with $\Delta = O\left(R_G^{opt}(\tau) \log n\right)$.*

Proof. By Theorem 3 we know that with high probability the invocation of CLUSTER(G, τ) at the beginning of the algorithm computes a K-clustering, with $K = O\left(\tau \log^2 n\right)$, of radius $R_G^{CL}(\tau) = O\left(R_G^{opt}(\tau) \log n\right)$. In what follows, we condition on this event. Then, the bounds on the number of growing steps and on $R_G^{CL2}(\tau)$ are straightforward. For $\gamma = 4/\log_2 e$, define H as the smallest integer such that $2^H/n \geqslant (\gamma K \log n)/n$, and let $t = \log n - H$. We now show that the number of original nodes of G not yet reached by any cluster decreases at least geometrically at each iteration of the **for** loop after the H-th one. Recalling that $V_{H+i} - C_{H+i}$ is the set of original nodes of G that at the beginning of Iteration $H + i$ have not been reached by any cluster, for $1 \leqslant i \leqslant t$, define the event $E_i =$*"at the beginning of Iteration $H + i$, $V_{H+i} - C_{H+i}$ contains at most $n/2^{i-1}$ nodes"*. We now prove that the event $\cap_{i=1}^{t} E_i$ occurs with high probability. Observe that:

$$\Pr\left(\cap_{i=1}^{t} E_i\right) = \Pr(E_1) \prod_{i=1}^{t-1} \Pr(E_{i+1} | E_1 \cap \cdots \cap E_i)$$
$$= \prod_{i=1}^{t-1} \Pr(E_{i+1} | E_1 \cap \cdots \cap E_i),$$

since E_1 clearly holds with probability one. Consider an arbitrary i, with $1 \leqslant i < t$, and assume that $E_1 \cap \cdots \cap E_i$ holds. We prove that E_{i+1} holds with high probability. Since E_i holds, we have that at the beginning ofIteration $H+i$, the number of nodes in $V_{H+i} - C_{H+i}$is at most $n/2^{i-1}$. Clearly, if $|V_{H+i} - C_{H+i}| \leqslant n/2^i$ then E_{i+1}trivially holds with probability one. Thus, we consider only the case

$$\frac{n}{2^i} < |V_{H+i} - C_{H+i}| \leqslant \frac{n}{2^{i-1}}.$$

In order to show that E_{i+1} holds also in this case, we resort to the same argument used in the proof of Lemma 1. Let $\bar{C}$ be a K-clustering of the whole graph with optimal radius $R_G^{opt}(K)$ and observe that $R_G^{opt}(K) \leqslant R_G^{CL}(\tau)$. To avoid confusion, we refer to its clusters as $\bar{C}$-*clusters*, while simply call *clusters* those grown by Cluster2. Consider the $\bar{C}$-clusters that include some nodes of $V_{H+i} - C_{H+i}$, and call one such cluster *large* if it contains at least

$$\frac{|V_{H+i} - C_{H+i}|}{2K} > \frac{n\gamma \log n}{2^{H+i+1}}$$

nodes of $V_{H+i} - C_{H+i}$. This implies that the large $\bar{C}$-clusters contain, altogether, at least half of the nodes of $V_{H+i} - C_{H+i}$. Moreover, by the choice of γ, it is easy to argue that, with probability $\geqslant (1 - 1/n)$, at least one new center is selected from each large $\bar{C}$-cluster in Iteration $H + i$. Consider now an arbitrary large $\bar{C}$-cluster centered at $\bar{c}_\ell$ and let $c' \in \bar{C}_\ell$ be a new center selected from this cluster in the iteration. For every $v \in \bar{C}_\ell \cap (V_{H+i} - C_{H+i})$ there is a path in G from c' to v (through $\bar{c}_\ell$) of weight $w \leqslant 2R_G^{CL}(\tau)$, hence there must be a path in G_i from c' to v of weight at most w. It then follows that node v will be covered by some cluster in Iteration $H + i$. Consequently, in the iteration at least half of the nodes of $V_{H+i} - C_{H+i}$ will be covered by clusters, with probability at least $1 - 1/n$.

By multiplying the probabilities of the $O(\log n)$ conditioned events, we conclude that event $\cap_{i=1}^{t} E_i$ occurs with high probability. Note that in the last iteration (Iteration $H + t$) all uncovered nodes are selected as centers with probability 1, and, if $\cap_{i=1}^{t} E_i$ occurs, these are $O(K \log n)$. Now, one can easily show that, with high probability, in the first H iterations, $O\left(K \log^2 n\right)$ clusters are added and, by conditioning on $\cap_{i=1}^{t+1} E_i$, at the beginning of each Iteration $H + i$, $1 \leqslant i \leqslant t$, $O(K \log n)$ new clusters are created, for a total of $O\left(K \log^2 n\right) = O\left(\tau \log^4 n\right)$ clusters. □

Observe that for fixed τ, the clustering returned by Cluster2 has a larger number of clusters and a weaker guarantee on its radius than the clustering returned by Cluster. As such, Cluster2 does not appear to be a very desirable clustering strategy in itself. However, Cluster2 enforces the following important property which will be needed for proving the diameter approximation. With reference to a specific execution of Cluster2, define the *light distance* between two nodes u and v as the weight of the minimum-weight path from u and v consisting only of edges of weight at most $2R_G^{CL}(\tau)$. (Note that the light distance is not necessarily defined for every pair of nodes.)

Due to the weight rescaling performed by Contract2 at the end of each iteration, given a center c selected at a certain Iteration i of the **for** loop, and a node v at light distance d from c, the cluster centered at c cannot grow to reach v in less than

$\lceil d/2R_G^{CL}(\tau)\rceil$ iterations and that in those many iterations v will be reached by some cluster (possibly the one centered at c). Consequently, no center selected at a later iteration at that same distance from v as c would be able to reach v.

We are now ready to present the main result of this section, which shows how we can employ CLUSTER2 to determine a good approximation to the graph diameter. Suppose we run CLUSTER2 on a graph $G = (V, E, w)$ to obtain a clustering C of radius $R_G^{CL2}(\tau)$. For each $u \in V$, let c_u be the center of the cluster assigned to u, and let d_u be distance between u and c_u returned by CLUSTER2. As in [Mey08], we define the weighted auxiliary graph associated to C as the graph G_C where nodes correspond to clusters and, for each edge (u, v) of G with $c_u \neq c_v$, there is an edge in G_C between the clusters of u and v with weight $w(u, v) + d_u + d_v$. In case of multiple edges between two clusters, we retain only the one yielding minimum weight. Let $\Phi(G)$ (resp., $\Phi(G_C)$) be the weighted diameter of G (resp., G_C). We approximate $\Phi(G)$ through the value

$$\Phi_{approx}(G) = \Phi(G_C) + 2R_G^{CL2}(\tau). \tag{3.1}$$

It is easy to see that our estimate is conservative, that is, $\Phi_{approx}(G) \geqslant \Phi(G)$. We have:

Theorem 4. *With high probability,*

$$\Phi_{approx}(G) = O\left(\Phi(G)\log^3 n\right).$$

Proof. Since $R_G^{CL2}(\tau) = O\left(R_G^{opt}(\tau)\log^2 n\right)$ (by Lemma 2), and $R_G^{opt}(\tau) = O\left(\Phi(G)\right)$, we have that $R_G^{CL2}(\tau) = O\left(\Phi(G)\log^2 n\right)$. Therefore, to prove the theorem it remains to show that $\Phi(G_C) = O\left(\Phi(G)\log^3 n\right)$. Let us fix an arbitrary pair of clusters C_1, C_2 and an arbitrary minimum-weight path π between their centers in G, and let w_π be the weight of π. Let π_C be the path of clusters in G_C traversed by π. We now show that with high probability the weight of π_C in G_C is $O\left(\Phi(G)\log^3 n\right)$, by distinguishing two cases.

Case 1. Suppose that $2R_G^{CL}(\tau) > \Phi(G)$ (note that this can happen since the clustering yielding the radius $R_G^{CL}(\tau)$ determined at the beginning of CLUSTER2 is built out of paths using only light edges of weight $O\left(R_G^{opt}(\tau)\right)$). In this case, it is easy to see that the first batch of centers ever selected in an iteration of the for loop of CLUSTER2 will cover the entire graph, and these centers are $O(\log n)$ with high probability. Therefore, π_C contains $O(\log n)$ clusters and its weight is $O\left(w_\pi + R_G^{CL2}(\tau)\log n\right) = O\left(\Phi(G)\log^3 n\right)$.

Case 2. Suppose now that $2R_G^{CL}(\tau) \leqslant \Phi(G)$. We show that, with high probability, at most $O\left(\lceil w_\pi/R_G^{CL}(\tau)\rceil \log^2 n\right)$ clusters intersect π (i.e.,contain nodes of π). It can be seen that π can be divided into $O\left(\lceil w_\pi/R_G^{CL}(\tau)\rceil\right)$ subpaths, where each subpath is either an edge of weight $> R_G^{CL}(\tau)$ or a segment of weight $\leqslant R_G^{CL}(\tau)$. It is then sufficient to show that the nodes of each of the latter segments belong to $O\left(\log^2 n\right)$ clusters. Consider one such segment S. Clearly, all clusters containing nodes of S must have their centers at light distance at most $R_G^{CL2}(\tau)$ from S (i.e., light distance at most $R_G^{CL2}(\tau)$ from the closest node of S). Recall that $R_G^{CL2}(\tau) \leqslant 2R_G^{CL}(\tau)\log n$. For $1 \leqslant j \leqslant 2\log n + 1$, let $C(S, j)$ be the set of nodes whose light distance from S

is between $(j-1)R_G^{CL}(\tau)$ and $jR_G^{CL}(\tau)-1$, and observe that any cluster intersecting S must be centered at a node belonging to one of the $C(S,j)$'s. We claim that, with high probability, for any j, there are $O(\log n)$ clusters centered at nodes of $C(S,j)$ which may intersect S. Fix an index j, with $1 \leqslant j \leqslant 2\log n + 1$, and let i_j be the first iteration of the for loop of CLUSTER2 in which some center is selected from $C(S,j)$. By the property of CLUSTER2 discussed after Lemma 2, $\lceil((j+1)R_G^{CL}(\tau)-1)/(2R_G^{CL}(\tau))\rceil$ iterations are sufficient for any of these centers to cover the entire segment. On the other hand, any center from $C(S,j)$ needs at least $\lceil(j-1)R_G^{CL}(\tau)/(2R_G^{CL}(\tau))\rceil$ iterations to touch the segment. Hence, we have that no center selected from $C(S,j)$ at Iteration i_j+2 or higher is able to reach S. It is easy to see that, due to the smooth growth of the center selection probabilities, the number of centers selected from $C(S,j)$ in Iterations i_j and i_j+1 is $O(\log n)$, with high probability. This implies that the nodes of segment S will belong to $O\left(\log^2 n\right)$ clusters, with high probability. By applying the union bound over all segments of π, we have that $O\left(\lceil w_\pi/R_G^{CL}(\tau)\rceil \log^2 n\right)$ clusters intersect π, with high probability. Therefore, with high probability the two clusters C_1, C_2 are connected in G_C by a path of weight at most π_C, which is

$$O\left(w_\pi + R_G^{CL2}(\tau)\left\lceil\frac{w_\pi}{R_G^{CL}(\tau)}\right\rceil \log^2 n\right) = O\left(\Phi(G) + R_G^{CL2}(\tau)\frac{\Phi(G)}{R_G^{CL}(\tau)}\log^2 n\right)$$
$$= O\left(\Phi(G)\log^3 n\right)$$

The theorem follows by applying the union bound over all pairs of clusters. □

3.4 Implementation in the MapReduce model

We now discuss the implementation of the above diameter approximation algorithm in the MapReduce model described in Section 2.2.1, using overall linear space. We will show that, for a relevant class of graphs, we can make its round complexity asymptotically smaller than the one required to obtain a 2-approximation through the state-of-the-art SSSP algorithm by [MS03].

The basic idea is to run the clustering algorithm CLUSTER2 described in the previous section with a cluster granularity τ such that the auxiliary graph built from the clustering fits into the memory of a single reducer. We first observe that, regardless of the number of active clusters and for any Δ, we can implement a Δ-growing step on the $MR(M_L, M_A)$ model through a constant number of simple prefix and sorting operations. Recall that, by Theorem 1, these operations require $O\left(\log_{M_L} n\right)$ rounds. Therefore, by combining Theorem 1 and Lemma 2, we easily derive the following result.

Lemma 3. *Let G be a connected graph with n nodes and m edges. On the $MR(M_L, M_A)$ model, with $M_A = \Theta(m)$, algorithm CLUSTER2(G,τ) can be implemented in*

$$O\left(\ell_{R_G^{opt}(\tau)\log n}\log n \log_{M_L} n\right)$$

rounds. In particular, if $M_L = \Omega(n^\varepsilon)$, for some constant $\varepsilon > 0$, the number of rounds becomes $O\left(\ell_{R_G^{opt}(\tau)\log n}\log n\right)$.

The theorem below states the performance of the diameter approximation algorithm on the $MR(M_L, M_A)$ model when M_A is linear in the graph size and sufficient local memory is available. In particular, the theorem shows an interesting tradeoff between the number of rounds and the size M_L of the local memory.

Theorem 5. *Let G be a connected weighted graph with n nodes, m edges, and weighted diameter $\Phi(G)$. Also, let $\varepsilon' < \varepsilon \in (0,1)$ be two arbitrary constants, and let $\tau = \lceil n^{\varepsilon'}/\log^4 n \rceil$. With high probability, an estimate $\Phi_{approx}(G)$ to the diameter $\Phi(G)$ of G, such that $\Phi(G) \leqslant \Phi_{approx}(G) \leqslant O(\Phi(G)\log^3 n)$, can be computed in*

$$O\left(\ell_{R_G^{opt}(\tau)\log n}\log n\right)$$

rounds on the MR(M_L, M_A) model with $M_L = \Theta(n^{\varepsilon})$ and $M_A = \Theta(m)$.

Proof. By setting $\tau = \lceil n^{\varepsilon'}/\log^4 n \rceil$, we have that Cluster2 returns $O(n^{\varepsilon'})$ clusters with high probability. In case the number of clusters is larger, we repeat the execution of Cluster2. Let $G_C = (V_C, E_C)$ be the auxiliary graph associated with the returned clustering. By the choice of τ, we have that $|V_C| \leqslant M_L$. Furthermore, if $|E_C| \leqslant M_L$ we can compute the diameter of G_C in a single reducer in one round. Otherwise, by employing the sparsification technique presented in [BS07] we transform G_C into a new graph $G'_C = (V, E'_C)$ with $|E'_C| \leqslant M_L$, whose diameter is a factor at most $O(\varepsilon'/(\varepsilon - \varepsilon')) = O(1)$ larger than the diameter of G_C. This sparsification technique requires a constant number of cluster growing steps similar in spirit to those described above, which can be realized through a constant number of prefix and sorting operations. By Theorem 1 we can implement this transformation in $O(1)$ rounds in the $MR(M_L, M_A)$ model. By combining the discussion above with Lemma 3, we have that the execution of Cluster2 followed by the computation of the diameter of the auxiliary graph takes $O\left(\ell_{R_G^{opt}(\tau)\log n}\log n\right)$. □

Note that the round complexity depends on the characteristics of the graph and is nonincreasing in the number of clusters as controlled by τ, which is in turn a function of M_L. To take into account the topological properties of the input graph, and remove the dependency on $\ell_{R_G^{opt}(\tau)\log n}$, we express the round complexity in terms of the *doubling dimension* of the graph, which we introduced in Definition 2: for a given graph G, its doubling dimension is the smallest D such that any ball of radius r can be covered by at most 2^D balls of radius $r/2$. Recall that $\Phi(G)$ and $\Psi(G)$ denote the weighted and unweighted diameter of G.

Before stating the performance of our algorithm parameterized with the doubling dimension of the graph, we need the following technical lemma.

Lemma 4. *Given a graph G with maximum degree d, if we remove edges with probability $p > 1 - 1/d$ (i.e. we leave an edge in the graph with probability $< 1/d$) then whp the graph becomes disconnected and each connected component has size $O(\log n)$.*

Proof. Let $G' = (V, E')$ be a graph obtained from $G = (V, E)$ by removing each edge in E with probability $1 - p > 1 - 1/d$. Equivalently, each edge in E is included in E' with probability $p < 1/d$, independent of other edges. If G' has a connected component of size k then it must have a tree of size k. We prove the claim by

showing that for $c > 6\left(\frac{1-p(d-1)}{p(d-1)}\right)^2$, the probability that a given vertex v is part of a tree of size $k = c\log n$ is bounded by $1/n^2$.

For a fixed vertex $v \in V$, let $Y_0 = \{v\}$ and let Y_i be the set of vertices connected to Y_{i-1} but not to Y_j, $j < i-1$, i.e.

$$Y_i = \{w \mid \exists (w,u) \in E',\ w \notin \cup_{i=0}^{i-1} Y_i,\ u \in Y_{i-1}\},$$

Consider a Galton–Watson branching process [Ken75] $\{Z_i,\ i \geqslant 0\}$, with $Z_0 = 1$, and $Z_i = \sum_{j=1}^{Z_{i-1}} X_{j,i}$, where $X_{j,i}$ are independent, identically distributed random variables with a Binomial distribution $B(d-1,p)$. Clearly for any $i \geqslant 0$, the distribution of Y_i is stochastically upper bounded by the distribution of Z_i.

Since a tree of k vertices has $k-1$ edges, and the number of decendents of different nodes are independent,

$$Pr(\sum_{i\geqslant 0} Z_i \geqslant k) \leqslant Pr(\sum_{i=1}^{k} X_i \geqslant k-1)$$

where X_i are independent random variables distributed $B(d-1,p)$ [Dwa69]. Now, we have $E[\sum_{i=1}^{k} X_i] = k(d-1)p$, therefore, by applying a Chernoff bound, the probability that v is part of a tree of size $c\log n$ is bounded by

$$Pr(\sum_{i\geqslant 0} Y_i \geqslant k) \leqslant Pr(\sum_{i\geqslant 0} Z_i \geqslant k) \leqslant e^{-c\log n/3} \leqslant n^{-2}$$

By union bound over the n nodes, the probability that the graph has a connected component of size greater than $c\log n$ is bounded by $1/n$. □

An alternative way of interpreting the above lemma is the following.

Observation 1. *Given a graph of maximum degree d with edge weights uniformly distributed in $[w_{min}, w_{max}]$, if we remove the edges whose weight is $\geqslant (w_{min}+w_{max})/d$, then with high probability the graph becomes disconnected and the size of each connected component is $O(\log n)$. That is, in any simple path the length of any segment of consecutive edges with weight $< (w_{min}+w_{max})/d$ is $O(\log n)$, whp.*

Thanks to the above interpretation of Lemma 4, we can state the following corollary of Theorem 5.

Corollary 1. *Let G be a connected graph with n nodes, m edges, maximum degree $d \in O(1)$, doubling dimension D, and positive edge weights chosen uniformly at random from $[w_{min}, w_{max}]$, with $w_{max}/w_{min} \in O(n^h)$ for some $h > 0$. Also, let $0 < \varepsilon' < \varepsilon < 1$ be two arbitrary constants. With high probability, an estimate $\Phi_{approx}(G)$ to the diameter $\Phi(G)$ of G, such that $\Phi(G) \leqslant \Phi_{approx}(G) \leqslant O(\Phi(G)\log^3 n)$, can be computed in*

$$O\left(\left\lceil \frac{\Psi(G)\log^{4/D} n}{n^{\varepsilon'/D}} \right\rceil \log^3 n\right)$$

rounds on the MR(M_L, M_A) model with $M_L = O(n^\varepsilon)$ and $M_A = \Theta(m)$.

Proof. As in Theorem 5, we set $\tau = \lceil n^{\varepsilon'}/\log^4 n \rceil$ so as to have CLUSTER2 returning $O(n^{\varepsilon'})$ clusters with high probability. By iterating the definition of doubling dimension starting from a single ball of unweighted radius $\Psi(G)$ containing the whole graph, we can decompose the graph into τ disjoint clusters of unweighted radius $\psi = O\left(\lceil \Psi(G)/\tau^{1/D} \rceil\right)$. Since w_{max} be the maximum edge weight, we have that $\psi \cdot w_{max}$ upper bounds $R_G^{opt}(\tau)$. We know that our algorithm computes the diameter approximation in $O\left(\ell_{R_G^{opt}(\tau)\log n} \log n\right)$ rounds. We will now give an upper bound on $\ell_{R_G^{opt}(\tau)\log n}$. By Lemma 4 and Observation 1, we have that by removing all edges of weight $\geqslant (w_{min} + w_{max})/d$ with high probability the graph becomes disconnected and each connected component has $O(\log n)$ nodes. As a consequence, with high probability any simple path in G will traverse an edge of weight $\geqslant (w_{min} + w_{max})/d = \Omega(w_{max})$ every $O(\log n)$ nodes. This implies that a path of weight at most $R_G^{opt}(\tau)\log n = O\left(\left\lceil \frac{\Psi(G)}{\tau^{1/D}} \right\rceil w_{max} \log n\right)$ has $\ell_{R_G^{opt}(\tau)\log n} = O\left(\left\lceil \frac{\Psi(G)}{\tau^{1/D}} \right\rceil \log^2 n\right)$ edges. The statement follows by the choice of τ. □

The above corollary ensures that, for graphs of constant doubling dimension, we can make the number of rounds polynomially smaller than the unweighted diameter $\Psi(G)$. This makes our algorithm particularly suitable for inputs that are otherwise challenging in MapReduce, like high-diameter, mesh-like sparse topologies (a mesh has doubling dimension 2). On these inputs, performing a number of rounds sublinear in the unweighted diameter is crucial to obtain good performance. Conversely, algorithms for the SSSP problem perform a number of rounds linear in the diameter. Consider for instance Δ-stepping that, being a state of the art parallel SSSP algorithm, is our most natural competitor. Given a graph G with random uniform weights, the analysis in [MS03] implies that under the linear-space constraint a natural MR-implementation of Δ-stepping requires $\Omega(\Psi(G))$ rounds. By Corollary 1, if G has bounded doubling dimension the round complexity of our algorithm can be made smaller by a sublinear yet polynomial factor, which is a function of the available local space M_L. We will verify experimentally this performance difference in the Section 3.7.

3.5 Improved performance for unweighted graphs

We can show that running the MapReduce implementation described in the previous section on unweighted graphs is faster than in the general case. In fact, in the unweighted case, the Δ-growing step is very efficient, since once a node is reached for the first time, it is always with the minimum distance, so it will not be further updated. In fact, in the unweighted case the Δ-growing step is conceptually equivalent to a BFS-like expansion. This results in an improvement in the running time by a logarithmic factor, as shown in the following corollary to Theorem 5.

Corollary 2. *Let G be a connected unweighted graph with n nodes, m edges, and doubling dimension D. Also, let $0 < \varepsilon' < \varepsilon < 1$ be two arbitrary constants. With high probability, an estimate $\Phi_{approx}(G)$ to the diameter $\Phi(G)$ of G, such that $\Phi(G) \leqslant \Phi_{approx}(G) \leqslant$*

$O(\Phi(G)\log^3 n)$, *can be computed in*

$$O\left(\left\lceil \frac{\Phi(G)\log^{4/D} n}{n^{\varepsilon'/D}} \right\rceil \log^2 n\right)$$

rounds on the MR(M_L, M_A) *model with* $M_L = O(n^\varepsilon)$ *and* $M_A = \Theta(m)$.

Proof. By reasoning as in the proof of Corollary 1 we can show that G can be decomposed into τ disjoint clusters of radius $\varphi = O\left(\lceil \Phi(G)/\tau^{1/D} \rceil\right)$. Therefore, φ upper bounds the optimal radius $R_G^{opt}(\tau)$.

Recall that, for a given x, ℓ_x is by definition one plus the maximum number of edges in any shortest path connecting two nodes at distance $\leqslant x$. Therefore, in an unweighted graph, which is equivalent to a weighted graph with unit edge weights, we have by definition that $\ell_{R_G^{opt}(\tau)\log n} = R_G^{opt}(\tau)\log n + 1$. Recall also that our algorithm computes the diameter approximation in $O\left(\ell_{R_G^{opt}(\tau)\log n}\log n\right)$ rounds. By combining these observations with the upper bound to $R_G^{opt}(\tau)$ discussed above, we have that the algorithm runs in $O\left(\lceil \Phi(G)/\tau^{1/D} \rceil \log^2 n\right)$ MapReduce rounds. The statement follows by choosing $\tau = \lceil n^{\varepsilon'}/\log^4 n \rceil$, as argued in the proof of Corollary 1. □

Observe that Corollary 1 for weighted graphs requires the input graph to have edge weights distributed uniformly at random, in order to ensure the result with high probability. Conversely, the above Theorem 2 holds with high probability based only on the choices of the algorithm.

In the case of unweighted graphs, the most natural competitor is the simple BFS search, instead of Δ-stepping. The same considerations of the previous section hold, since a BFS search requires $\Theta(\Phi(G))$ rounds, similarly to Δ-stepping. Another family of competitors is represented by neighbourhood function-based algorithms [PGF02; BRV11; Kan+11], which we reviewed in Section 3.1.3. These algorithms, like the BFS, require $\Theta(\Phi(G))$ rounds, and are therefore outperformed by our approach on graphs with constant doubling dimension.

3.6 Approximation to k-center

In the previous sections, we introduced diameter approximation algorithms which used, as a subroutine, the construction of clusters of small radius around center nodes. It is natural to envision the application of our clustering strategies to the solution of the k-center problem on graphs, which was introduced in Section 2.4. In particular, the following theorem shows that Algorithm Cluster can be used to solve the k-center problem for unweighted graphs.

Theorem 6. *Let* G *be a connected graph with* n *nodes,* m *edges and doubling dimension* D. *For* $k = \Omega(\log^2 n)$, *algorithm* Cluster *can be employed to compute, with high probability, an* $O(\log^3 n)$*-approximation to the* k*-center problem on* G *in*

$$O\left(\left\lceil \frac{\Phi(G)\log^{2/D}}{k^{1/D}} \right\rceil \log n\right)$$

rounds, on the MR(M_L, M_A) *model with* $M_A = \Theta(m)$ *and* $M_L = O(n^\varepsilon)$*, for some constant* $\varepsilon \in (0, 1]$*.*

Proof. Fix $\tau = \Theta\left(k/\log^2 n\right)$ so that Cluster(τ) returns at most k clusters with high probability, and let M be the set of centers of the returned clusters. Without loss of generality, we assume that M contains exactly k nodes. In case $|M| < k$, we can add $k - |M|$ arbitrary nodes to M, which will not increase the value of the objective function. Let $R_{CL(\tau)}$ be the maximum radius of the clusters returned by our algorithm. As proved in Theorem 3, we have that, with high probability, $R_{CL}(\tau) = O\left(R_G(\tau) \log n\right)$. We now argue that $R_G^{opt}(\tau) = O\left(R_G^{opt}(k) \log^2 n\right)$. Consider the optimal solution to the k-center problem on the graph, and the associated clustering of radius $R_G(k)$. Furthermore, consider the auxiliary graph built from this clustering as described in Section 3.3, and let T be a spanning tree of such a graph. It is easy to see that T can be decomposed into τ subtrees of height $O\left(\log^2 n\right)$ each. Merge the clusters associated with the nodes of each such subtree and pick any node as center of the merged cluster. By construction, we have that given an arbitrary node u and its closest newly picked center c, the path from u to c traverses at most $\log^2 n$ original clusters, each contributing a factor $O\left(R_G^{opt}(k)\right)$ to the path length. Since the graph has unit edge weights, the edges between clusters account for an additive $O\left(\log^2 n\right)$ factor, overall. We have then that every node in the graph is at distance $D = O\left(R_G(k) \log^2 n + \log^2 n\right) = O\left(R_G(k) \log^2 n\right)$ from one of the picked nodes. Since $D \geqslant R_G^{opt}(\tau)$, we conclude that $R_G^{opt}(\tau) = O\left(R_G^{opt}(k) \log^2 n\right)$, and bound on the approximation factor follows.

As for the number of rounds in the MR(M_L, M_A) model, by iterating the definition of doubling dimension, starting from a single ball of radius $\Phi(G)$ and containing the whole graph, we can cover the graph with τ disjoint clusters of radius $\varphi = O\left(\lceil \Phi(G)/\tau^{1/D} \rceil\right)$. Clearly, φ upper bounds the optimal radius of the clustering $R_G^{opt}(\tau)$. By Lemma 3, Cluster performs $O\left(\ell_{R_G^{opt}(\tau)} \log n\right)$ Δ-growing step, which by the discussion above is $O\left(\ell_\varphi \log n\right)$. Each of these Δ-growing steps can be implemented with a constant number of sorting and prefix operations. Thus, by Theorem 1, we have that a Δ-growing step can be implemented in $O\left(\log_{M_L} n\right)$ rounds in the MR(M_L, M_A) model. Since by hypothesis the local memory is $M_L = O\left(n^\varepsilon\right)$ for constant $\varepsilon \in (0, 1]$, we have that each Δ-growing step can be implemented in a constant number of rounds. Therefore, Cluster computes a clustering in $O\left(\ell_\varphi \log n\right)$ rounds. By reasoning as in the proof of Theorem 2 we have that $\ell_\varphi = O(\varphi)$, therefore the number of rounds is $O\left(\lceil \Phi(G)/\tau^{1/D} \rceil \log n\right)$. The round complexity follows by the choice of τ. □

In Section 2.4.2 we reviewed the MapReduce k-center algorithm by Malkomes et al. [Mal+15], which runs in two rounds, and provides a 4-approximation for the k-center problem. Therefore it seems to have a better performance both from the perspective of round complexity and approximation ratio. However, it requires either a distance oracle or the $\Theta\left(n^2\right)$ distances to be given in input. Hence, it seems unlikely that the algorithm from [Mal+15] can be applied to the k-center problem on graphs in 2 rounds using in linear space. To the best of our knowledge, ours is the first MapReduce algorithm that solves the k-center problem on unweighted

graphs with a provable approximation guarantee, and with a number of rounds sublinear in the diameter for graphs with bounded doubling dimension.

Our k-center approximation algorithm can also be applied to disconnected graphs. Consider a graph G with $h > 1$ connected components. Observe that for $k \geqslant h$, the k-center problem still admits a solution with non-infinite radius. Given $k \geqslant h$, we can still get a $O\left(\log^3 n\right)$-approximation to k-center as follows. If $k = \Omega\left(h \log^2 n\right)$, then the graph has enough connected components to run $\textsc{Cluster}(G, \tau)$ with $\tau = \Theta\left(k/\log^2 n\right)$ as we described before. If instead $h \leqslant k = o(h \log^2 n)$, setting $\tau = \Theta\left(k/\log^2 n\right)$ would not do, because $\textsc{Cluster}$ will try to use less cluster centers than the number of connected components. In this case we run $\textsc{Cluster}(G, h)$, obtaining a clustering with $O\left(h \log^2 n\right)$ components, by Theorem 3. From this clustering, we can obtain a k-clustering by applying the same merging technique introduced in the proof of Theorem 6 as follows. We consider the auxiliary graph built from the clustering as described in Section 3.3, and we let T be a spanning tree of such a graph. We then decompose this spanning tree into k subtrees, merging all the clusters associated to the nodes of a given subtree. It is easy to see that the approximation ratio is still $O\left(\log^3 n\right)$.

Theorem 6 holds only for unweighted graphs. As for weighted graphs, the derivation of a theoretical bound on the approximation factor that can be obtained by using algorithm $\textsc{Cluster}$ remains an open problem. In particular, when dealing with weighted graphs, the spanning tree-based construction adopted in the proof cannot be used: while we can still decompose the spanning tree in such a way that any path traverses at most $\log^2 n$ clusters, the edges between different clusters may have an arbitrary weight. The development of a proof for the case of weighted graphs is the object of ongoing work.

3.7 Experimental Analysis

In this section we present an experimental analysis of our algorithm. The goal is to verify the quality of the approximation provided by our algorithm over a wide range of topologies, and to compare with the performance of other algorithms presented in the literature. Furthermore, we analyze the scalability of our implementations. We run experiments on both weighted and unweighted graphs in order to compare with existing algorithms for both cases. For weighted graphs, in subsection 3.7.1, we compare with our most natural competitor, the Δ-stepping algorithm [MS03]. As for unweighted graphs (subsection 3.7.2), we compare our diameter approximation algorithm with the BFS algorithm and with HADI [Kan+11]. Finally, we compare our $\textsc{Cluster}$ algorithm with the clustering algorithm introduced in [MPX13], which we reviewed in Section 2.4.2, from the perspective of the quality of clustering.

We implemented our algorithms, whose code is openly accessible [Cec15] within the Spark framework, which we introduced in Section 2.2.2. We also contributed our implementation to the repository of publicly available Spark libraries[1], to allow an easy integration of our algorithm in other software [Cec16b]. All the experiments have been run on the 16-machines clusters described in Section 2.2.2. We implemented a simplified version of our diameter approximation algorithm,

[1] `https://spark-packages.org/`

dubbed CL-DIAM, where, for efficiency, we used CLUSTER for computing the graph clustering, rather than CLUSTER2. In fact, CLUSTER2 first runs CLUSTER to obtain an estimate of the radius, and then computes a second clustering which is instrumental to provide a theoretical bound to the approximation factor, but which does not seem to provide a significant improvement to the quality of the approximation in practice.

As a second optimization, we ran CLUSTER using an initial value of Δ larger than the minimum edge weight, as was specified in the pseudocode. We observe that by increasing the initial value of Δ, the round complexity improves since less doublings are required before reaching the final value. On the other hand, setting the initial value of Δ too large may yield a larger cluster radius, possibly incurring a worse diameter approximation. To explore this phenomenon, we experimented on a 2048×2048 mesh with random edge weights, such that an edge has weight 1 with probability 0.1 and 10^{-6} otherwise. With high probability, such a graph can be completely covered using clusters that do not contain edges of weight 1: including one of those edges in a cluster would make its radius far bigger than it needs to be. We ran our algorithm with two configurations. The first configuration started with $\Delta = 10^{-6}$ (i.e., the minimum edge weight) so as to let the algorithm tune itself to the final value $\Delta_{end} = 6.4 \cdot 10^{-5}$; the second configuration started with an inital Δ equal to the graph diameter (≈ 2.004) so that no doubling of Δ was needed. The diameter approximation obtained by the second configuration was about 2.5 times larger than the actual diameter, whereas the first configuration obtained an approximation ratio of 1.0001. A set of experiments (omitted here for the sake of brevity) showed that a good initial guess for Δ is the average edge weight, which reduces the round complexity without affecting the approximation quality significantly. Therefore, all our experiments have been run with this initial guess of Δ.

In all experiments the parameter τ was set so that the final diameter computation in the auxiliary graph would not dominate the running time. Since the auxiliary graphs turned out in all cases to be sufficiently sparse, the use of sparsification techniques mentioned in the proof of Theorem 5 was not needed. Furthermore, whenever we refer to the "true diameter" of the input graph, we refer to the maximum distance found with a very large number of sequential SSSP runs. This lower bound to the actual diameter is also used to compute the approximation ratios, which are therefore overestimated, since the approximation algorithms we deal with provide upper bounds to the diameter. Finally, all values reported as results of the experiments are averages over at least 5 runs.

3.7.1 Experiments on weighted graphs

To explore the performance of our diameter approximation algorithm, we used several weighted graphs, whose properties are summarized in Table 3.1. For some graphs we added edge weights distributed uniformly at random in $(0, 1]$. Note that edge weights generated in this way comply with the assumption of polynomiality we introduced at the beginning of Section 3.2. In fact, since we use double precision floating point numbers (standard IEEE 754) the smallest positive real number that can be represented is a constant. Therefore, the ratio between the maximum weight

and the smallest one is actually constant in n. The graphs can be classified as follows:

road networks: `roads-USA` and `roads-CAL` are respectively the road networks of USA and California with distances in meters, as taken from the collection of datasets of the 9th DIMACS challenge[2]. Both graphs are sparse and feature a large unweighted diameter.

social networks: `livejournal` and `twitter` are two social network datasets taken respectively from the SNAP collection[3] and the WebGraph collection[4]. Both graphs are denser than the aforementioned road networks, and their unweighted diameter is small. Since these graphs were originally unweighted, we assigned random uniform edge weights in $(0, 1]$, according to the approach commonly adopted in the literature. Note that the `twitter` graph is particularly massive, with almost 1.5 billion edges.

synthetic graphs: we also use three classes of artificially generated graphs whose size can be made arbitrarily large through a parameter S and whose topological properties reflect those of the real networks in the first two classes:

- `mesh(S)`: a $S \times S$ mesh, with edge weights uniformly distributed in $(0, 1]$. These are graphs of known doubling dimension $D = 2$, for which the results of Corollary 1 hold. Moreover, this is a topology with properties similar to those of road networks.
- `R-MAT(S)`: these are graphs mimicking the structure of social networks, with a power-law degree distribution and small diameter. They are included in the GRAPH500 benchmark set [Mur+10]. Edge weights have a uniform distribution between in $(0, 1]$.
- `roads(S)`: graphs are obtained as the cartesian product of a linear array of $S > 1$ nodes and unit edge weights with `roads-USA`. Basically, graphs of this family are created by layering S copies of `roads-USA`, and by connecting the corresponding nodes of adjacent layers with unit edge weights.

Comparison with the SSSP-based approximation algorithm

Recall that an SSSP algorithm can be used to yield an upper bound to the diameter, at most a factor-2 away from the actual value, by returning twice the weight of the heaviest shortest path. Thus, we compared our algorithm CL-DIAM with a Spark implementation of the Δ-stepping SSSP algorithm (starting from a random node), which, as we saw in the preliminaries, is the state of the art for parallel SSSP and is in fact our only practical competitor on weighted graphs. In Δ-stepping, parameter Δ can be set to control the tradeoff between parallel time (i.e., rounds in the MapReduce context) and total work. For each graph, we tested Δ-stepping

[2]http://www.dis.uniroma1.it/challenge9/
[3]http://snap.stanford.edu/data
[4]http://law.di.unimi.it/webdata/twitter-2010/

Graph	n	m	$\Phi(G)$
`roads-USA`	23,947,347	29,166,673	55,859,820
`roads-CAL`	1,890,815	2,328,872	16,485,258
`livejournal` $\star$	3,997,962	32, 681, 189	9.41
`twitter` $\star$	41,652,230	1,468,365,182	9.07
`mesh(S)` $\star$	S^2	$2S(S-1)$	†
`R-MAT(S)` $\star$	2^S	$16 \cdot 2^S$	†
`roads(S)`	$\approx S \cdot 2.3 \cdot 10^7$	$\approx S \cdot 5.3 \cdot 10^7$	†

$\star$ edge weights randomly distributed $\in (0, 1]$.
† the diameter depends on the size of the graph, controlled by $S > 1$.

Table 3.1: Weighted benchmark graphs. $\Phi(G)$ is the weighted diameter.

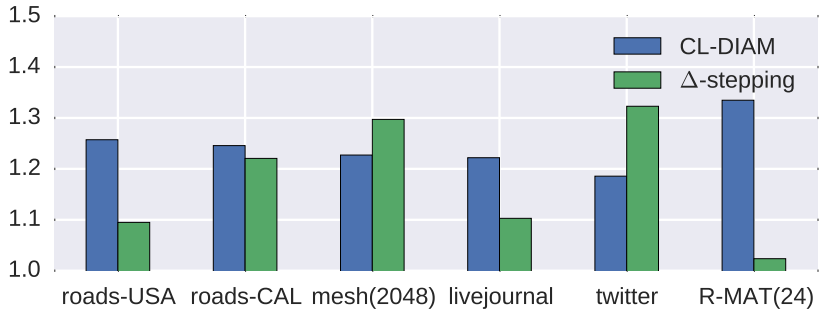

Figure 3.1: Approximation ratio of CL-DIAM and Δ-stepping.

with several values of Δ, selecting the value yielding the best running time. Since in MapReduce-like environments the number of rounds has a significant impact on the running time, for all graphs the best value of Δ turned out to be always the one minimizing the number of rounds. As for our algorithm, we set τ so that CL-DIAM yields an auxiliary graph with $\leqslant 100,000$ nodes. In Section 3.7.2, we will study the influence of the clustering granularity on the approximation factor, finding that it has no impact, in accordance with the theory.

The results of the comparison between CL-DIAM and Δ-stepping are summarized in Table 3.2 and graphically represented in Figures 3.1, 3.2, and 3.3. In the table we report, for each graph, the approximation factor and the running time. Along with these, we also report two additional measures, namely, the number of rounds and the work (defined as the sum of node updates and messages generated), that allow to compare the two algorithms in a more platform-independent way. It has to be remarked that the approximation factor featured by CL-DIAM on all benchmark graphs, which is always less that 1.4, is much better that the theoretical $O\left(\log^3 n\right)$ bound. Also, our algorithm is from about one to two orders of magnitude faster than Δ-stepping, while featuring comparable approximation ratios (Figure 3.1). As expected, the higher performance of CL-DIAM is consistent with the fact that it requires far less rounds than Δ-stepping, as shown in Figure 3.2.

For what concerns the work, the better performance of CL-DIAM, as shown in Figure 3.3, is mainly due to the smaller number relaxations with respect to Δ-

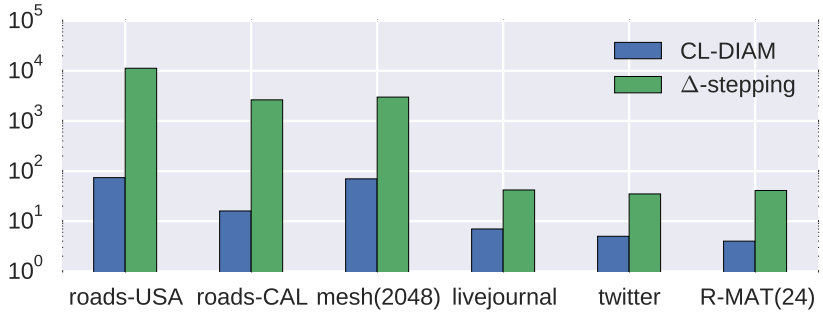

Figure 3.2: Number of rounds required by CL-DIAM and Δ-stepping. The scale is logarithmic.

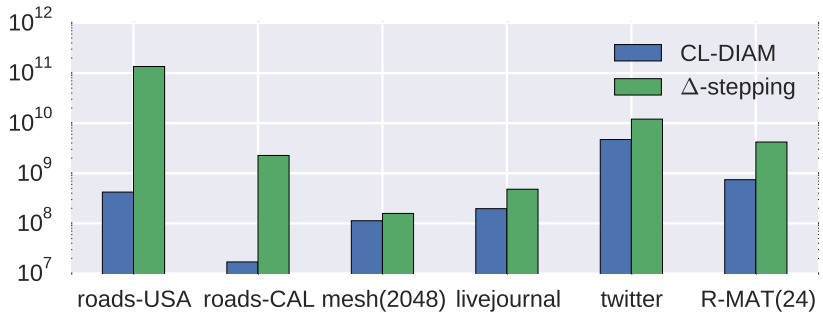

Figure 3.3: Work performed by CL-DIAM and Δ-stepping. The scale is logarithmic.

	approximation		time		rounds		work	
graph	CL	ΔS	CL	ΔS	CL	ΔS	CL	ΔS
roads-USA	1.26	1.09	158	14,982	74	11,268	$4.22 \cdot 10^8$	$1.35 \cdot 10^{11}$
roads-CAL	1.25	1.22	13	917	16	2,639	$1.70 \cdot 10^7$	$2.27 \cdot 10^9$
mesh(2048)	1.23	1.30	46	1,239	70	2,997	$1.13 \cdot 10^8$	$1.58 \cdot 10^8$
livejournal	1.22	1.10	19	74	7	42	$1.97 \cdot 10^8$	$4.81 \cdot 10^8$
twitter	1.19	1.32	236	601	5	35	$4.71 \cdot 10^9$	$1.20 \cdot 10^{10}$
R-MAT(24)	1.33	1.02	144	1,493	4	41	$7.45 \cdot 10^8$	$4.20 \cdot 10^9$

Table 3.2: CL-DIAM vs Δ-stepping (resp. CL and ΔS in the table, for brevity). For each benchmark, the table shows the running time (in seconds), the approximation ratio, the number of rounds, and the work of the two algorithms.

stepping. Indeed, CL-DIAM explores paths only up to a limited depth, whereas Δ-stepping needs to run until all the nodes are labeled with the optimal distance from the source. In fact, Δ-stepping could limit the amount of relaxations by using a smaller Δ, but in doing so it would incur an increase of the number of rounds, hence exhibiting worse performance. The gap between Δ-stepping and CL-DIAM in terms of both number of rounds and work suggests that our algorithm is likely to remain competitive on other distributed-memory platforms employing programming frameworks alternative to MapReduce.

We remark that the experiments reported in Table 3.2 involve graphs of moderate size for which, not surprisingly, much better running times can be obtained on a single machine equipped with sufficient main memory. In fact, the purpose of those experiments was not to attain best absolute performance on the individual graphs but, rather, to compare the relative performance of CL-DIAM with the one of Δ-stepping. Since it is conceivable to expect that the relative performance of two algorithms does not change as the graphs size grows, performing this comparison on much larger graphs would have only encumbered the experimental work without changing the overall outcome. Nevertheless, we performed further experiments, reported in the next subsection, to provide evidence that our algorithm scales well with respect to the number of machines and input size.

Scalability.

To check the scalability of CL-DIAM with respect to the number of machines, we ran it using 2, 4, 8, and 16 machines on R-MAT(26) and roads(3) which have approximately the same number of nodes but topologies of different nature. The results are reported in Figure 3.4 which shows that, for both graphs, the algorithm exhibits excellent scalability.

Graph	time (seconds)
R-MAT(29)	6218
roads(32)	14054

Table 3.3: Experiments on big graphs

Finally, we ran CL-DIAM on R-MAT(29) and roads(32), which are much larger graphs than those employed in the experiments reported in Table 3.2, and for which the running time of Δ-stepping would be impractically high on our platform. The

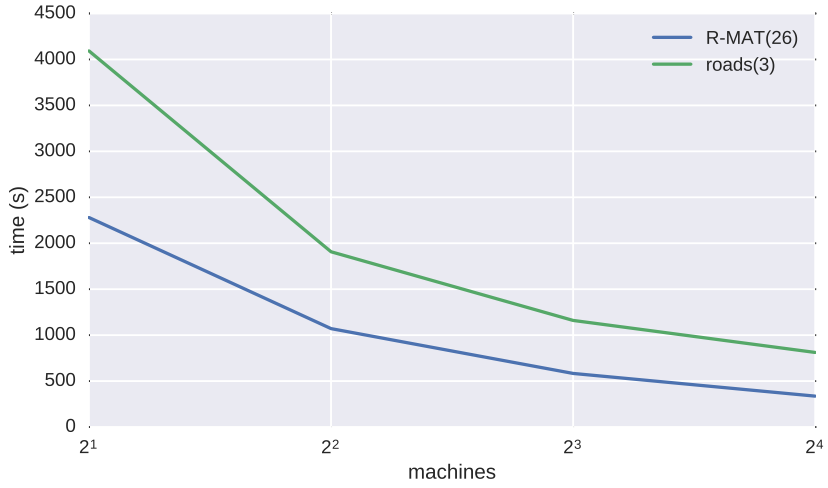

Figure 3.4: Scalability of CL-DIAM wrt the number of machines.

running times on 16 machines are shown in Table 3.3. Consider that the size of `R-MAT(29)` is 32 times larger than the size of `R-MAT(24)`, in terms of number of edges. Similarly, the number of edges of `roads(32)` is about 57 times larger than the size of `roads-USA`, because of the edges between each layer. Table 3.3 shows that CL-DIAM is ≈ 43 (resp. ≈ 88) times slower on `R-MAT(29)` (resp. `roads(32)`) with respect to `R-MAT(26)` (resp. `roads-USA`). Given this, the experiment shows that CL-DIAM's performance scales well with the graph size on the same machine configuration, with a slight penalty in the running time on large graphs, which is to be expected due to the increased number of interactions with the disks occurring in each machine because of the large graph size.

Altogether, the experiments suggest that our algorithm can be effectively employed to provide a good estimate of the diameter of huge graphs on sufficiently large clusters of commodity processors.

3.7.2 Experiments on unweighted graphs

As we showed in Section 3.5, on unweighted graphs our algorithm runs potenitally faster than in the general case. Moreover, some other diameter approximation algorithms, like HADI [Kan+11], only work on unweighted graphs. Hence, we run on a set of unweighted graphs whose main characteristics are reported in Table 3.4. The first graph is a symmetrization of the biggest connected component of the Twitter network introduced in the previous section. The next four graphs are from the SNAP collection[5] and represent, respectively, the Livejournal social network, also described in the previous section, and three road networks, which are the road networks of California, Pennsylvania, and Texas. The weights on the edges of the road networks have been ignored. The last graph is a synthetic 1000×1000 mesh, which has been included since its doubling dimension is known, unlike the other graphs, and constant ($b = 2$), hence it is an example of a graph where our algorithms are provably effective.

[5]http://snap.stanford.edu/data

Dataset	n	m	$\Phi(G)$
twitter	39,774,960	684,451,342	16
livejournal	3,997,962	34,681,189	21
roads-CA	1,965,206	2,766,607	849
roads-PA	1,088,092	1,541,898	786
roads-TX	1,379,917	1,921,660	1,054
mesh1000	1,000,000	1,998,000	1,998

Table 3.4: Unweighted benchmark graphs. $\Phi(G)$ is the diameter.

We run two sets of experiments: one aiming at assessing the quality of the solution found by our algorithm relative to the main competitors, the other at comparing different clustering strategies. The graphs considered in this section are slightly smaller than the ones used previously to ease the comparison with the competitors, which are prohibitively slow when run on large graphs.

Comparison with HADI and the BFS-based approximation algorithm

We performed three sets of experiments to investigate different aspects of our algorithm applied to unweighted graphs.

The first set of experiments aimed at testing the quality of the diameter approximation provided by our algorithm, and its dependence on the clustering granularity. The results are reported in Table 3.5. For each graph of Table 3.4 we estimated the diameter by running our algorithm with two clusterings of different granularities (dubbed *coarser* and *finer* clustering, respectively) reporting, in each case, the number of nodes (n_C) and edges (m_C) of the auxiliary graph G_C, the approximation $\Phi_{approx}(G)$ and the true diameter $\Phi(G)$. We observe that in all cases $\Phi_{approx}(G)/\Phi(G) < 2$. Also, we observe that, consistently with the theoretical results, the quality of the approximation does not seem to be significantly affected by the granularity of the clustering. Therefore, for very large graphs, or distributed platforms where individual machines are provided with small local memory, one can resort to a very coarse clustering in order to fit the whole auxiliary graph in one machine, and still obtain a good approximation to the diameter, at the expense, however, of an increased number of rounds, which are needed to compute the clustering.

With the second set of experiments, we assessed the time performance of our algorithm against two competitors: HADI [Kan+11], which was reviewed in Section 3.1.3 and provides a rather tight diameter estimation through the approximation of the neighborhood function; and Breadth First Search (BFS), which, as the Δ-stepping algorithm employed in the previous section, can be used to obtain an upper bound to the diameter within a factor two. The original code for HADI was written for the Hadoop framework[6]. Because of Hadoop's known large overhead, for fairness, we reimplemented HADI in Spark, with a performance gain of at least

[6]HADI website: www.cs.cmu.edu/~pegasus

	Coarser clustering			Finer clustering		
Dataset	n_C	m_C	approx.	n_C	m_C	approx.
twitter	1835	18865	1.43	5276	895356	1.68
livejournal	1933	24442	1.38	7837	570608	1.38
roads-CA	1835	5888	1.77	3863	10946	1.73
roads-PA	1087	3261	1.57	4286	12314	1.58
roads-TX	1316	3625	1.48	3821	10880	1.52
mesh1000	880	3224	1.06	3588	14198	1.01

Table 3.5: Size of the auxiliary graph, in terms of number of nodes a and edges, and diameter approximation returned by CL-DIAM on the benchmark graphs of Table 3.4. The approximation ratio is the ratio $\Phi_{approx}(G)/\Phi(G)$, where $\Phi_{approx}(G)$ is the estimated diameter and $\Phi(G)$ is the true diameter.

Dataset	CL-DIAM	BFS	HADI
twitter	61 (1.68)	45 (1.37)	3697 (0.875)
livejournal	9 (1.38)	15 (1.24)	388 (1.24)
roads-CA	21 (1.73)	191 (1.67)	11008 (0.98)
roads-PA	15 (1.58)	173 (1.58)	10090 (0.97)
roads-TX	24 (1.52)	260 (1.39)	12572 (0.94)
mesh1000	16 (1.01)	368 (1.11)	17287 (1.00)

Table 3.6: Comparison of our approach (CLUSTER) with HADI and BFS. Numbers in parentheses are the approximation factors, computed as the ratio $\Phi_{approx}(G)/\Phi(G)$ between the estimated diameter and the true diameter.

one order of magnitude. We did not compare our algorithm with the other two neighbourhood function-based algorithms, ANF and HYPERANF [PGF02; BRV11], since they have been developed for shared-memory multiprocessor architectures, which are different from our target architecture. In fact, we expect that if implemented on distributed-memory platforms, the performance of both ANF and HyperANF for high-diameter graphs is negatively affected by their long critical paths, as is the case for HADI e BFS. By comparing with HADI, we compare with the algorithmic stratecy based on the neighbourhood function, which is the core idea of the other two algorithms as well. As for the BFS, we implemented a simple and efficient version in Spark. Table 3.6 reports the running times and the diameter estimates obtained with the three algorithms where, for our algorithm, we used the finer clustering granularity adopted in the experiments described earlier. The figures in the table clearly show that HADI, while yielding a very accurate estimate of the diameter, is much slower than our algorithm, by orders of magnitude for large-diameter graphs. This is due to the fact that HADI requires $\Theta(\Phi(G))$ rounds and in each round the communication volume is linear in the number of edges of the input graph. On the other hand BFS, whose approximation guarantee is similar

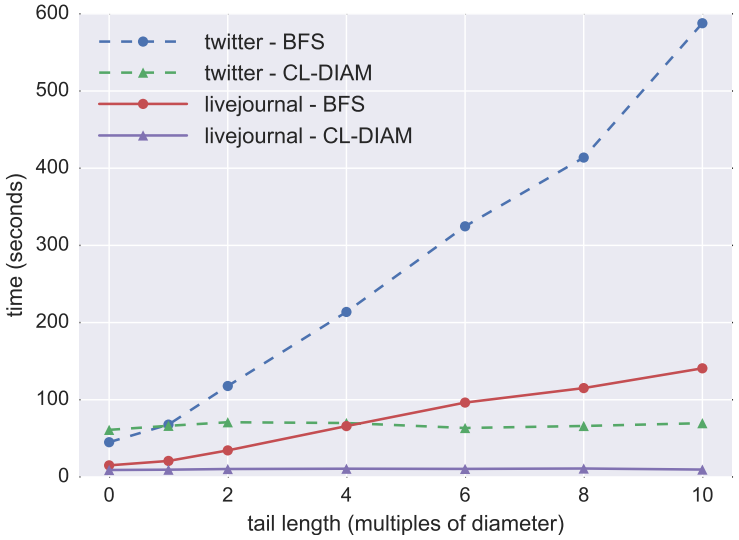

Figure 3.5: Performance of CLUSTER and BFS on graphs with small variations.

to ours in practice, outperforms HADI and, as expected, is considerably slower than our algorithm on large-diameter graphs. Indeed, BFS still requires $\Theta(\Phi(G))$ rounds as HADI, but its aggregate communication volume (rather than the per round communication volume) is linear in the number of edges of the input graph.

A desirable feature of our strategy is its capability to adapt to irregularities of the graph topology, which may have a larger impact on the performance of the other strategies. In order to provide experimental evidence of this phenomenon, our third set of experiments reports the running times of our algorithm and BFS on three variants of the two small-diameter graphs (livejournal and twitter) obtained by appending a chain of $c \cdot \Phi(G)$ extra nodes to a randomly chosen node, with $c = 1, 2, 4, 6, 8, 10$, thus increasing the diameter accordingly, without substantially altering the overall structure of the base graph. The plots in Figure 3.5 clearly show that while the running time of our algorithm is basically unaltered by the modification, that of BFS grows linearly with c, as expected due to the strict dependence of the BFS number of rounds from the diameter. A similar behaviour is to be expected with HADI for the same reason.

Comparison of Clustering Algorithms

We compared the quality of the clustering returned by algorithm CLUSTER against that of the clustering returned by the algorithm presented in [MPX13] and reviewed in Section 2.4.2, which we call MPX, for brevity. Recall that MPX assigns to each node $u \in V$ a random time shift δ, drawn from an exponential distribution of with mean $1/\beta$, with β set by the user. Then, it starts growing a cluster centered at u starting at time $\delta_{max} = \delta_u$, where δ_{max} is the maximum shift, unless u has not been already covered by some other cluster. The authors of [MPX13] show that the radius of clusters built this way is $O((\log n)/\beta)$. Moreover, the expected number of edges between clusters is at most $O(\beta m)$. We omit from the comparison the algorithm

	Algorithm CL-DIAM			Algorithm MPX		
Dataset	n_C	m_C	r	n_C	m_C	r
twitter	40001	17216285	5	41431	109348	6
livejournal	4020	230326	7	5796	17098	9
roads-CA	15038	40597	31	16429	34021	61
roads-PA	7710	13300	30	8529	18446	58
roads-TX	10653	28582	30	11238	23308	55
mesh1000	7641	18476	34	9112	25885	56

Table 3.7: Comparison between the clusterings returned by CLUSTER and MPX. n_C is the number of clusters, m_C is the number of edges between clusters, and r is the maximum cluster radius.

from [Mey08], which we also reviewed in Section 3.1.3, since its reliance on an Euler tour of a spanning tree makes it inherently sequential, hence impractical for very large graphs.

Recall that CLUSTER uses a parameter τ to control the number of cluster centers. As for MPX the parameter β we described above indirectly controls the number of clusters. Both algorithms aim at computing a clustering of the graph into clusters of small radius, so we focused the experiments on comparing the maximum radius of the returned clusterings. However, since the minimum maximum radius attainable by any clustering is a nonincreasing function of the number of clusters, but neither algorithm is able to precisely fix such a number a priori, we structured the experiments as follows.

We aimed at clustering granularities (i.e., number of clusters) which are roughly three orders of magnitude smaller than the number of nodes for small-diameter graphs, and roughly two orders of magnitude smaller than the number of nodes for large-diameter graphs. We ran MPX and CLUSTER setting their parameters β and τ so as to obtain a granularity close enough to the desired one, and compared the maximum cluster radius obtained by the two algorithms. In order to be conservative, since our algorithm features a better control on the number of returned clusters, we first ran CLUSTER with a value of τ suitably chosen to obtain the desired number of clusters. Then we searched a value of β such that MPX returned a comparable but larger number of clusters, thus giving it a slight advantage.

Table 3.7 shows the results of the experiments for the benchmark graphs. Each row reports the graph, and, for each algorithm, the number of nodes (n_C) and edges (m_C) of the auxiliary graph associated with the clustering (whose construction is described in Section 3.3), and the maximum cluster radius (r). Note that, in the auxiliary graph, multiple edges between the same pair of clusters are represented by a single edge. The table provides a clear evidence that our algorithm is more effective in keeping the maximum cluster radius small, especially for graphs of large diameter. This is partly due to the fact that MPX starts growing only a few clusters, and before more cluster centers are activated the radius of the initial clusters is already grown large. On the other hand, MPX is often more effective in reducing

the number of edges of the auxiliary graph, which is in fact its main objective. This is particularly evident for the first two graphs in the table, which represent social networks, hence feature low diameter and high expansion (thus, probably, high doubling dimension). In these cases, the few clusters initially grown by MPX are able to absorb entirely highly expanding components, thus resulting in a more drastic reduction of the edges.

Chapter 4

Clustering uncertain graphs

In Chapter 3, we dealt with deterministic graphs and the approximation of their diameter. Oftentimes, however, the nature of the data is such that it presents some level of uncertainty. In these cases, this uncertainty in the relationships between entities of the graph can be modeled by associating existence probabilities to each edge: such a graph is called an *uncertain graph*. For example, in Protein-Protein Interaction (PPI) Networks, an edge between two proteins corresponds to an interaction that is observed through a noisy experiment characterized by some level of uncertainty. This uncertainty can be conveniently cast as the probability of existence of that edge [Ast+04]. Also, in social networks, the probability of existence of an edge between two individuals may be used to model the likelihood of an interaction between the two individuals, or the influence of one of the two over the other [AR07]. Other applications of uncertainty in graphs arise in the realm of mobile ad-hoc networks [BM05; Gho+07] and graph obfuscation for privacy enforcement [Bol+12].

While in the previous chapter we used clustering as a means of building a succinct representation of the input, in this chapter we study clustering in itself. The main objective is to introduce novel strategies for clustering uncertain graphs, aiming at partitioning the node set in such a way that a suitable measure related to the probability of connectivity inside the clusters is maximized.

In accordance with the mainstream literature, an *uncertain graph* is defined over a set of nodes V, a set of undirected edges E between nodes of V, and a probability function $p : E \to (0, 1]$. We denote such a graph by $\mathcal{G} = (V, E, p)$, and we adopt the *possible world* semantics [Pot+10]: $\mathcal{G}$ can be viewed as a probability space whose outcomes (referred to as *possible worlds*) are graphs $G = (V, E')$, where $E' \subseteq E$ and any edge $e \in E$ is included in E' with probability $p(e)$, independently of other edges. Note that, even for graphs of moderate size, the exponential number of possible worlds puts computational problems on uncertain graphs in the big data realm by rights. In this case, instead of the *explicit* size of the input, is its *implicit* dimension that rules out traditional approaches, and requires the design of new

algorithmic strategies.

Clustering uncertain graphs is challenging for several orthogonal reasons. As observed in [Lin+12], aiming at high probability of connectivity within clusters is certainly a good criterion but it is hard to pursue, both because of the inherent difficulty of clustering per se, and because the estimation of cluster-wide connection probabilities in the uncertain graph scenario is #P-complete, as we will see in the next section. Also, it has been observed [Pot+10; Lin+12] that the straightforward reduction to the analysis of a deterministic graph where edge probabilities become weights may not yield significant outcomes because it disregards the possible world semantics.

Therefore, instead of adopting a costly reliability measure defined on whole clusters, we focus on the *connection probability* of pairs of nodes (also known as two-terminal reliability) as a measure upon which base our clustering. This is the probability that the two nodes belong to the same connected component in a random possible world. As a first technical contribution, which may be of independent interest for the broader area of network reliability, we show that this measure satisfies some form of triangle inequality, unlike other measures used in previous works. This property allows us to cast the problem of clustering uncertain graphs into the same framework of traditional clustering approaches on metric spaces while still enabling an effective integration with the possible world semantics.

We study the following clustering problem, together with some variations. Given in input an uncertain graph $\mathcal{G}$ and an integer τ, we seek a partition of the nodes of $\mathcal{G}$ into τ clusters, each with a *center* vertex, such that the minimum connection probability of a node to the center of its cluster is maximized. For this problem, we develop a simple deterministic algorithm based on a sequential cluster growing strategy which returns a clustering where the minimum connection probability of a node to the center of its cluster is $\Omega\left(p_{opt}^2(\tau)\right)$, where $p_{opt}(\tau)$ is the connection probability of the optimal solution. We also show how to adapt the strategy to approximate the somewhat dual problem of determining, for a fixed threshold q for the connection probability, a clustering of minimum cardinality whose featured connection probability is at least q. In order to speed up the cluster-growing process, we also develop an alternative strategy made of only a logarithmic number of concurrent cluster growing phases, which attains the same approximation quality as the sequential cluster-growing strategy at the expense of an increase in the number of clusters by an at most doubly-logarithmic factor. In fact, we show that this blow-up in the number of clusters can be eliminated at the expense of worsening the connection probability guarantee. We also describe a variant of our algorithms that allows to impose a limit on the length of the paths that contribute to the connection probability between two nodes.

For the sake of clarity, the algorithms introduced above, which we will describe in detail in Sections 4.3 and 4.4, are presented assuming the existence of an oracle for the connection probabilities. Section 4.5 will present a variant of our algorithms where only limited-depth paths are considered in the definition of connection probabilities. In Section 4.6, we will see how to estimate connection probabilities in an efficient way. Specifically, we will integrate a progressive sampling scheme for the Monte-Carlo estimation of the required probabilities which essentially preserves

the approximation quality. While the Monte-Carlo estimation can be very computationally intensive for small probabilities, our clustering strategies tackle this obstacle by only requiring precise estimates of probabilities not much smaller than the value of the optimal solution.

Finally, in order to validate our approaches, in Section 4.7 we devise a preliminary suite of experiments aimed both at evaluating the relative efficiency of our strategies and at comparing the performance of the returned clustering against some competing strategies.

The material of this chapter is the account of ongoing work [Cec+16a].

4.1 Related work

4.1.1 Network reliability

Uncertain graphs have been studied, even if somewhat implicitly, in the context of network reliability. Given an uncertain graph, we can interpret edge probabilities as the complement of failure probabilities. A typical objective of network reliability analysis is to determine the probability that a given set of nodes is connected under random failures. Such a probability can be estimated with a Monte-Carlo approach. The drawback of this approach is that it becomes prohibitively expensive for very low probability values (for further details, see Section 4.2). Actually, even the simplest problem of estimating the probability that two distinguished nodes s and t are connected is known to be #P-complete [Val79; Bal86]. In the last three decades, several works have tried to come up with better heuristics for various reliability problems on uncertain graphs. For a more detailed account of the research on uncertain graphs in the context of network reliability, see [Jin+11] and references therein.

4.1.2 Uncertain graphs

The definition of uncertain graph that we adopt was introduced in [Pot+10], where the authors investigate several probabilistic notions of distance between nodes, and develop efficient algorithms for determining the k nearest nodes of a given source under the various distance measures. It has to be remarked that the proposed measures do not satisfy the triangle inequality, thus ruling out the applicability of traditional graph clustering approaches [Sch07].

Boldi et al. [Bol+12] studied the problem of turning a deterministic graph G into an uncertain graph $\mathcal{G}$ for the purpose of identity obfuscation. The main goal of their work is to build the uncertain graph $\mathcal{G}$ in such a way that individual vertices of the original graph G can no longer be identified by their topological properties. At the same time some global properties of G, such as degree distribution, diameter, and clustering coefficient are preserved in expectation over the possible worlds of $\mathcal{G}$. In particular, the authors introduce the concept of a (k, ε)-obfuscation with respect to a given property P of the original graph G. In the paper [Bol+12], the authors consider the degree distribution as the property P. The parameter $k \geqslant 1$ is related to the entropy of the distribution of property P in the uncertain graph: a higher k implies a higher level of obfuscation. $\varepsilon \in (0, 1]$ is a tolerance parameter, and

controls the fraction of nodes which are not obfuscated. This tolerance ε is allowed because some networks have a small set of clearly identifiable nodes, for which obfuscation is difficult and not needed. For instance, in a social network, the profile of a pop-star does not need to be obfuscated. The basic idea of the algorithm is the following. For a given graph $G = (V, E)$, the algorithm builds an uncertain graph $\mathcal{G}$ with V as the vertex set. As for the edge set, the algorithm selects a subset of $V \times V$ of cardinality $c \cdot |E|$, where c is a user-defined parameter. Edge probabilities are then suitably assigned so as to ensure that the result graph is a (k, ε)-obfuscation of G. The authors provide an implementation of their obfuscation algorithm, which we will use in Section 4.7 to generate synthetic uncertain graphs.

A detailed account of the state of the art on the subject of uncertain graphs can be found in [Par+15]. In the same work, the authors investigate the problem of extracting a single representative possible world providing a good summary of an uncertain graph for the purposes of query processing.

A number of recent works have studied different forms of clustering of uncertain graphs. In [KPT13] the authors consider the identification of a deterministic *cluster graph*, which corresponds to a clique-cover of the nodes of the uncertain graph. The aim is to minimize the expected *edit distance* between the clique-cover and a random possible world of the uncertain graph, where the edit distance is measured in terms of edge additions and deletions. The authors provide a 5-approximation algorithm for this problem in the paper. The main drawback of this approach is that the formulation of the clustering problem does not allow to control the number of clusters. Moreover, the approximate solution returned by the proposed algorithm relies on a shallow star-decomposition of the topology of the uncertain graph, which does not exploit more systemic information about its connectivity properties. The same clustering problem has been also studied by Gu et al. in [Gu+14]. They consider a more general class of uncertain graphs where the assumption of edge independence is lifted and the existence of an edge (u, v) is correlated to the existence of its adjacent edges (i.e., edges incident on either u or v). The authors propose two algorithms for this problem, one that, as in [KPT13], does not fix a bound on the number of returned clusters, and another that fixes such a bound. Neither algorithm provides worst-case guarantees on the approximation ratio.

Liu et al. [Lin+12] define a clustering problem with the objective of minimizing the expected entropy of the returned clustering, defined with respect to the adherence of the clustering to the connected components of a random possible world. With this objective in mind, the authors develop a clustering algorithm which combines a standard k-means strategy with the Monte-Carlo sampling approach for reliability estimation. No theoretical guarantee is offered on the quality of the returned clustering with respect to the defined objective function. Moreover, the efficiency of the algorithm, which depends on a convergence parameter that cannot be estimated analytically, does not appear to scale well with the size of the graph. In summary, while the pursued approach to clustering has merit, there is no rigorous analysis of the tradeoffs that can be exercised between the quality of the returned clustering and the running time of the algorithm.

In [Don08] the Markov Cluster Algorithm (MCL) is proposed for clustering weighted graphs. In particular, in MCL weights are considered as a *similarity score* between the endpoints. The algorithm does not specifically target uncertain graphs,

but it can be run on these graphs by considering the edge probabilities as weights. In fact, some of the aforementioned works on clustering of uncertain graphs have used MCL for comparison purposes. The algorithm focuses on finding so-called *natural clusters*, that is sets of nodes characterized by the presence of many edges and paths between their members. The basic idea of the algorithm is then to simulate random walks on the graph, defining the clustering based on the probability of a random walk to stay within a cluster. Edge weights (i.e. the similarity scores) are used by the algorithm to define the probability that a given random walk traverses a given edge. The algorithm's behaviour is controlled with a parameter, called *inflation*, which indirectly controls the granularity of the clustering. The author of the algorithm provides a very efficient implementation of MCL, with which we will compare our algorithms in Section 4.7.

4.2 Preliminaries

Let $\mathcal{G} = (V, E, p)$ be an uncertain graph, as defined in the introduction. In accordance with the established notation used in previous work, we write $G \sqsubseteq \mathcal{G}$ to denote that G is a possible world of $\mathcal{G}$. Given two nodes $u, v \in V$, the probability that they are connected (an event denoted as $u \sim v$) in a random possible world can be defined as

$$\Pr[u \sim v] = \sum_{G \sqsubseteq \mathcal{G}} \Pr[G]\, \mathbf{I}_G(u, v), \quad \text{where} \quad \mathbf{I}_G(u, v) = \begin{cases} 1 & \text{if } u \sim v \text{ in } G \\ 0 & \text{otherwise} \end{cases}$$

We refer to $\Pr[u \sim v]$ as the *connection probability* between u and v in $\mathcal{G}$. The uncertain graphs we consider are undirected and no weights are attached to their nodes/edges. Therefore, also the possible worlds of an uncertain graph are undirected and unweighted graphs.

Recall that in Section 2.4 we defined a k-clustering as a partition of a set S in k subsets according to some criterion. Similarly, for an uncertain graph $\mathcal{G} = (V, E, p)$, we define a τ-*clustering* as a partition of V into τ *clusters* $C_1, \ldots, C_\tau$ and a set of *centers* $c_1, \ldots, c_\tau$ with $c_i \in C_i$, for $1 \leqslant i \leqslant \tau$. We study the problem of determining a τ-clustering which *maximizes* the following objective function

$$\min_{1 \leqslant i \leqslant \tau} \min_{v \in C_i} \Pr[c_i \sim v]\,. \tag{4.1}$$

We denote the optimum value of the above objective function with $p_{\text{opt}}(\tau)$. In other words, we want the clusters to be well connected, in the sense that each node is likely to be connected to its cluster's center in a random possible world. We can rephrase this problem as an instance of the k-center problem as shown in the following.

4.2.1 A triangle inequality for connection probabilities

For every pair of nodes $u, v \in V$ define $d(u, v) = \ln(1/\Pr[u \sim v])$ with the understanding that $d(u, v) = \infty$ if $\Pr[u \sim v] = 0$. Under this transformation, the aim becomes the *minimization* of the following objective function

$$\max_{1 \leqslant i \leqslant \tau} \max_{v \in C_i} d(c_i, v). \tag{4.2}$$

The above objective function is clearly reminiscent of the k-center objective function (see Definition 3 in Chapter 2). We want the function $d(\cdot,\cdot)$ to satisfy the triangle inequality, since this would allow us to develop efficient algorithms. The triangle inequality for $d(\cdot,\cdot)$ is expressed as follows

$$\ln \frac{1}{\Pr[u \sim v]} \leqslant \ln \frac{1}{\Pr[u \sim z]} + \ln \frac{1}{\Pr[z \sim v]} \quad \forall u, v, z \in V$$

The above relation can be cast in terms of probabilities as

$$\Pr[u \sim v] \geqslant \Pr[u \sim z] \cdot \Pr[z \sim v] \quad \forall z \in V \tag{4.3}$$

In what follows we prove that the above inequality actually holds for pairwise connection probabilities in uncertain graphs. Fix an arbitrary edge $e \in E$ and let $A(e)$ be the event: "edge e is present". We need the following technical lemma.

Lemma 5. *For any $u, v \in V$, $e \in E$, we have* $\Pr[u \sim v | A(e)] \geqslant \Pr[u \sim v | \neg A(e)]$

Proof. Let $\mathcal{G}_e^{u,v}$ (resp., $\mathcal{G}_{\neg e}^{u,v}$) be the set of possible worlds where $u \sim v$, and edge e is present (resp., not present). We have that

$$\Pr[u \sim v | A(e)] = \sum_{G \sqsubseteq \mathcal{G}_e^{u,v}} \Pr[G] / p(e)$$
$$\Pr[u \sim v | \neg A(e)] = \sum_{G \sqsubseteq \mathcal{G}_{\neg e}^{u,v}} \Pr[G] / (1 - p(e)).$$

The lemma follows by observing that for any graph G in $\mathcal{G}_{\neg e}^{u,v}$ the same graph with the addition of e belongs to $\mathcal{G}_e^{u,v}$, and the corresponding terms in the two summations are equal. □

We can now prove that Inequality (4.3) holds.

Theorem 7. *For any uncertain graph $\mathcal{G} = (V, E, p)$ and any triplet $u, v, z \in V$, we have:*

$$\Pr[u \sim v] \geqslant \Pr[u \sim z] \cdot \Pr[z \sim v].$$

Proof. The proof proceeds by induction on the number k of *uncertain edges*, that is, edges $e \in E$ with $p(e) \neq 0$ and $p(e) \neq 1$. Fix three arbitrary nodes $u, v, z \in V$. The base case $k = 0$ is trivial, since, in this case, the uncertain graph is deterministic and for each pair of nodes $x, y \in V$, $\Pr[x \sim y]$ is either 1 or 0, which implies that when $\Pr[u \sim z] \cdot \Pr[z \sim v] = 1$, then $\Pr[u \sim v] = 1$ as well. Suppose that the property holds for uncertain graphs with at most k uncertain edges, with $k \geqslant 0$, and consider an uncertain graph $\mathcal{G} = (V, E, p)$ with $k + 1$ uncertain edges. Fix an arbitrary edge $e \in E$ and let A_e denote the event that edge e is present. For any two arbitrary nodes $x, y \in V$, we can write

$$\begin{aligned} \Pr[x \sim y] &= \Pr[x \sim y | A(e)] \cdot p(e) + \Pr[x \sim y | \neg A(e)] \cdot (1 - p(e)) \\ &= (\Pr[x \sim y | A(e)] - \Pr[x \sim y | \neg A(e)]) \cdot p(e) + \Pr[x \sim y | \neg A(e)]. \end{aligned} \tag{4.4}$$

By Lemma 5, the term multiplying $p(e)$ in the above expression is nonnegative. As a consequence, we have that

$$\Pr[u \sim z] \cdot \Pr[z \sim v] - \Pr[u \sim v] = A \cdot (p(e))^2 + B \cdot p(e) + C,$$

for some constants A, B, C independent of $p(e)$, with $A \geqslant 0$. Therefore, the maximum value of $\Pr[u \sim z] \cdot \Pr[z \sim v] - \Pr[u \sim v]$, as a function of $p(e)$, is attained for $p(e) = 0$ or $p(e) = 1$. Since in either case the number of uncertain edges is decremented by one, by the inductive hypothesis, the difference must yield a nonpositive value, hence the theorem follows. □

Now that we have a relation similar to the triangle inequality for pairwise connection probabilities, we could apply existing k-center algorithms such as GMM to solve our clustering problem. However, as we shall see in the following, things do not go so smoothly, because connection probabilities must be estimated, and the cost of this operation rules out GMM and similar approaches.

4.2.2 Estimating connection probabilities

We know that, given two nodes $u, v \in V$, the exact computation of $\Pr[u \sim v]$ is #P-complete [Bal86]. However, for reasonably large values of this probability a very accurate estimate can be obtained through Monte-Carlo sampling. More precisely, for $r > 0$ let $G_1, \ldots, G_r$ be r sample possible worlds drawn independently at random from $\mathcal{G}$. For any pair of nodes u and v we can define the following estimator

$$\tilde{q}(u, v) = \frac{1}{r} \sum_{i=1}^{r} \mathbf{I}_{G_i}(u, v) \tag{4.5}$$

It is easy to see that $\tilde{q}(u, v)$ is an unbiased estimator of $\Pr[u \sim v]$. Moreover, for fixed $\varepsilon \in (0, 1]$ and $\delta \in (0, 1)$, by taking

$$r \geqslant \frac{3 \ln \frac{2}{\delta}}{\varepsilon^2 \Pr[u \sim v]} \tag{4.6}$$

samples, we have that $\tilde{q}(u, v)$ is an (ε, δ)-approximation of $\Pr[u \sim v]$, that is,

$$\Pr\left[\frac{|\tilde{q}(u, v) - \Pr[u \sim v]|}{\Pr[u \sim v]} \leqslant \varepsilon\right] \geqslant 1 - \delta \tag{4.7}$$

(e.g., see [MU05, Theorem 10.1]). Hence, given a lower bound on $\Pr[u \sim v]$, we can choose a number of samples r which ensures that the actual connection probability between u and v is estimated with small relative error, with a desired confidence level. However, when $\Pr[u \sim v]$ approaches 0, the number of samples, hence the work, required to attain an accurate estimation becomes prohibitively large. As we will see in the next section, this limitation is a problem for the application of existing clustering algorithms.

4.3 Sequential cluster growing

In the previous section, we mentioned how the existence of relation similar to the triangle inequality for connection probabilities could enable us to apply the GMM algorithm for clustering uncertain graphs. Recall from Section 2.4.1 that GMM relied on the repeated determination of the *farthest* node from the centers selected

so far. In our scenario, using $d(u, v) = \ln(1/\Pr[u \sim v])$ as a distance function, this approach might entail the computation of very small values of $\Pr[u \sim v]$, for some pair of nodes (u, v). As we pointed out before, this is computationally inefficient. Consequently, we have to resort to strategies which are able to create clusters based on *proximity* rather than on *farness*. In this section, we present two proximity-based algorithms for clustering uncertain graphs, with provable approximation guarantees. First, we present a simple deterministic algorithm based on a sequential growing strategy; secondly we introduce a variant which, for a target connection probability, aims at minimizing the number of returned clusters. The presentation of these algorithms assumes the existence of an oracle that, given two nodes $u, v \in V$, returns $\Pr[u \sim v]$. The discussion of the Monte-Carlo estimation of such probability, and its integration with the algorithms presented in this section, is deferred to Section 4.6.

4.3.1 Clustering with a target number of clusters

Consider an uncertain graph $\mathcal{G} = (V, E, p)$. Recall that $p_{opt}(\tau)$ denotes the optimum value of the objective function of Equation (4.1), and observe that $p_{opt}(\tau) > 0$ if and only if the uncertain graph $\mathcal{G}$ features no more than τ connected components. For convenience, in the rest of the presentation we assume that this is the case. Algorithm 6 describes a simple sequential cluster growing strategy that returns a τ-clustering which approximates $p_{opt}(\tau)$. The algorithm features a design parameter $\gamma \in (0, 1)$ that can be employed to exercise a quality-performance tradeoff.

Before describing the algorithm in detail, we have to define two helper functions. Let $c(u, S) = \arg\max_{c \in S}\{\Pr[c \sim u]\}$ be the function that, given a node $u \in V$ and a set of nodes $S \subset V$, returns the node $\in S$ connected with u with the highest probability. Moreover, we define $\pi(u, S) = \Pr[c(u, S) \sim u]$, that is the maximum among the probabilities of connection of u with any node $\in S$.

The algorithm keeps track of a threshold probability p_{curr} and proceeds iteratively. In each iteration, it starts by selecting an arbitrary vertex $c_1 \in V$, which is inserted into the set of current centers S. Then, all the vertices $v \in V$ such that $\Pr[c_1 \sim v] \geqslant p_{curr}$ are marked as covered by c_j. These nodes are not yet assigned to the cluster centered in c_j. This operation is repeated by selecting a new cluster center from the unmarked nodes until there are no more unmarked nodes or τ centers have been selected. In the former case, the algorithm assigns each node $u \in V$ to the cluster centered in $c(u, S)$. In the latter, if some nodes have not been included in any cluster, the algorithm updates p_{curr} with a schedule controlled with the parameter γ, and starts building a new clustering from scratch.

It is easy to see that the algorithm terminates returning an ℓ-clustering, with $\ell \leqslant \tau$, since p_{curr} cannot drop below half the minimum connection probability between two nodes in the same connected component of $\mathcal{G}$ before the termination condition is reached. Let $C_1, \ldots, C_\ell$ be the clusters returned by the algorithm centered at $c_1, \ldots, c_\ell$, and let p_{alg} be the value of the objective function corresponding to this clustering, namely

$$p_{alg} = \min_{1 \leqslant i \leqslant \ell} \min_{v \in C_i} \Pr[c_i \sim v] .$$

Recall that we defined $p_{opt}(\tau)$ as the objective function of an optimal τ-clustering

Algorithm 6: SEQUENTIALCLUSTER$(\mathcal{G}, \tau, \gamma)$

```
p_curr ← 1;
while true do
    S ← ∅;                                  ▷ Centers selected by the algorithm
    V_1 ← V;
    for i ← 1 to τ do
        select one arbitrary cluster center c_i ∈ V_i;
        M_i ← {u ∈ V_i : Pr[c_i ~ u] ⩾ p_curr};        ▷ Nodes marked by c_i
        V_{i+1} ← V_i − M_i;
        S ← S ∪ {c_i};
        if V_{i+1} = ∅ then
            for u ∈ V do
                Assign u to the cluster centered in c(u, S);
            end
            return the clusters centered at c_1, c_2, ..., c_i;
        end
    end
    p_curr ← p_curr/(1 + γ)
end
Error "No clustering found";
```

$\hat{C}_1, \ldots, \hat{C}_\tau$, with centers $\hat{c}_1, \ldots, \hat{c}_\tau$. The quality of the approximation is guaranteed by the following theorem.

Theorem 8. *Algorithm 6 terminates after at most* $\lfloor 2\log_{1+\gamma}(1/p_{\mathrm{opt}}(\tau))\rfloor + 1$ *iterations of the while loop and returns a clustering with*

$$p_{alg} \geqslant \frac{p^2_{opt}(\tau)}{(1+\gamma)}$$

Proof. Let $\ell = \lfloor 2\log_{1+\gamma}(1/p_{\mathrm{opt}}(\tau))\rfloor + 1$. Suppose that the algorithm reaches the ℓ-th iteration of the while loop and note that at the beginning of this iteration $p^2_{\mathrm{opt}}(\tau)/(1+\gamma) < p_{\mathrm{curr}} \leqslant p^2_{\mathrm{opt}}(\tau)$. Let $\hat{C}_{k_i}$ be the cluster in the optimal τ-clustering which contains c_i (the center selected in the i-th iteration of the for loop), for every $i \geqslant 1$. Moreover, let $\hat{c}_{k_i}$ be the center of $\hat{C}_{k_i}$. Observe that for every node $v \in \hat{C}_{k_i}$, and thanks to Theorem 7

$$\Pr[c_i \sim v] \geqslant \Pr[c_i \sim \hat{c}_{k_i}] \cdot \Pr[\hat{c}_{k_i} \sim v] \geqslant p^2_{\mathrm{opt}}(\tau) \geqslant p_{\mathrm{curr}}$$

Therefore, V_{i+1} cannot contain any node of $\hat{C}_{k_i}$. In fact, inductively, we have that V_{i+1} cannot contain any node of $\bigcup_{j=1}^{i} \hat{C}_{k_j}$. Therefore, V_{t+1} for some $t \leqslant \tau$ must be empty. The theorem follows by observing that $p_{\mathrm{alg}} \geqslant p_{\mathrm{curr}} \geqslant p^2_{\mathrm{opt}}(\tau)/(1+\gamma)$ in the last iteration of the algorithm. □

The above theorem also states the influence of the design parameter γ on both the approximation quality and the running time. Thanks to γ we can control the rate of the adjustment of the probability guess p_{curr}. Setting γ to a higher value

Algorithm 7: SEQUENTIALCLUSTER2$(\mathcal{G}, q)$

$V_1 \leftarrow V$; $i \leftarrow 1$;
while *true* **do**
  select one arbitrary cluster center $c_i \in V_i$;
  $C_i \leftarrow \{u \in V_i : \Pr[c_i \sim u] \geqslant q\}$;
  $V_{i+1} \leftarrow V_i \setminus C_i$;
  if $V_{i+1} = \emptyset$ **then return** $C_1, C_2, \ldots, C_i$;
  $i \leftarrow i + 1$;
end

makes the guessing more aggressive, allowing to converge to a solution faster. At the same time, however, this worsens the approximation ratio. Conversely, a smaller value makes the adjustment of p_{curr} more conservative, and consequently slower but more accurate.

4.3.2 Clustering with a target probability

In some application scenarios, it may be useful to control the connectivity within the clusters by imposing a lower bound on the connection probability of a node to the center of its cluster. Consider an uncertain graph $\mathcal{G} = (V, E, p)$. We can define an alterative clustering problem which, given a probability threshold q, requires to find a partition of nodes into a minimum number of clusters where the connection probabilities of the nodes to their assigend cluster centers is at least q. Let $\tau_{\text{opt}}(q)$ denote such minimum number of clusters. It is easy to see that $\tau_{\text{opt}}(q)$ is a non-decreasing function of q. Algorithm 7 computes an approximate solution to this problem. The algorithm proceeds iteratively: in iteration i, it selects from the unassigned nodes an arbitrary vertex c_i as a cluster center, greedily placing in the cluster centered in c_i all the unassigned nodes $v \in V$ such that $\Pr[c_i \sim v] \geqslant q$. The following theorem bounds the number τ_{alg} of clusters returned by the algorithm.

Theorem 9. *Algorithm 7 returns a τ-clustering with $\tau_{alg} \leqslant \tau_{opt}(\sqrt{q})$, where each node is connected to the center of its cluster with probability at least q.*

Proof. By construction, the algorithm generates clusters featuring the given connection probability. Consider an optimal $\tau(\sqrt{q})$-clustering, where each node is connected to the center of its cluster with probability at least $\sqrt{q}$. Note that by Theorem 7, two nodes c, u belonging to the same cluster of this optimal $\tau(\sqrt{q})$-clustering are connected with probability $\geqslant q$. An argument similar to the one used in the proof of Theorem 8 shows that at the end of the i-th iteration of the while loop all nodes which in the optimal $\tau(\sqrt{q})$-clustering belong to the same cluster as one of the c_j's, with $j \leqslant i$, must have been already included in some cluster. Hence, no more than $\tau_{\text{opt}}(\sqrt{q})$ clusters are returned by the algorithm. □

4.4 Concurrent cluster growing

The main drawback of the τ-clustering algorithm presented in the previous subsection (Algorithm 6) is the fact that for each of the $O\left(\log_{1+\gamma}(1/p_{\text{opt}}(\tau))\right)$ guesses of

Algorithm 8: ConcurrentCluster$(\mathcal{G}, \tau)$

$V_1 \leftarrow V$; $i \leftarrow 1$;
while $|V_i| > \alpha\tau \log n$ **do**
  select $\alpha\tau \log n$ new cluster centers at random with replacement from V_i;
  $S_i \leftarrow$ {newly selected centers};
  let $u_1, u_2, \ldots, u_{|V_i|}$ be the nodes of V_i in decreasing order of $\pi(\cdot, S_i)$;
  for $j \leftarrow 1$ **to** $|V_i|/2$ **do**
    assign u_j to cluster centered at $c(u_j, S_i)$;
  end
  $V_{i+1} \leftarrow \{u_j : j > |V_i|/2 \wedge u_j \notin S_i\}$;
  $i \leftarrow i + 1$;
end
return the clusters centered at $S_1 \cup S_2 \cup \cdots \cup S_{i-1} \cup V_i$;

the connection probability, τ clusters must be computed sequentially from scratch one after the other, which may be computationally cumbersome for large graphs and large values of τ. In this section, we propose an alternative and more efficient, but randomized, clustering algorithm (Algorithm 8) inspired by the strategy we developed for the diameter approximation problem discussed in Chapter 3 for deterministic graphs. On the one hand, this approach allows many clusters to be grown concurrently while, on the other, it embodies an implicit and more accurate guessing of the target connection probability. Consider an uncertain graph $\mathcal{G} = (V, E, p)$. The algorithm proceeds in *phases*. In each phase, a new batch of random centers is added and at least of the yet uncovered nodes are assigned to the clusters of these new centers, or to previously grown clusters. As in Chapter 3, the idea underlying this strategy is to push more clusters in regions of the graph with low connection probability. The clusters covering these regions have fewer nodes each and will not include nodes connected with very low probabilities.

Recall from Section 4.3.2 that, for $u \in V$ and $S \subset V$, $c(u, S) = \arg\max_{c \in S}\{\Pr[c \sim u]\}$ and $\pi(u, S) = \Pr[c(u, S) \sim u]$. As before, we assume the existence of an oracle that, given a pair of nodes $u, v \in V$, returns $\Pr[u \sim v]$. In Section 4.6 we will describe how to remove this assumption.

Let $\alpha > 0$ be a suitable constant which will be fixed in the analysis. The algorithm, whose pseudocode is given in Algorithm 8, works in phases as follows. In phase i, it first selects at random, from the nodes that are still uncovered, a batch of $\alpha\tau \log n$ centers. Let S_i be the set of newly selected centers, and let V_i be the set of uncovered nodes at the beginning of phase i. The algorithm then considers the nodes $u \in V_i$ by decreasing order of $\pi(u, S_i)$, i.e. by decreasing probability of connection to some center. It then assigns each node u of the first $|V_i|/2$ nodes in this ordering to the cluster centered at $c(u, S_i)$, before proceeding to phase $i + 1$.

Suppose that the algorithm returns ℓ clusters, $C_1, \ldots, C_\ell$, centered at $\{c_1, \ldots, c_\ell\}$, and note that the clusters form indeed a partition of V. As before, let p_{alg} be the value of the objective function corresponding to this clustering, namely

$$p_{alg} = \min_{1 \leqslant i \leqslant \ell} \min_{v \in C_i} \Pr[c_i \sim v] .$$

Recall that we defined $p_{opt}(\tau)$ as the objective function of an optimal τ-clustering $\hat{C}_1, \ldots, \hat{C}_\tau$, with centers $\hat{c}_1, \ldots, \hat{c}_\tau$. We have:

Theorem 10. *Algorithm 8 returns an ℓ-clustering with $\ell = O\left(\tau \log^2 n\right)$ and such that $p_{alg} \geqslant p^2_{opt}(\tau)$, with high probability.*

Proof. The upper bound on ℓ follows immediately by observing that the while loop executes less than $\log n$ iterations. In order to prove the lower bound on p_{alg}, consider a generic Iteration i of the while loop and let n_i be the number of nodes not yet assigned to clusters (i.e., $n_i = |V_i|$). It is sufficient to show that there exist at least $n_i/2$ nodes $u \in V_i$ such that $\pi(u, S_i) \geqslant p^2_{opt}(\tau)$. Define an optimal cluster $\hat{C}_j$ to be *large* if $|\hat{C}_j \cap V_i| \geqslant (n_i/(2\tau))$. It is easy to see that the optimal clusters which are not large account for less than half of the nodes of V_i, hence large clusters contain at least $n_i/2$ nodes of V_i. We now argue that at least one center of S_i falls into every large cluster $\hat{C}_j$, with high probability. Consider an arbitrary large cluster $\hat{C}_j$. The probability that no center of S_i is selected among the nodes of $\hat{C}_j \cap V_i$ is

$$\left(1 - \frac{|\hat{C}_j \cap V_i|}{n_i}\right)^{\alpha\tau \log n} \leqslant \left(1 - \frac{1}{2\tau}\right)^{\alpha\tau \log n} \leqslant e^{(1/2)\alpha \log n},$$

which can be made polynomially small in n for large enough α. Recall that $\hat{C}_j$ includes a node $\hat{c}_j$ such that $\Pr\left[\hat{c}_j \sim u\right] \geqslant p_{opt}(\tau)$, for every $u \in \hat{C}_j$. Suppose that a center $c \in \hat{C}_j \cap V_i$ is selected as a new center in the iteration. Then, for every $u \in \hat{C}_j \cap V_i$, by Theorem 7 we have that

$$\Pr[c \sim u] \geqslant \Pr\left[c \sim \hat{c}_j\right] \cdot \Pr\left[\hat{c}_j \sim u\right] \geqslant p^2_{opt}(\tau).$$

This immediately implies that $\pi(u, S_i) \geqslant \Pr[c \sim u] \geqslant p^2_{opt}(\tau)$. As observed before, there are at least $n_i/2$ nodes of V_i in large clusters, hence the statement follows. □

Observe that in contrast to Algorithm 6, Algorithm 8 lends itself to parallel execution, since it only $O(\log n)$ sequential phases where, in each phase, the assignment of nodes to clusters (i.e., the growing of clusters) can be don ein parallel. Moreover, it ensures a slightly better upper bound on the connection probability. However, these better features are counter-balanced by a number of clusters which is a factor $O\left(\log^2 n\right)$ larger than the target τ. We can reduce the number of clusters τ at the expense of an increased connection probability, as shown by the following corollary.

Corollary 3. *The clustering returned by Algorithm 8 can be transformed into a τ-clustering where each node is connected to its assigned cluster center with probability at least $p^4_{opt}(\tau)/16$.*

Proof. Again, suppose that the algorithm returns $\ell \in O\left(\tau \log^2 n\right)$ clusters $C_1, \ldots, C_\ell$, centered at $c_1, \ldots, c_\ell$. Assume for now that $p_{opt}(\tau)$ is known. Consider an auxiliary graph H whose nodes are the c_i's and whose edges are all pairs (c_i, c_j) such that $\Pr\left[c_i \sim c_j\right] \geqslant p^2_{opt}(\tau)$. Take one arbitrary center c_i and merge C_i and all clusters centered at the neighbors of c_i in H into one new cluster with center c_i. Let ν be a

node in this new merged cluster and suppose it originally belonged to C_j, where c_j is a neighbor of c_i in H. By Theorem 7, we have that

$$\Pr[v \sim c_i] \geqslant \Pr\left[v \sim c_j\right] \cdot \Pr\left[c_j \sim c_i\right] \geqslant p_{opt}^4(\tau).$$

Remove c_i and its neighbors from H and repeat the above procedure. We now show that no more than τ new clusters are created in this fashion. Consider a generic merging step where a center c_i and all of its neighbors in H are removed from H. Let $\hat{C}_{k_i}$ be the cluster in the optimal τ-clustering which contains c_i. Observe that any other center c_j that belongs to $\hat{C}_{k_i}$ is a neighbor of c_i in H, hence, all clusters centered at nodes of $\hat{C}_{k_i}$ are merged together in the step under consideration. As a consequence, no more than τ new clusters can be created in this fashion. Finally, if $p_{opt}(\tau)$ is not known, one can repeat the procedure for geometrically decreasing guesses of $p_{opt}(\tau)$, starting from $\sqrt{p_{alg}}$, where p_{alg} is the objective function value of the original ℓ-clustering returned by the algorithm, until τ or less clusters are obtained. Clearly, after no more than $O\left(\log(\sqrt{p_{alg}}/p_{opt}(\tau))\right)$ attempts, the last guess will be at least $p_{opt}(\tau)/2$, and the corollary follows. □

4.5 Limiting the path length

The algorithms described in the previous sections consider paths of any length to define the connection probability. In other words, for an uncertain graph $\mathcal{G} = (V, E, p)$, a possible world $G \sqsubseteq \mathcal{G}$ gives a positive contribution to the connection probability of a pair of nodes (u, v) if u and v are in the same connected component in G. Sometimes, however, we might be interested in considering only nodes which are topologically close. For instance, in Protein-Protein Interaction networks we might be interested in grouping in the same cluster only proteins which are connected with a short chain of interactions. In such cases, we impose a maximum length d on the paths used to define the connection probability.

More precisely, for a fixed integer d, with $1 \leqslant d \leqslant n$, we define with $u \overset{d}{\sim} v$ the event "u and v are connected by a path of length at most d". We can then define the d-*connection probability* between u and v as

$$\Pr\left[u \overset{d}{\sim} v\right] = \sum_{G \sqsubseteq \mathcal{G}} \Pr[G]\, \mathbf{I}_G(u, v; d) \tag{4.8}$$

where $\mathbf{I}_G(u, v; d)$ is 1 if $u \overset{d}{\sim} v$ in the possible world G, and 0 otherwise. By easily adapting the proofs of Lemma 5 and Theorem 7 we have that the following relation holds for any triplet $u, z, v \in V$

$$\Pr\left[u \overset{d}{\sim} v\right] \geqslant \Pr\left[u \overset{\lfloor d/2 \rfloor}{\sim} z\right] \cdot \Pr\left[z \overset{\lfloor d/2 \rfloor}{\sim} v\right]$$

Consider a τ-clustering $C_1, \ldots, C_\tau$ with centers $c_1, \ldots, c_\tau$. We now aim at the maximization of the following objective function.

$$\min_{1 \leqslant i \leqslant \tau} \min_{v \in C_i} \Pr\left[c_i \overset{d}{\sim} v\right] \tag{4.9}$$

We denote by $p_{opt}(\tau, d)$ the maximum value of this objective function exhibited by any τ-clustering of the uncertain graph $\mathcal{G}$. It is important to remark that $p_{opt}(\tau, d) > 0$ if and only if there exists a tau-clustering of $\mathcal{G}$ such that for each node u there is at least a path of length d between u and the center of its cluster.

Suppose that we run Algorithms 6 and 8 using the d-connection probability rather than the unconstrained connection probability, anywhere it is required, and let p_{alg} denote the value of the above objective function for the clusterings returned by the algorithms. We have the following theorem for the algorithm based on the sequential cluster growing strategy.

Theorem 11. *Considering only d-connection probabilities, Algorithm 6 terminates after at most $\lfloor 2\log_{1+\gamma}(1/p_{opt}(\tau, \lfloor d/2 \rfloor))\rfloor + 1$ iterations of the while loop and returns a clustering with*

$$p_{alg} \geqslant \frac{p^2_{opt}(\tau, \lfloor d/2 \rfloor)}{(1+\gamma)}$$

if $p_{opt}(\tau, \lfloor d/2 \rfloor) > 0$.

The assumption on $p_{opt}(\tau, \lfloor d/2 \rfloor) > 0$ is required in the above statement to ensure that the algorithm terminates. As for the algorithm based on the concurrent cluster growing strategy, we have the following result.

Theorem 12. *Considering only d-connection probabilities, Algorithm 8 returns an ℓ-clustering with $\ell = O\left(\tau \log^2 n\right)$ and such that $p_{alg} \geqslant p^2_{opt}(\tau, \lfloor d/2 \rfloor)$, with high probability.*

The proofs of the two theorems above follow easily, with virtually the same arguments, by substituting $p_{opt}(\tau, \lfloor d/2 \rfloor)$ for $p_{opt}(\tau)$ in the proofs of Theorem 8 and 10, respectively.

As for Algorithm 7, again suppose we run it using the d-connection probability rather than the unconstrained connection probability, anywhere it is required. Define $\tau(\sqrt{q}, d)$ to be the minimum number of clusters that yield a value of the objective function of Equation 4.9 greater than or equal to the threshold q, and let τ_{alg} be the number of clusters returned by the algorithm. Then, by substituting $\tau(\sqrt{q})$ with $\tau(\sqrt{q}, \lfloor d/2 \rfloor)$ in the statement of Theorem 9, we can prove the following theorem with virtually the same arguments.

Theorem 13. *Considering only d-connection probabilities, Algorithm 7 returns a τ-clustering with $\tau_{alg} \leqslant \tau_{opt}(\sqrt{q}, \lfloor d/2 \rfloor)$, where each node is connected to the center of its cluster with probability at least q.*

4.6 Implementing the oracle

So far, the presentation of the algorithms relied on the assumption that a perfect oracle for pairwise connection probabilities exists. Unfortunately the implementation of such a perfect oracle is very hard, as we saw in Section 4.1. Indeed, the estimation of the pairwise connection probability between two nodes $u, v \in V$ is the most critical part for the implementation of our algorithms. In Section 4.2, we saw how to use Monte-Carlo sampling to estimate such probabilities, with bounds on both the probability of error and the entity of the error itself. The problem is that to

get reliable estimates one has to set the number of samples r according to Eq. (4.6), namely $r \geqslant (3 \ln \frac{2}{\delta})/(\varepsilon^2 \Pr[u \sim v])$ which embodies the probability to be estimated. Clearly, since such a probability is not known a priori, we have to resort to suitable guesses. It is important to observe that choosing too few samples would lead to unreliable results, and choosing too many would make the algorithms inefficient.

In this section we present improved Monte-Carlo sampling approaches to address the above issues, and we will see how to embed them in both the algorithms described in the previous section. We assume that a lower bound p_L to $p^2_{opt}(\tau)$ is available. This lower bound can be obtained by squaring the probability of the most unlikely world, but in practice it may represent a minimum connection probability set by the user, which may not be interested in clusterings with very low connection probabilities. In the latter case, if the algorithm does not find a clustering using only probabilities above p_L, it terminates by reporting that no clustering has been found. Remember that $\tilde{q}(u, v)$ denotes the estimate of the probability $\Pr[u \sim v]$ obtained by sampling possible worlds (Section 4.2, Equation (4.5)). Moreover, for a node $u \in V$ and a set of nodes $S \subset V$, we define $\tilde{c}(u, S) = \arg\max_{c \in S}\{\tilde{q}(c, u)\}$ as the function returning the node $\in S$ connected to u with the highest *estimated* probability. Similarly, we define $\tilde{\pi}(u, S) = \Pr[\tilde{c}(u, S) \sim u]$ as the maximum estimated probability of connection between u and any node $\in S$. We use $\varepsilon > 0$ to denote an approximation parameter to be fixed by the user. The basic idea of the sampling approaches presented here is that we adjust the number of samples dynamically during the execution of the algorithms.

4.6.1 Implementation of the sequential growing strategies

Consider Algorithm 6, which given a parameter τ looks for a τ-clustering maximizing the connection probabilities to cluster centers. Recall that the algorithm uses a probability threshold p_{curr} which is lowered at each iteration, and which is essentially a guess on the target value of the objective function. In this section we present an implementation of Algorithm 6 which maintains the same overall structure and uses progressive sampling to estimate connection probabilities. This implementation is shown in Algorithm 9 and works as follows.

In Iteration i, Algorithm 9 seeks only connection probabilities that are $\geqslant p_{curr}$. Therefore, we can limit the number of samples used in Iteration i just to the ones necessary to estimate reliably probabilities greater than p_{curr}. This number of samples is given by the following equation

$$r_i = \frac{12}{p_{curr}\varepsilon^2} \ln\left(\frac{2n^2}{\delta}\left(1 + \left\lfloor \log_{1+\gamma} \frac{1}{p_L} \right\rfloor\right)\right) \tag{4.10}$$

where $\delta \in O\left(n^{-d}\right)$ for some constant $d > 1$. To take into account the error introduced by the estimation, which is bounded by ε, we modify the rule with which nodes are marked as covered by cluster centers as follows: given the value of p_{curr}, the algorithm marks as covered by c_i all nodes v with estimated probability $\tilde{q}(v, c) \geqslant (1 - \frac{\varepsilon}{2})p_{curr}$.

Let $\tilde{p}_{alg}$ be the objective function of the clustering obtained with the implementation of the sequential clustering algorithm augmented with the above sampling strategy. The following theorem ensures the quality of such a clustering.

Algorithm 9: SEQUENTIALCLUSTERIMPL$(\mathcal{G}, \tau, \gamma)$

$p_{curr} \leftarrow 1$;
while $p_{curr} \geqslant p_L$ **do**
  $r \leftarrow \frac{12}{p_{curr}\varepsilon^2} \ln\left(\frac{2n^2}{\delta}\left(1 + \left\lfloor \log_{1+\gamma} \frac{1}{p_L} \right\rfloor\right)\right)$;
  $V_1 \leftarrow V$;
  $S \leftarrow \emptyset$; ▷ `Centers selected by the algorithm`
  Sample r possible worlds $G_1, \ldots G_h, \ldots, G_r \sqsubseteq \mathcal{G}$;
  for $i \leftarrow 1$ **to** τ **do**
    select one arbitrary cluster center $c_i \in V_i$;
    for $u \in V$ **do**
      $\tilde{q}(c_i, u) \leftarrow \frac{1}{r} \sum_{h=1}^{r} \mathbf{I}_{G_h}(u, v)$;
    end
    $M_i \leftarrow \{u \in V_i : \tilde{q}(c_i, u) \geqslant (1 - \frac{\varepsilon}{2})p_{curr}\}$; ▷ `Nodes marked by` c_i
    $V_{i+1} \leftarrow V_i - M_i$;
    $S \leftarrow S \cup \{c_i\}$;
    if $V_{i+1} = \emptyset$ **then**
      for $u \in V$ **do**
        Assign u to the cluster centered in $\tilde{c}(u, S)$;
      end
      return the clusters centered at $c_1, c_2, \ldots, c_i$;
    end
  end
  $p_{curr} \leftarrow p_{curr}/(1 + \gamma)$;
end
Error "No clustering found";

Theorem 14. *Algorithm 9 terminates after at most* $\lfloor 2\log_{1+\gamma}(1/p_{opt}(\tau))\rfloor + 1$ *iterations of the while loop and returns a clustering with*

$$\tilde{p}_{alg} \geqslant \frac{(1-\varepsilon)}{(1+\gamma)} p^2_{opt}(\tau)$$

if $p^2_{opt}(\tau) > p_L$*, with high probability.*

Proof. Consider an arbitrary iteration i of the algorithm, and a pair of nodes $u, v \in V$. Let

$$\delta' = \frac{\delta}{n^2\left(1 + \left\lfloor \log_{1+\gamma} \frac{1}{p_L} \right\rfloor\right)}$$

By the choice of the number of samples (Equation (4.10)) and Inequality (4.7) we have the following properties on the estimate $\tilde{q}(u, v)$

- if $\Pr[u \sim v] > p_{curr}$, then $\tilde{q}(u, v) < (1 - \frac{\varepsilon}{2})$ with probability $< \delta'$
- if $\Pr[u \sim v] < (1-\varepsilon)p_{curr}$, then $\tilde{q}(u, v) \geqslant (1 - \frac{\varepsilon}{2})$ with probability $< \delta'$

Moreover, note that the algorithm performs at most $1 + \lfloor \log_{1+\gamma} \frac{1}{p_L} \rfloor$ iterations of the outer **while** loop. Therefore, by the choice of δ' and by union bound on the

number of node pairs and the number of iterations, we have that the following holds with probability $> 1 - \delta$: in each iteration every node connected with some center with probability $\geqslant p_{curr}$ is added to a cluster, and no cluster contains nodes whose connection probability with the center is $< (1-\varepsilon)p_{curr}$. Note that some node with connection probability to a center $\geqslant (1-\varepsilon)p_{curr}$ may be added to a cluster. This implies that $p_{alg} \geqslant (1-\varepsilon)p_{curr}$, with probability $1-\delta$.

Consider now the ℓ-iteration, with $\ell = \lfloor 2\log_{1+\gamma}(1/p_{opt}(\tau))\rfloor + 1$, in which we have $p_{curr} \leqslant p^2_{opt}(\tau)$ and the algorithm stops returning a clustering. Note that at the beginning of this iteration we have $p_{curr} > p^2_{opt}(\tau)/(1+\gamma)$. Therefore, by the discussion above, we have

$$p_{alg} \geqslant (1-\varepsilon)p_{curr} \geqslant \frac{(1-\varepsilon)}{(1+\gamma)}p^2_{opt}(\tau)$$

with probability $1-\delta$. By the choice of $\delta \in O\left(n^{-d}\right)$ for some $d > 1$, the statement holds with high probability. □

We now move to considering the implementation of Algorithm 7 which, given a threshold probability q, seeks a clustering with connection probabilities $\geqslant q$, while minimizing the number of clusters. Algorithm 10 shows the implementation of this clustering strategy augmented with the use of sampling for estimating connection probability. Note that in this case, the threshold probability is fixed by the user, therefore there is no need to adapt the number of samples dynamically. Therefore, we use a fixed number of samples r given by the following expression.

$$r = \frac{12}{q\varepsilon^2}\ln\frac{2n^2}{\delta} \tag{4.11}$$

with $\delta \in O\left(n^{-d}\right)$ for some constant $d > 1$. In the implementation, a vertex u is added to a cluster C_i of center c_i if $\tilde{q}(c_i, u) \geqslant q(1-\frac{\varepsilon}{2})$, where q is the minimum connection probability threshold given in input, where estimates $\tilde{q}(c_i, u)$ are obtained through Equation (4.5) using the above number of samples.

Theorem 15. *With high probability Algorithm 10 returns a partition of V into $\tau_{alg} \leqslant \tau(\sqrt{q})$ clusters such that each node is connected to the center of its cluster with probability at least $(1-\varepsilon)q$.*

Proof. We first prove that each node is connected to the center of its cluster with probability at least $(1-\varepsilon)q$. By reasoning analogously to the proof of Theorem 14, we have that with probability $1-\delta$ no cluster contains nodes with a connection probability to the center $< (1-\varepsilon)q$. Therefore, we have that clusters contain only nodes with connection probability to the center $\geqslant (1-\varepsilon)q$, with probability $1-\delta$.

As we did in Theorem 9, we consider an optimal $\tau(\sqrt{q})$-clustering, where each node is connected to the center of its cluster with probability at least $\sqrt{q}$. At the end of the i-th iteration of the **while** loop of Algorithm 10, all the nodes u that in the optimal $(\sqrt{\tau})$-clustering belong to the same cluster as one of the c_j's, with $j \leqslant i$, must have already been included in some cluster, since $\Pr\left[c_j \sim v\right] \geqslant q$ by Theorem 7. Hence, no more than $\tau_{opt}(\sqrt{q})$ clusters are returned by the algorithm. □

Algorithm 10: SEQUENTIALCLUSTER2IMPL($\mathcal{G}, q$)

$V_1 \leftarrow V$;
$i \leftarrow 1$;
$r \leftarrow \frac{12}{q\varepsilon^2} \ln \frac{2n^2}{\delta}$;
Sample r possible worlds $G_1, \ldots G_h, \ldots, G_r \sqsubseteq \mathcal{G}$;
while *true* **do**
  select one arbitrary cluster center $c_i \in V_i$;
  for $u \in V$ **do**
    $\tilde{q}(c_i, u) \leftarrow \frac{1}{r} \sum_{h=1}^{r} \mathbf{I}_{G_h}(u, v)$;
  end
  $C_i \leftarrow \{u \in V_i \,:\, \tilde{q}(c_i, u) \geqslant (1 - \frac{\varepsilon}{2})q\}$;
  $V_{i+1} \leftarrow V_i \setminus C_i$;
  if $V_{i+1} = \emptyset$ **then return** $C_1, C_2, \ldots, C_i$;
  $i \leftarrow i + 1$;
end

4.6.2 Implementation of the concurrent growing strategy

In Algorithm 8, the most critical part in a given iteration i is the identification of tfhe $|V_i|/2$ nodes to add to clusters. These are the nodes which are connected to some center with the highest probability. Note that the algorithm does not enforce a limit on the minimum probability that can be used in a given iteration. However, we can still apply a progressive sampling technique.

Algorithm 11 depicts our concurrent cluster growing strategy with progressive sampling to estimate connection probabilities For a given iteration i, let V_i be the set of nodes that are still uncovered, and consider the set of newly selected centers S_i. Moreover, consider a probability threshold $\pi_h = \pi_0/2^h$, for $h \geqslant 0$ with $\pi_0 = 1$. We will use π_h as a bound on the minimum probability that the algorithm will consider when adding a node to a cluster. Similarly to the previous section, we also need a bound p_L on the lowest possible probability that the algorithm may need to estimate. Such a bound, as before, can either be obtained by squaring the probability of the most unlikely world, or can be set by the user. For a given threshold π_h, the number of samples is, for $\varepsilon \in (0, 1)$ and $\delta \in (0, 1)$

$$r_h = \frac{12}{\pi_h \varepsilon^2} \ln \left(\frac{2n^2}{\delta} \left(2 + \left\lfloor \log \frac{1}{p_L} \right\rfloor + \lceil \log n \rceil \right) \right) \tag{4.12}$$

Iteration i is then as follows. Using the above number of samples, starting from r_0, the algorithm will estimate the values of $\pi(u, S_i)$, for each uncovered node $u \in V_i$ (Procedure PROGRESSIVESAMPLING in Algorithm 11). Then if the number of estimates that are larger than $\pi_h \cdot (1 + \frac{\varepsilon}{2})$ is less than $|V_i|/2$, the algorithm repeats the estimation using r_{h+1} samples, with $\pi_{h+1} = \pi_h/2$. This progressive sampling is iterated until there are at least $|V|/2$ nodes such that the estimate of their connection probability to some center is higher than $\pi_h \cdot (1 + \frac{\varepsilon}{2})$. At this point, the algorithm builds clusters as described in Algorithm 8 and moves to iteration $i + 1$. Let $\tilde{p}_{alg}$ be the objective function of the clustering obtained from the concurrent clustering algorithm with embedded progressive sampling.

Algorithm 11: ConcurrentClusterImpl$(\mathcal{G}, \tau)$

$V_1 \leftarrow V$; $i \leftarrow 1$;
while $|V_i| > \alpha\tau \log n$ **do**
  select $\alpha\tau \log n$ new cluster centers at random with replacement from V_i;
  $S_i \leftarrow$ {newly selected centers};
  ▷ `If the call below stops with an error, terminate execution`
  $u_1, u_2, \ldots, u_{|V_i|/2} \leftarrow$ ProgressiveSampling$(\mathcal{G}, V_i, S_i)$;
  foreach $u_1, \ldots u_j \ldots, u_{|V_i|/2}$ **do**
    assign u_j to cluster centered at $\tilde{c}(u_j, S_i)$;
  end
  $V_{i+1} \leftarrow \{u_j : j > |V_i|/2 \wedge u_j \notin S_i\}$;
  $i \leftarrow i + 1$;
end
return the clusters centered at $S_1 \cup S_2 \cup \cdots \cup S_{i-1} \cup V_i$;

Procedure *ProgressiveSampling*$(\mathcal{G}, \bar{V}, S)$
  $h \leftarrow 1$;
  $\pi_h \leftarrow 1$;
  while *true* **do**
    $r_h \leftarrow \frac{12}{\pi_h \varepsilon^2} \ln\left(\frac{2n^2}{\delta}\left(2 + \left\lfloor \log \frac{1}{p_L} \right\rfloor + \lceil \log n \rceil\right)\right)$;
    Sample r_h possible worlds $G_1, \ldots G_j, \ldots, G_r \sqsubseteq \mathcal{G}$;
    for $c \in S, u \in \bar{V}$ **do** $\tilde{q}(c, u) \leftarrow \frac{1}{r} \sum_{j=1}^{r} \mathbf{I}_{G_j}(u, v)$;
    let $u_1, u_2, \ldots, u_{|\bar{V}|}$ be the nodes of V_i in decreasing order of $\tilde{\pi}(\cdot, S_i)$;
    if $\tilde{\pi}(u_{|V_i|/2}) \geqslant \pi_h \cdot (1 + \frac{\varepsilon}{2})$ **then return** $u_1, u_2, \ldots, u_{|\bar{V}|/2}$;
    else if $\pi_h/2 < p_L$ **then Error** "No clustering found" ;
    else
      $\pi_{h+1} \leftarrow \pi_h/2$;
      $h \leftarrow h + 1$;
    end
  end
end

Theorem 16. *Algorithm 11 returns an ℓ-clustering with $\ell = O\left(\tau \log^2 n\right)$ and such that $\tilde{p}_{alg} \geqslant (1 - \varepsilon)p^2_{opt}(\tau)$, if $p^2_{opt}(\tau) > 2p_L$, with high probability.*

Proof. We prove the theorem in two steps: first we will prove that if $p^2_{opt}(\tau) > 2p_L$ then the algorithm will find a clustering, with high probability; then we will show that in each iteration, there is no node v assigned to any cluster centered in c such that $\Pr[c \sim v] < (1 - \varepsilon)p^2_{opt}$, with high probability. Let

$$\delta' = \frac{\delta}{n^2\left(2 + \left\lfloor \log \frac{1}{p_L} \right\rfloor + \lceil \log n \rceil\right)}$$

Assume that $p^2_{opt}(\tau) > 2p_L$. In the ProgressiveSampling subroutine in Algorithm 11, consider iteration h such that $\pi_h \leqslant p^2_{opt}(\tau)$, if a clustering has not been found already. By the choice of the number of samples, for a fixed pair of nodes u, v such that $\Pr[u \sim v] \geqslant p^2_{opt}(\tau)$, we have $\tilde{q}(u, v) \geqslant (1 + \frac{\varepsilon}{2})p^2_{opt}(\tau)$, with probability

$\leqslant \delta'$. By union bound on the number of pairs, we have $\tilde{q}(u,v) \geqslant (1+\frac{\varepsilon}{2})p^2_{opt}(\tau)$ for any $u,v \in V$ such that $\Pr[u \sim v] \geqslant p^2_{opt}(\tau)$, with probability $\leqslant \delta$. By reasoning as in the proof of Theorem 10 we can prove that there are $|V_i|/2$ such nodes, and therefore the algorithm computes a clustering.

Consider now Iteration i of the outer **while** loop of Algorithm 11, and the set of newly selected centers S_i. We are going to prove that, for $v \in V_i$ and $c \in S_i$ such that $\Pr[v \sim c] < (1-\varepsilon)p^2_{opt}(\tau)$, v is added to the cluster centered in c with probability $\leqslant \delta'$. We condition on the event that at least one center of S_i falls into every *large* as defined in Theorem 10, which happens with high probability. Let $w_1, \ldots, w_j, \ldots, w_{|V_i|/2}$ be $|V_i|/2$ nodes of V_i such that $\pi(w_j, S_i) \geqslant p^2_{opt}(\tau)$. Consider an arbitrary iteration h of the inner call to ProgressiveSampling. For v to be added to the cluster centered in c we need that $\tilde{q}(v,c) > \left(1+\frac{\varepsilon}{2}\right)\pi_h$. We have two cases, which we analyze in the following

- $\Pr[v \sim c] < \pi_h$. Note that this implies that

$$\Pr\left[\tilde{q}(v,c) > \left(1+\frac{\varepsilon}{2}\right)\pi_h\right] < \Pr\left[\tilde{q}(v,c) > \left(1+\frac{\varepsilon}{2}\right)\Pr[v \sim c]\right] \leqslant \frac{\delta'}{2}$$

  where the last inequality follows by Chernoff bound.

- $\Pr[v \sim c] \geqslant \pi_h$. For $1 \leqslant j \leqslant |V_i|/2$, let c_{w_j} be the center maximizing $\Pr[w_j \sim c_{w_j}]$. Note that $\Pr[v \sim c] < \Pr[w_j \sim c_{w_j}]$ by hypothesis. Moreover, observe that for v to be added to the cluster centered in c, it must be $\tilde{q}(v,c) > \tilde{q}(w_j, c_{w_j})$, for any $j \in [1, |V_i|/2]$. We bound the probability of this event. Note that $(1-\frac{\varepsilon}{2})\Pr[w_j \sim c_{w_j}] > (1+\frac{\varepsilon}{2})\Pr[v \sim c]$. By Chernoff bound and the choice of the number of samples

$$\Pr\left[\tilde{q}(w_j, c_{w_j}) \leqslant \left(1-\frac{\varepsilon}{2}\right)\Pr[w_j \sim c_{w_j}]\right] \leqslant e^{-\frac{r_h \Pr[w_j \sim c_{w_j}]}{3}\frac{\varepsilon^2}{4}} \leqslant e^{-\frac{r_h \pi_h}{3}\frac{\varepsilon^2}{4}} \leqslant \frac{\delta'}{2}$$

  and similarly

$$\Pr\left[\tilde{q}(v,c) \leqslant \left(1+\frac{\varepsilon}{2}\right)\Pr[v \sim c]\right] \leqslant \frac{\delta'}{2}$$

  Therefore, $\tilde{q}(v,c) > \tilde{q}(w_j, c_{w_j})$ with probability $\leqslant \delta'$, for a fixed j.

By combining the above two cases, and by union bound on the w_j's, we have that $\tilde{q}(v,c) > \left(1+\frac{\varepsilon}{2}\right)\pi_h$ with probability $\leqslant \delta' \cdot n$. Hence, in iteration i of the while loop and iteration h of the inner call of ProgressiveSampling, we have that v is added to the cluster centered in c with probability $\leqslant \delta' \cdot n$. By applying the union bound on all nodes v, and on the $2 + \left\lfloor \log \frac{1}{p_L} \right\rfloor + \lceil \log n \rceil$ worst case iterations of the algorithm, we have that, with probability $\leqslant 1-\delta$, for any cluster center c there is no node v added to the cluster centered in c such that $\Pr[v \sim c] < (1-\varepsilon)p^2_{opt}(\tau)$, and the theorem follows. □

4.7 Experimental evaluation

In this section we report some preliminary experiments, with the goal of comparing the performance of our algorithms with respect to other approaches proposed in

graph	nodes	edges
`Collins`	1622	9074
`Gavin`	1855	7669
`Krogan`	2708	7123
`DBLP`	226413	1432920

Table 4.1: Graphs considered in our experiments.

the literature. We run experiments along three lines, organized as follows. First, we compare the sequential clustering strategy of Algorithm 9 with MCL [Don08], which is often used for comparison in previous works on uncertain graphs[1], as observed in Section 4.1. Then, we consider Protein-Protein Interaction (PPI) networks, which are one of the main motivating applications of the analysis of uncertain graphs. Here, we use Algorithm 9 to predict so-called *protein complexes*, validating the results against the ground truth provided by hand-curated MIPS database of protein complexes [Mew+04]. Finally, we compare the relative performance of our sequential and parallel strategies.

We implemented our algorithms in C++ [Cec16c] with the Monte Carlo sampling of possible worlds executed in parallel using OpenMP. Considering that the sample sizes defined in Section 4.6 are derived to tolerate very conservative union bounds, we verified that in practice starting the progressive sampling schedule from 50 samples always yields very accurate probability estimates. We ran this implementation on a single node of the cluster described in Section 2.2.2. We remark that our implementation of the parallel algorithm does not leverage on all the parallelism exposed by the concurrent growth of multiple clusters, because estimating connection probabilities in parallel is already enough to employ all the parallelism provided by our machine.

In our experiments we employed four different graphs, whose characteristics are summarized in Table 4.1. Three graphs are PPI networks, with different distributions of edge probabilities: `Collins` [Col+07], with mostly high-probability edges; `Gavin` [Gav+06], with the majority of edges having low probabilities, and the CORE network introduced in [Kro+06] (`Krogan` in the following), which has one fourth of the edges with probability greater than 0.9, and the others almost uniformly distributed between 0.27 and 0.9. As a computationally more challenging topology, we also included in the experiments a subgraph of the DBLP collaboration network, obfuscated with the algorithm presented in [Bol+12] (reviewed in Section 4.1), with parameters $k = 20$ and $\varepsilon = 10^{-3}$.

4.7.1 Comparison with MCL

In this section we compare our sequential cluster growing strategy (with progressive sampling, Algorithm 9, dubbed SEQ from now on) with MCL [Don08] in terms of quality of the returned clustering, measured by three metrics. The first metric

[1]We were not able to compare with the algorithms proposed in [Lin+12] and [KPT13], since their code was not made available by the authors.

graph	# clusters	p_{min} MCL	p_{min} SEQ	ACR MCL	ACR SEQ	AVPR MCL	AVPR SEQ	time (ms) MCL	time (ms) SEQ
Collins	263	0.24	0.48	0.50	0.33	0.92	0.85	377	911
	300	0.28	0.53	0.56	0.37	0.93	0.89	231	906
	317	0.44	0.56	0.60	0.39	0.94	0.90	223	629
Gavin	214	0.02	0.18	0.12	0.12	0.74	0.55	525	3400
	320	0.07	0.25	0.23	0.19	0.81	0.62	272	3320
	386	0.07	0.32	0.31	0.24	0.81	0.67	250	833
Krogan	351	0.07	0.31	0.20	0.14	0.71	0.67	1,173	3997
	580	0.16	0.43	0.36	0.23	0.72	0.76	490	4311
	705	0.16	0.57	0.43	0.30	0.71	0.83	417	1761
DBLP	17,774	≈ 0	1.00	0.89	1.0	0.99	1.0	2,002,441	2,109
	32,893	≈ 0	1.00	0.95	1.0	0.99	1.0	958,595	2,083
	42,236	≈ 0	1.00	0.96	1.0	0.99	1.0	891,443	2,102

Table 4.2: Comparison of MCL and SEQ. The second column reports the number of clusters returned by MCL for different inflation values (1.5, 2, and 2.5) and, consequently, the target number of nodes τ for our algorithm. Values of p_{min} reported as ≈ 0 indicate that there was a pair of nodes not connected in any sampled possible world, for 50,000 samples. Running times are in milliseconds.

is the minimum connection probability p_{min} between any node and its cluster's center, that is the metric introduced in Eq. (4.1) and optimized by our algorithms. For MCL, when computing this metric we consider as cluster centers the *attractor nodes* as defined in [Don08]. The other two metrics we consider were introduced in [Lin+12]. The Average Vertex Pairwise Reliability (AVPR), measures the average pairwise connection probability of nodes within the same clusters, and is defined as follows

$$\text{AVPR} = \frac{\sum_{i=1}^{\tau} \sum_{u,v \in C_i} \Pr[u \sim v]}{\sum_{i=1}^{\tau} \binom{|C_i|}{2}}$$

The Average Cluster Reliability (ACR), instead, is the weighted average probability that a cluster is connected in a random possible world

$$\text{ACR} = \frac{\sum_{i=1}^{\tau} |C_i| \cdot \Pr[\text{“}C_i \text{ is connected”}]}{|V|}$$

Recall from Section 4.1 that the granularity of the clustering computed by MCL cannot be controlled accurately but it is influenced by the inflation parameter. Therefore, for each graph in Table 4.1, we run MCL with inflation set to 1.5, 2.0, and 2.5, and then we run SEQ with a value of τ which matches the granularity of the clustering returned by each MCL run. The results of the experiments are summarized in Table 4.2, which also reports the running times.

With respect to the p_{min} metric, SEQ is always better than MCL, which is not surprising given the fact that SEQ optimizes this metric. In particular, in the case of DBLP MCL finds a clustering with an almost-zero p_{min}, meaning that there is at

least one pair of nodes in the same cluster with almost zero connection probability. Conversely, SEQ finds a clustering with $p_{min} = 1$, meaning that all pairs of nodes in the same cluster are connected with probability 1, as supported by Theorem 7.

As for the ACR and AVPR metrics, the scores obtained by the two algorithms are comparable, with MCL performing slightly better in most cases. We remark that p_{min} is a harder worst-case metric, whereas ACR and AVPR are averages that may hide the presence of low-probability connections in the same cluster. Also, these experiments show that for some graphs, such as DBLP, a clustering strategy that ignores the possible-world semantics, as the one at the base of MCL, may provide a very poor clustering with respect to our metric. As for the running times, MCL is almost always faster with the notable exception of the biggest DBLP graph. This is due to the fact that, thanks to progressive sampling, SEQ is able to find the clustering with $p_{min} = 1$ with a small number of samples.

This preliminary comparison of the two algorithms shows that, with respect to the goal of maximizing the minimum connection probability to centers, our approach exhibits encouragingly better performance compared to MCL.

4.7.2 Experiments on Protein-Protein Interaction networks

One of the motivating applications for the study of uncertain graphs is the analysis of Protein-Protein Interaction (PPI) networks. Recall that in these graphs the nodes represent proteins, and edges represent interactions between different proteins. Since the measures used to determine such interactions are affected by noise, a PPI network is modeled as an uncertain graph. Interacting proteins can be grouped in *protein complexes*, that are groups of proteins that stably interact with one another. Since the experiments to detect both protein interactions and protein complexes are expensive, the interest in analyzing PPI networks is to find clusters of proteins to predict unknown protein complexes.

The evaluate the performance of sequential clustering strategy SEQ in this predictive setting, we use the benchmark is the Krogan graph for which the authors published a clustering with 547 clusters obtained with MCL [Kro+06, Suppl. Table 10]. We consider a ground truth derived from the MIPS database [Mew+04], which is publicly available [BB06]. For the purpose of the evaluation, we restrict ourselves to proteins appearing in both Krogan and MIPS, thus obtaining a ground truth of 13,496 protein pairs. The input to the clustering algorithms is the entire Krogan graph. We evaluated the returned clusterings in terms of the confusion matrix. Namely, a pair of proteins in the same cluster is considered a true positive if both proteins appear in the same MIPS complex, and a false positive otherwise; similarly, a pair of proteins in different clusters is a true negative if the two proteins do not appear in the same MIPS complex, and false negative otherwise.

We run SEQ considering only d-connection probabilities (see Section 4.5) for different values of d, and by setting $\tau = 547$ so as to match the cardinality of the reference clustering from [Kro+06]. The idea behind the use of limited path length is that we want to place in the same predicted complex proteins that are connected with a high probability, but which are also as topologically close. We do not report results for $d = 1$ since there is no clustering of the Krogan graph with $d = 1$ and $\tau = 547$. We remark that in [KPT13] the authors applied to the same dataset

	d	TPR	FPR	FNR
	2	0.125	0.005	0.875
	3	0.148	0.011	0.852
SEQ	4	0.168	0.035	0.832
	6	0.642	0.589	0.358
	8	0.769	0.737	0.231
MCL		0.142	0.002	0.858

Table 4.3: Comparison of MCL and SEQ with limited path length on `Krogan` w.r.t. the MIPS ground truth.

a clustering strategy based on a star-cover of the graph which is reminiscent of our algorithm with $d = 1$, but their clustering has more clusters than the reference clustering by [Kro+06]. Table 4.3 reports the True Positive Rate (TPR), False Positive Rate (FPR) and False Negative Rate (FNR) for SEQ with different values of d, and for MCL. We observe that, for low path lengths, our algorithm is able to find a clustering with scores similar to the reference. Note that higher values of d yield fewer false negatives at the expense of an increased number of false positives. This is expected, since with higher d the algorithm may place proteins that are topologically distant in the same predicted complex. Observe that a moderate number of false positives may be a desirable feature, since these pairs can be the target of further investigation to verify unknown protein interactions.

This experiment supports our intuition that considering topologically close proteins while aiming at high connection probabilities makes our algorithm comparable with state of the art solutions in this predictive setting. Moreover, it can be regarded as a preliminary evidence that our algorithm can be applied to the important context of the analysis of Protein-Protein Interaction networks with good results, prompting us to further develop of our approach. In particular, ongoing work is focused on comparing our approaches with the clustering algorithm of [NYP12], which is developed to consider explicitly the semantics of PPI networks. Moreover, a comparison of the relative performance of our sequential and concurrent cluster growing strategies is ongoing.

4.7.3 Comparison of sequential and parallel strategy

In the last set of experiments we compare our sequential and parallel cluster growing strategies described in Algorithms 9 and 11 (SEQ and PAR, respectively). We use `dblp` as a benchmark because its large size makes it computationally challenging for the clustering task, and compare the two algorithms in terms of the number of clusters found, the quality metric p_{min}, and the running time. We ran SEQ to find τ clusters, and PAR with batch size τ, for several values of τ. Actually, the theory developed in Section 4.4 prescribes the use of a larger batch size (i.e.. $\Theta(\tau \log n)$) to ensure that the approximation guarantees hold with high probability. However, we found that in practice a smaller batch size yields clusterings of comparable quality but smaller granularity.

The results of this experiment are reported in Table 4.4. We observe that PAR

	# clusters		p_{min}		time (s)	
τ	PAR	SEQ	PAR	SEQ	PAR	SEQ
32	71	32	0.01	0.01	299	412
64	180	62	0.05	0.02	94	155
128	488	128	0.16	0.05	55	113
256	1031	256	0.15	0.11	74	115
512	1957	512	0.50	0.23	112	143

Table 4.4: Comparison of PAR and SEQ on DBLP. Parameter τ is the batch size for PAR and the target number of clusters for SEQ. Running times are in seconds.

is faster than SEQ, even though the parallelism of the platform available for the experiment is not sufficient to unleash its full potential, due to the parallel cluster growing. Nonetheless, PAR is faster than SEQ since it does not build the clustering from scratch for each guess. PAR pays this better running time with a slight increase in the number of clusters, much lower than the theretical worst-case bound. Also, this slackness allows PAR to find a clustering with a higher p_{min}, hence making it more adaptive to the input. Observe that the values of p_{min} reported here are lower than the ones in Table 4.2 because different clustering granularities are considered in the two cases.

Chapter 5

Diversity maximization

With this chapter we move from the realm of graphs into the domain of more general metric spaces. We will consider the problem of *diversity maximization*, a fundamental primitive in massive data analysis, which provides a succinct summary of a dataset while preserving the diversity of the data [MB08; AMT13; Wu13; Yan+15].

Consider for instance news aggregation websites, which collect a large number of news articles from several sources. Users of these websites are not interested in dealing with all this information at once. On the contrary, the role of a news aggregator is to present its users with a small number of news, covering a *diverse* set of topics. Similarly, consider the case of web search. A particular query might give millions of results, possibly referring to different topics. These results need to be ranked before being presented to the user, which most likely will pay attention only to the first few results. Therefore, the interest is in maximizing the diversity of the top-ranked pages presented to the user. In general, combinations of relevance ranking and diversity maximization have been explored in a variety of applications, including web search [AK11; CG98; Xin+06], e-commerce [BGM11], recommendation systems [YLA09; MRK06], aggregator websites [MZR09] and database query-result navigation [KSI06; CL07]. The common problem in all these applications is that even after filtering and ranking for relevance, the output set is often too large to be presented to the user. A practical solution is to present a diverse subset of the results, so that the user can evaluate the variety of options and possibly refine the search. Other than the aforementioned applications, there are contexts where it is desirable to place a set of facilities in such a way that they are as *dispersed* as possible. One such case is the placing of "undesirable" facilities, like nuclear power plants, ammunition dumps or oil storage tanks: these facilities should be spread out so that an accident happening to one does not damage the others [EN89; Tam91; RRT07]. Similarly, when planning the distribution of business franchises it is desirable to minimize the competition between different units by separating them [Kub87]. All the aforementioned problems, along with others listed in [RRT07; AMT13; Ind+14],

Problem	Diversity measure	Sequential approximation
remote-edge	$\min_{p,q \in S} d(p,q)$	2 (2) [Tam91]
remote-clique	$\sum_{p,q \in S} d(p,q)$	2 (2) [HRT97; BG09]
remote-star	$\min_{c \in S} \sum_{q \in S \setminus \{c\}} d(c,q)$	2 (−) [CH01]
remote-bipartition	$\min_{Q \subset S, \lvert Q \rvert = \lfloor \lvert S \rvert / 2 \rfloor} \sum_{q \in Q, z \in S \setminus Q} d(q,z)$	3 (−) [CH01]
remote-tree	$w(\mathrm{MST}(S))$	4 (2) [Hal+99]
remote-cycle	$w(\mathrm{TSP}(S))$	3 (2) [Hal+99]

Table 5.1: Diversity measures considered in this thesis. $w(\mathrm{MST}(S))$ (resp., $w(\mathrm{TSP}(S))$) denotes the minimum weight of a spanning tree (resp., Hamiltonian cycle) of the complete graph whose nodes are the points of S and whose edge weights are the pairwise distances among the points. The last column lists the best known approximation factor, the lower bound under the hypothesis $P \neq NP$ (in parentheses), and the related references.

can be formulated as diversity maximization problems, whose pervasiveness motivates us in finding efficient algorithms.

There are a number of ways to formulate the goal of finding a set of k points which are as diverse, or as far as possible from each other. We adopt the classification introduced in [CH01], where diversity maximization problems are formulated in terms of specific graph-theoretic measures defined on sets of k points, seen as the nodes of a clique where each edge is weighted with the distance between its endpoints. In this setting, measuring the diversity of a set in terms of the minimum distance between its points corresponds to the *remote-edge* measure; whereas measuring the diversity by the sum of the distances corresponds to the *remote-clique* problem. Several diversity measures defined in this way are reported in Table 5.1. Finding a subset of cardinality k maximizing any of the diversity criteria reported in the table is known to be NP-hard for general metric spaces. While the most appropriate ones in the context of web search, e-commerce, aggregator systems and query results navigation are the remote-edge and the remote-clique measures [GS09; AMT13], our results also extend to the other measures in the table, which have important applications in analyzing network performance, locating strategic facilities or noncompeting franchises, or determining initial solutions for iterative clustering algorithms or heuristics for hard optimization problems such as TSP [Hal+99; CH01; RRT07]. We include all of these measures here to demonstrate the versatility of our approach to a variety of diversity criteria. We want to stress that different measures characterize the diversity of a set in a different fashion: indeed, an optimal solution with respect to one measure is not necessarily optimal with respect to another measure.

In this chapter, we analyze our algorithms in terms of the *doubling dimension* of the input set (see Definition 1), which we will review in Section 5.2. This powerful notion has been used in the analysis of algorithms in a number of recent works [ABS10; RG06; KRX08; GKK14], including our diameter-approximation algorithms presented in Chapter 3. While our methods yield provably tight bounds in spaces of bounded doubling dimension (e.g., any bounded dimension Euclidian

space) they have the potential of providing good approximations in more general spaces based on important practical distance functions such as the cosine distance in web search [AK11] and the dissimilarity (Jaccard) distance in database queries [LRU14], as we shall see in Section 5.8.

We present MapReduce and streaming algorithms for the diversity maximization problem in Sections 5.4 and 5.5. Our algorithms are based on the core-set framework [AHV05], which captures the idea of a small subset preserving a good approximation of the diversity of the input set. In Section 5.3 we will define the characteristics that a subset must exhibit in order to be a core-set for the diversity-maximization problem. In particular, we will use variations of k-center clustering as a means of building such core-sets in an efficient way. The basic idea of our algorithms is to build a clustering with a very fine granularity, based on the amount of available resources, so that the cluster centers provide an accurate representation of the input. This core-set construction, paired with the concept of doubling dimension, will yield efficient and accurate algorithms. In Section 5.6 we will see how to reduce the memory requirements of our algorithms. Finally, we will present an extensive experimental work in Section 5.8.

The results presented in this chapter have been published in the Proceedings of the VLDB Endowment [Cec+17].

5.1 Related work

Diversity maximization for various diversity objective measures has been studied in the literature under different names, including *p-Dispersion*, *Max-Min Facility Dispersion*, and *p-Maximin* [Kub87; RRT94; Erk90; Tam91], *Maxisum Dispersion* and *Average Facility Dispersion* [Kub87; RRT94], *MaxMinSum* and *MaxSumMin dispersion* [EN91], *Remote-MST* and *Remote-TSP* [Hal+99]. A uniform taxonomy for diversity maximization problems has been given in [CH01], and it is the one we adopt. All of these problems are known to be NP-hard, and several sequential approximation algorithms have been proposed in the literature. Table 5.1 lists, for each problem, the approximation factor attained by the best sequential approximation algorithm in general metric spaces. There are also some specialized results for some spaces with bounded doubling dimension: for the remote-clique problem, a polynomial-time $(\sqrt{2}+\varepsilon)$-approximation algorithm on the Euclidean plane, and a polynomial-time $(1+\varepsilon)$-approximation algorithm on d-dimensional spaces with rectilinear distances, for any positive constants $\varepsilon > 0$ and d, are presented in [FM03]. In [Hal+99] it is shown that a natural greedy algorithm attains a 2.309 approximation factor on the Euclidean plane for remote-tree. Recently, the remote-clique problem has been considered under matroid constraints [AMT13; CEZ16], which generalize the cardinality constraints considered in previous literature.

In recent years, the notion of (composable) core-set has been introduced as a key tool for the efficient solution of optimization problems on large datasets. A *core-set* [AHV05], with respect to a given computational objective, is a (small) subset of the entire dataset which contains a good approximation to the optimal solution for the entire dataset. A *composable core-set* [Ind+14] is a collection of core-sets, one for each subset in an arbitrary partition of the dataset, such that the union of these core-sets is a good core-set for the entire dataset. The approximation factor attained

by a (composable) core-set is defined as the ratio between the value of the global optimal solution and the value of the optimal solution when limited to points of the (composable) core-set.

Core-sets have been applied in a wide variety of contexts, including computational geometry [AHV05], clustering [HM04; HK07; Bat+14], and submodular maximization [Bad+14; MZ15]. We remark that the diversity maximization and the submodular maximization problems are fundamentally different, since usually diversity functions are not submodular. In particular, the approaches devised in the aforementioned papers do not apply to the problems we consider. For the diversity maximization problems listed in Table 5.1, composable core-sets with constant approximation factors have been devised in [Ind+14; AFZ15], and we report their approximation guarantees in Table 5.2. Note that once a (composable) core-set is available one has to run an approximation algorithm on such a (composable) core-set to actually find a solution, since the problems are NP-hard, hence exact solutions are out of reach even for smaller input sizes. For this reason, in Table 5.2 we report also the approximation ratio obtained by combining the approximation of introduced by the core-set with the one introduced by the approximation algorithm applied on the core-set. As observed in [Ind+14], (composable) core-sets may become key ingredients for developing efficient algorithms for the MapReduce and streaming frameworks, where the memory available for a processor's local operations is typically much smaller than the overall input size. In particular, the composable core-sets construction of [Ind+14; AFZ15] can be used to yield two-rounds MapReduce algorithms in the following way. In the first round the algorithm partitions the input in ℓ subsets distributed among the reducers, so that each reducer can compute a composable core-set of size k. In the second round the composable core-sets are aggregated in the memory of a single reducer, which runs a sequential approximation algorithm for the problem at hand on this set of $\ell \cdot k$ points. Hence, the implementation of the algorithms of [Ind+14; AFZ15] in the $\mathrm{MR}(M_L, M_A)$ model requires local memory $M_L = \Theta(\ell \cdot k)$, and aggregate memory $M_A = \Theta(n)$. As for the application of composable core-sets in the streaming model, in [Ind+14] the authors propose a one-pass implementation. The stream of n input points is partitioned into $\sqrt{n/k}$ blocks of size $\sqrt{kn}$ each, and a core-set of size k is computed from each block and kept in memory. At the end of the pass, the final solution is computed on the union of the core-sets, whose total size is $\sqrt{kn}$. The algorithm used to build the core-sets in each reducer depends on the problem being solved. While for some problems the simple and efficient GMM algorithm is used, for other problems (namely remote-clique, remote-bipartition, and remote-star) the core-sets are built using the LocalSearch [AMT13] algorithm, which works as follows. Given a set S of n points, an integer k, and a parameter γ, the LocalSearch algorithm keeps track of a tentative solution T of size k, initialized to a subset of the input containing the two farthest points. Then, it iteratively improves the diversity of T by a factor at least $(1 + \gamma/n)$ by suitably swapping a point $\in T$ with a point $\in S \setminus T$, converging to a solution in $O\left(\frac{n}{\gamma} \log k\right)$ iterations. Note that, in each iteration, the LocalSearch algorithm has to compute the diversity of T for $O(kn)$ swaps. Therefore, depending on the diversity measure being considered, the LocalSearch algorithm may have a high complexity. For instance,

	Core-set approximation [Ind+14; AFZ15]	Overall approximation
remote-edge	3	6
remote-clique	$6+\gamma$ †	$12+\gamma$ †
remote-star	12	24
remote-bipartition	18	54
remote-tree	4	16
remote-cycle	3	9

† γ is introduced in the description of the LocalSearch algorithm.

Table 5.2: Approximation factors of the composable core-sets for diversity maximization proposed in the literature [Ind+14; AFZ15]. The second column reports the overall approximation factor for the given diversity problem, obtained by multiplying the core-set approximation factor with the best sequential approximation factor, as reported in Table 5.1.

evaluating the remote-clique diversity of a set of k points takes $\Omega(k)$ time: in this case the complexity of LocalSearch becomes $\Omega\left(\frac{k^2n^2}{\gamma}\log k\right)$.

5.2 Preliminaries

Let S be a set of points from a metric space $\mathcal{M} = (M, \mathrm{dist})$, and recall that the doubling dimension of $\mathcal{M}$ is the smallest D such that any ball of radius r is covered by at most 2^D balls of radius $r/2$ (See definitions in Chapter 2). Note that as an immediate consequence, for any $0 < \varepsilon \leqslant 1$, any ball of radius r can be covered by at most $(1/\varepsilon)^D$ balls of radius εr.

For ease of presentation, we concentrate on metric spaces of constant doubling dimension D, although the results can be immediately extended to nonconstant D by suitably adjusting the ranges of variability of the parameters involved. As we noted in Chapter 2, several relevant metric spaces have constant doubling dimension, a notable case being Euclidean space of constant dimension D, which has doubling dimension $O(D)$ [GKL03].

Let $\mathrm{div} : 2^M \to \mathbb{R}$ be a *diversity function* that maps a set $S \subset M$ to some nonegative real number. We will consider the instantiations of function div listed in Table 5.1, which have been studied under different names in the literature. We adopt the taxonomy that was introduced in [CH01], and which is used also in [Ind+14; AFZ15]. For a specific diversity function div, a set $S \subset M$ of size n and a positive integer $k \leqslant n$, the goal of the *diversity maximization problem* is to find some subset $S' \subseteq S$ of size k that maximizes the value $\mathrm{div}(S')$. In the following, we refer to the k-*diversity* of S as

$$\mathrm{div}_k(S) = \max_{S' \subseteq S, |S'|=k} \mathrm{div}(S')$$

When we reviewed the related work, we introduced the concept of core-set as found in the literature [AHV05]. The idea captured by the core-set concept is that

of a small set of points that approximate some property of a larger set. We now formalize this intuition for the property of the k-diversity of a set.

Definition 7. *Let* $\mathrm{div}(\cdot)$ *be a diversity function,* k *be a positive integer, and* $\beta \geqslant 1$*. A set* $T \subseteq S$*, with* $|T| \geqslant k$*, is a* β-core-set *for* S *with respect to the* k*-diversity if*

$$\mathrm{div}_k(T) \geqslant \frac{1}{\beta}\,\mathrm{div}_k(S)$$

As we reviewed in Section 5.1, the concept of core-set has been extended in [Ind+14; AFZ15] so that, given an arbitrary partition of the input set, the union of the core-sets of each subset in the partition is a core-set for the entire input set. A core-set with this property is deemed a *composable core-set*, for which we give a formal definition below.

Definition 8. *Let* $\mathrm{div}(\cdot)$ *be a diversity function,* k *be a positive integer, and* $\beta \geqslant 1$*. A function* $c(S)$ *that maps* $S \subset M$ *to one of its subsets computes a* β-composable core-set *w.r.t.* div *if, for any collection of disjoint sets* $S_1, \ldots, S_\ell \subset M$ *with* $|S_i| \geqslant k$*, we have*

$$\mathrm{div}_k\left(\bigcup_{i=1}^{\ell} c(S_i)\right) \geqslant \frac{1}{\beta}\,\mathrm{div}_k\left(\bigcup_{i=1}^{\ell} S_i\right)$$

Finally, we will make use of the concepts of *radius* and *farness* of a set as introduced in Chapter 2, and we report here the definitions for convenience. Given a set S and a subset T, the *radius* of T with respect to S is $r_T(S) = \max_{p \in S} \mathrm{dist}(p, T)$. The *farness* of T is the minimum distance between any two points of T, that is $\rho_T = \min_{p \in T} \mathrm{dist}(p, T \setminus \{p\})$. For a given integer $k > 0$, we also define the *optimal radius* $r_k^*(S)$ for S with respect to k as the minimum radius of a subset of k points of S. Similarly, we define the *optimal farness* $\rho_k^*(S)$ for S with respect to k to be the maximum farness of any subset of k points of S.

5.3 Core-set characterization

In this section we identify some properties that, when exhibited by a subset T of the pointset S, guarantee that T is a $(1+\varepsilon)$-core-set for the diversity problems listed in Table 5.1. In the subsequent sections we will show how core-sets with these properties can be obtained in the streaming and MapReduce settings. In fact, when we discuss the MapReduce setting, we will also show that these properties also yield composable core-sets featuring tighter approximation factors than existing ones, for spaces with bounded doubling dimension.

First, we need to establish a fundamental relation between the optimal radius $r_k^*(S)$ and the optimal farness $\rho_k^*(S)$ for the pointset S. In Chapter 2 we considered the GMM greedy algorithm for the k-center problem [Gon85], and we proved in Fact 1, that the set of centers found by the GMM algorithm is an *anticover*: that is, $r_C(S) \leqslant \rho_C$. Thanks to this property, we can prove the following fact.

Fact 2. *Given a set* S *and* $k > 0$*, we have* $r_k^*(S) \leqslant \rho_k^*(S)$.

Proof. Consider the set of centers C found by the GMM algorithm. By the anticover property (Fact 1) we have $r_C(S) \leqslant \rho_C$. Since $r^*_k(S) \leqslant r_C(S)$ and $\rho_C(S) \leqslant \rho^*_k(S)$, the fact immediately follows. □

Consider an arbitrary diversity measure $\mathrm{div}(\cdot)$ from Table 5.1 and let $O \subseteq S$ be the subset of k points such that $\mathrm{div}(O) = \mathrm{div}_k(S)$. Consider a subset $T \subseteq S$. Intuitively, T is a good core-set for some diversity measure on S, if for each point of the optimal solution O it contains a point sufficiently close to it. We formalize this intuition by suitably adapting the notion of *proxy function* introduced in [Ind+14]. We aim at defining a function $p : O \to T$ such that the distance between o and $p(o)$ is bounded, for any $o \in O$. For some problems we will require this function to be injective, whereas for some others, injectivity will not be needed. We begin by studying the remote-edge and the remote-cycle problem.

Lemma 6. *For any given $\varepsilon > 0$, let ε' be such that $(1 - \varepsilon') = 1/(1 + \varepsilon)$. A set $T \subseteq S$ is a $(1 + \varepsilon)$-core-set for the remote-edge and the remote-cycle problems if $|T| \geqslant k$ and there is a function $p : O \to T$ such that, for any $o \in O$, $d(o, p(o)) \leqslant (\varepsilon'/2)\rho^*_k(S)$.*

Proof. Consider the remote-edge problem first, and observe that $\mathrm{div}_k(T) \leqslant \mathrm{div}(O) = \rho^*_k(S)$. By applying the triangle inequality and the stated property of the proxy function p we get

$$\begin{aligned}
\mathrm{div}_k(T) &\geqslant \min_{o_1, o_2 \in O} d(p(o_1), p(o_2)) \\
&\geqslant \min_{o_1, o_2 \in O} \{d(o_1, o_2) - d(o_1, p(o_1)) - d(o_2, p(o_2))\} \\
&\geqslant \min_{o_1, o_2 \in O} d(o_1, o_2) - \varepsilon' \rho^*_k(S) \\
&= \mathrm{div}(O)(1 - \varepsilon') = \mathrm{div}(O)/(1 + \varepsilon)
\end{aligned}$$

Note that $p(\cdot)$ does not need to be injective: in fact, if two points of the optimal solution are mapped into the same proxy, the first inequality trivially holds, its right hand side being zero.

Consider now the remote-cycle problem. Note that $\mathrm{div}_k(T) \leqslant \mathrm{div}(O)$. Let $\bar{\rho} = \mathrm{div}(O)/k$ and observe that $\rho^*_k(S) \leqslant \bar{\rho}$. Let $P = \{p(o) : o \in O\} \subseteq T$ be the image of the proxy function. Following the argument given in [Ind+14; AFZ15], consider $\mathrm{TSP}(P)$, an optimal tour on P. We build a weighted graph G whose vertex set is $O \cup P$ and whose edges are those induced by $\mathrm{TSP}(P)$ plus two copies of edge $(o, p(o))$, for each $o \in O$. The weight of an edge (u, v) is $d(u, v)$. Clearly, the resulting graph G is connected and all its vertices have even degree, therefore it admits an Euler tour T_E of its edges. From T_E we obtain a cycle C of O by shortcutting all nodes that are not in O. By repeated applications of the triangle inequality during shortcutting and the fact that $d(o, p(o)) \leqslant (\varepsilon'/2)\bar{\rho}$, we obtain:

$$\begin{aligned}
w(\mathrm{TSP}(O)) &\leqslant w(C) \leqslant w(T_E) \\
&\leqslant w(\mathrm{TSP}(P)) + k\varepsilon'\bar{\rho} \\
&\leqslant \mathrm{div}_k(T) + \varepsilon' \,\mathrm{div}(O)
\end{aligned}$$

Therefore, $\mathrm{div}(O) \leqslant \mathrm{div}_k(T)/(1 - \varepsilon') = \mathrm{div}_k(T)(1 + \varepsilon)$. □

Note that the proof of the above lemma does not require $p(\cdot)$ to be injective. Instead, injectivity is required for the remote-clique, remote-star, remote-bipartition, and remote-tree problems, which are considered next.

Lemma 7. *For a given $\varepsilon > 0$, let ε' be such that $1 - \varepsilon' = 1/(1+\varepsilon)$. A set $T \subseteq S$ is a $(1+\varepsilon)$-core-set for the remote-clique, remote-star, remote-bipartition, and remote-tree problems if $|T| \geqslant k$ and there is an injective function $p : O \to T$ such that, for any $o \in O$, $d(o, p(o)) \leqslant (\varepsilon'/2)\rho_k^*(S)$.*

Proof. Observe that for each of the four problems it holds that $\mathrm{div}_k(T) \leqslant \mathrm{div}(O)$. Let us consider the remote-clique problem first, and define

$$\bar{\rho} = \mathrm{div}(O)/\binom{k}{2} = \sum_{o_1,o_2 \in O} d(o_1, o_2)/\binom{k}{2}$$

Clearly, $\rho_k^*(S) \leqslant \bar{\rho}$. In fact, a set of k points where the minimum distance between any pair of points is $\rho_k^*(S)$ will have a remote-clique diversity $\geqslant \rho_k^* \cdot \binom{k}{2}$ and $\leqslant \mathrm{div}(O)$, which implies $\rho_k^* \cdot \binom{k}{2} \leqslant \mathrm{div}(O)$. By combining this observation with the triangle inequality we have

$$\begin{aligned}
\mathrm{div}_k(T') &\geqslant \sum_{o_1,o_2 \in O} d(p(o_1), p(o_2)) \\
&\geqslant \sum_{o_1,o_2 \in O} [d(o_1, o_2) - d(o_1, p(o_1)) - d(o_2, p(o_2))] \\
&\geqslant \sum_{o_1,o_2} d(o_1, o_2) - \binom{k}{2}\varepsilon'\bar{\rho} = \mathrm{div}(O)/(1+\varepsilon)
\end{aligned}$$

The injectivity of $p(\cdot)$ is needed in this case for the first inequality above to be true, since k distinct proxies are needed to get a feasible solution.

The argument for the other problems is virtually identical, the only change being in the definition of $\bar{\rho}$, and in the expression of the objective function. For the remote-star problem, we have that $\bar{\rho} = \mathrm{div}(O)/(k-1)$; for remote-bipartition $\bar{\rho} = \mathrm{div}(O)/(k/2)^2$; and for remote-tree $\bar{\rho} = \mathrm{div}(O)/k$. By suitably replacing the definition of $\bar{\rho}$ in the appropriate places, the statement for the remote-star, remote-bipartition, and remote-tree follows by the same argument. □

5.4 Applications to data streams

We now see how to construct core-sets in the streaming model. Recall from Section 2.3 that in the streaming model [HRR98] one processor with a limited-size main memory is available for the computation. The input is provided as a continuous stream of items which is typically too large to fit in main memory, hence it must be processed on the fly within the limited memory budget. Streaming algorithms aim at performing as few passes as possible (ideally just one) over the input.

Recall that in [Ind+14], the authors proposed to use composable core-sets to approximate diversity in the streaming model by partitioning the input stream into blocks of size $\sqrt{kn}$. For each block, then, a core-set of size k is built, and the final solution is computed on the union of all the core sets, whose size is $\sqrt{kn}$.

In this section, we show that substantial savings can be obtained by computing a *single* core-set from the entire stream. In particular, we show how to obtain a space requirement independent of n, through two variants of the 8-approximation *doubling algorithm* for the k-center problem presented in [Cha+04], which we described in Section 2.4.3.

5.4.1 The SMM core-set algorithm

In the following, we describe our first variant of the doubling algorithm, which we call SMM (for Streaming Minimum Maximum). SMM is used to compute core-sets for problems in which the injectivity of the proxy function is not required (i.e. remote-edge and remote-cycle).

Let k, k' be two positive integers, with $k \leqslant k'$. SMM(S, k, k'), which returns a core-set of size $\in [k, k']$, works in phases and maintains in memory a set T of at most $k' + 1$ points. Each Phase i is associated with a distance threshold d_i, and is divided into a *merge step* and an *update step*. Phase 1 starts after an initialization in which the first $k' + 1$ points of the stream are added to T, and d_1 is set equal to $\min_{c \in T} d(c, T \setminus \{c\})$. At the beginning of Phase i, with $i \geqslant 1$, the following invariant holds. Let S_i be the prefix of the stream processed so far. Then:

1. $\forall p \in S_i$, $d(p, T) \leqslant 2d_i$
2. $\forall t_1, t_2 \in T$, with $t_1 \neq t_2$, we have $d(t_1, t_2) \geqslant d_i$

Observe that the invariant holds at the beginning of Phase 1. The merge step operates on a graph $G = (T, E)$ where there is an edge (t_1, t_2) between two points $t_1 \neq t_2 \in T$ if $d(t_1, t_2) \leqslant 2d_i$. In this step, the algorithm seeks a maximal independent set $I \subseteq T$ of G, and sets $T = I$. The update step accepts new points from the stream. Let p be one such new point. If $d(p, T) \leqslant 4d_i$, the algorithm discards p, otherwise it adds p to T. The update step terminates when either the stream ends or the $(k' + 1)$-st point is added to T. At the end of the step, d_{i+1} is set equal to $2d_i$. As shown in [Cha+04], at the end of the update step, the set T and the threshold d_{i+1} satisfy the above invariants for Phase $i + 1$.

To be able to use SMM for computing a core-set for our diversity problems, we have to make sure that the set T returned by the algorithm contains at least k points. However, in the algorithm described above the last phase could end with $|T| < k$. To fix this situation, we modify the algorithm so as to retain in memory, for the duration of each phase, the set M of points that have been removed from T during the merge step performed at the beginning of the phase. Consider the last phase. If at the end of the stream we have $|T| < k$, we can pick $k - |T|$ arbitrary nodes from M and add them to T. Note that we can always do so because $M \cup I = k' + 1 \geqslant k$, where I is the independent set found during the last merge step.

Suppose that the input set S belongs to a metric space with doubling dimension D. We have that the core-set built by the SMM algorithm has the properties to be a good core-set for diversity maximization, as shown in the following lemma.

Lemma 8. *For any $0 < \varepsilon' \leqslant 1$, let $k' = (32/\varepsilon')^D \cdot k$, and let T be the set of points returned by SMM(S, k, k'). Then, given an arbitrary set $X \subseteq S$ with $|X| = k$, there exist a function $p : X \to T$ such that, for any $x \in X$, $d(x, p(x)) \leqslant (\varepsilon'/2)\rho^*_k(S)$.*

Proof. Let $r^*_{k'}(S)$ to be the optimal radius for S w.r.t. k'. Suppose that SMM(S, k, k') performs ℓ phases. It is immediate to see that $r_T(S) \leqslant 4d_\ell$. As was proved in [Cha+04], $4d_\ell \leqslant 8r^*_{k'}(S)$, thus $r_T(S) \leqslant 8r^*_{k'}(S)$. Consider now an optimal clustering of S with k centers and radius $r^*_k(S)$ and, for notational convenience, define $\varepsilon'' = \varepsilon'/32$. From the doubling dimension property, we know that there exist at most k' balls in the space (centered at nodes not necessarily in S) of radius at most $\varepsilon'' r^*_k(S)$ which contain all of the points in S. By choosing one arbitrary center in S for each such ball, we obtain a feasible solution to the k'-center problem for S with radius at most $2\varepsilon'' r^*_k(S)$. Consequently, $r^*_{k'}(S) \leqslant 2\varepsilon'' r^*_k(S)$. Hence, we have that $r_T(S) \leqslant 8r^*_{k'}(S) \leqslant 16\varepsilon'' r^*_k(S)$. By Fact 2, we know that $r^*_k(S) \leqslant \rho^*_k(S)$. Therefore, we have $r_T(S) \leqslant 16\varepsilon'' \rho^*_k(S) = (\varepsilon'/2)\rho^*_k(S)$. Given a set $X \subseteq S$ of size k, the desired proxy function $p(\cdot)$ is the one that maps each point $x \in X$ to the closest point in T. By the discussion above, we have that $d(x, p(x)) \leqslant (\varepsilon'/2)\rho^*_k(S)$. □

5.4.2 The SMM-EXT core-set algorithm

For the diversity problems mentioned in Lemma 7 — namely remote-cycle, remote-star, remote-bipartition, and remote-tree — we need that for each point of an optimal solution the final core-set extracted from the data stream contains a *distinct* point very close to it. In what follows, we describe a variant of SMM, dubbed SMM-EXT, which ensures this property. Algorithm SMM-EXT proceeds as SMM but maintains for each $t \in T$ a set E_t of at most k points (referred to as *delegates* in what follows) which are close to t, and include t itself. More precisely, at the beginning of the algorithm, T is initialized with the first $k'+1$ points of the stream, as before, and E_t is set equal to $\{t\}$, for each $t \in T$. In the merge step of Phase i, with $i \geqslant 1$, iteratively for each point t_1 not included in the independent set I, we determine an arbitrary point $t_2 \in I$ such that $d(t_1, t_2) \leqslant 2d_i$ and let E_{t_2} inherit $\max\{|E_{t_1}|, k-|E_{t_2}|\}$ points of E_{t_1}. Note that one such point t_2 must exist, otherwise I would not be a maximal independent set. Also, note that a point $t_2 \in I$ may inherit points from sets associated with different points not in I. As for the update step of Phase i, let p be a new point from the stream, and let $t \in T$ be the point currently in T which is closest to p. If $d(p, t) > 4d_i$ we add it to T setting $E_p = \{p\}$. If instead $d(p, t) \leqslant 4d_i$ and $|E_t| < k$, then we add p to E_t, otherwise we discard it. Finally, we define $T' = \bigcup_{t \in T} E_t$ to be the output of the algorithm, and observe that $T \subseteq T'$.

Lemma 9. *For any $0 < \varepsilon' \leqslant 1$, let $k' = (64/\varepsilon')^D \cdot k$, and let T' be the set of points returned by SMM-EXT(S, k, k'). Then, $k \leqslant |T'| \leqslant k \cdot k'$ and given an arbitrary set $X \subseteq S$ with $|X| = k$, there exist an injective function $p : X \to T'$ such that, for any $x \in X$, $d(x, p(x)) \leqslant (\varepsilon'/2)\rho^*_k(S)$.*

Proof. The upper bound on $|T|$ is immediate, while the lower bound follows from the injectivity proved below. Let $r_{T'}(S) = \max_{p \in S} d(p, T')$ be the radius of T', and suppose that SMM(S, k, k') performs ℓ phases. By defining $\varepsilon'' = \varepsilon'/64$, and by reasoning as in the proof of Lemma 8 we can show that $r_{T'}(S) \leqslant 4d_\ell \leqslant 16\varepsilon'' \rho^*_k(S)$. Consider a point $x \in X$. If $x \in T'$ then we define $p(x) = x$. Otherwise, suppose that x is discarded during Phase j, for some j, because either in the merging or in the update step the set E_t that was supposed to host it had already k points. Let T_i denote the set T at the end of Phase i, for any $i \geqslant 1$. A simple inductive argument

shows that at the end of each Phase i, with $j \leqslant i \leqslant \ell$ there is a point $t \in T_i$ such that $|E_t| = k$ and $d(x,t) \leqslant 4d_i$. In particular, there exists a point $t \in T_\ell$ such that $|E_t| = k$ and $d(x,t) \leqslant 4d_\ell \leqslant 16\varepsilon''\rho^*_k(S)$. Since $E_t \subset T'$, any point in E_t is at distance at most $4d_\ell \leqslant 16\varepsilon''\rho^*_k(S)$ from t, and $|X| = k$, we can select a proxy $p(x)$ for x from the k points in E_t such that $d(x,p(x)) \leqslant 32\varepsilon''\rho^*_k(S) = (\varepsilon'/2)\rho^*_k(S)$ and $p(x)$ is not a proxy for any other point of X. □

It is easy to see that the set T characterized in Lemma 8 satisfies the hypotheses of Lemma 6. Similarly, the set T' of Lemma 9 satisfies the hypotheses of Lemma 7. Therefore, as a consequence of these lemmas, for metric spaces with bounded doubling dimension D, we have that SMM and SMM-EXT compute $(1+\varepsilon)$-core-sets for the problems listed in Table 5.1, as stated by the following two theorems.

Theorem 17. *For any $0 < \varepsilon \leqslant 1$, let ε' be such that $(1-\varepsilon') = 1/(1+\varepsilon)$, and let $k' = (32/\varepsilon')^D \cdot k$. Algorithm* SMM$(S,k,k')$ *computes a $(1+\varepsilon)$-core-set of size k' for the remote-edge and remote-cycle problems using $O\left((1/\varepsilon)^D k\right)$ memory.*

Theorem 18. *For any $0 < \varepsilon \leqslant 1$, let ε' be such that $(1-\varepsilon') = 1/(1+\varepsilon)$, and let $k' = (64/\varepsilon')^D \cdot k$. Algorithm* SMM-EXT$(S,k,k')$ *computes a $(1+\varepsilon)$-core-set of size $k \cdot k'$ for the remote-clique, remote-star, remote-bipartition, and remote-tree problems using $O\left((1/\varepsilon)^D k^2\right)$ memory.*

Note that, for a fixed approximation factor $(1+\varepsilon)$, the size of the core-sets built by SMM and SMM-EXT is different, with SMM-EXT building bigger core-sets. We remark that this is because of the injectivity of the proxy function required by Lemma 7 for the diversity problems addressed by SMM-EXT.

5.4.3 Streaming approximation algorithm

The core-sets discussed above can be immediately applied to yield the following streaming algorithm for diversity maximization. Let S be the input stream of n points. One pass on the data is performed using SMM, or SMM-EXT, depending on the problem, to compute a core-set in main memory. At the end of the pass, a sequential approximation algorithm is run on the core-set to compute the final solution. The following theorem is immediate.

Theorem 19. *Let S be a stream of n points of a metric space of doubling dimension D, and let A be a linear-space sequential approximation algorithm for any one of the problems of Table 5.1, returning a solution $S' \subseteq S$, with $\mathrm{div}_k(S) \leqslant \alpha\,\mathrm{div}(S')$, for some constant $\alpha \geqslant 1$. Then, for any $0 < \varepsilon \leqslant 1$, there is a 1-pass streaming algorithm for the same problem yielding an approximation factor of $\alpha + \varepsilon$, with memory*

- $\Theta\left((\alpha/\varepsilon)^D k\right)$ *for the remote-edge and the remote-cycle problems;*
- $\Theta\left((\alpha/\varepsilon)^D k^2\right)$ *for the remote-clique, the remote-star, the remote-bipartition, and the remote-tree problems.*

Proof. The proof follows by running A on $(1+\varepsilon/\alpha)$-core-sets obtained through Theorems 17 and 18, which have size $\Theta\left((\alpha/\varepsilon)^D k\right)$ and $\Theta\left((\alpha/\varepsilon)^D k^2\right)$, respectively. □

In section 5.7 we will compare the guarantees of the above theorem with the results of previous works.

5.5 Applications to MapReduce

In this section, we consider another popular computational model for processing large amounts of data: MapReduce, which we reviewed in detail in Section 2.2, and which was also the computational model we used for our diameter-approximation algorithm (Chapter 3). We employ composable core-sets in order to leverage the strengths of MapReduce. The basic idea of our approach is the following. We partition the input among the reducers so that, in a first MapReduce round, each reducer can build a composable core-set locally. Then, in a second round, all the composable core-sets are collected in a single reducer, which applies a sequential approximation algorithm for the diversity problem at hand.

Consider a set S belonging to a metric space of doubling dimension D, and a partition of S into ℓ disjoints sets $S_1, S_2, \ldots, S_\ell$. In what follows we consider in turn all the diversity problems listed in Table 5.1: $\mathrm{div}(\cdot)$ denotes the diversity function of the problem under consideration, and O denotes an optimal solution to the problem with respect to instance $S = \cup_{i=1}^{\ell} S_i$. Note that, for any $i \in [1, \ell]$, $\rho_k^*(S_i) \leqslant \rho_k^*(S)$. As we mentioned above, the basic idea of our MapReduce algorithms is to derive, for each S_i, a core-set $T_i \subseteq S_i$. Then, the core-sets are aggregated into one single core-set $T = \cup_{i=1}^{\ell} T_i$, which will be used as input for a sequential approximation algorithm. We want the set T to satisfy the hypotheses of Lemmas 6 and 7, so as to have a guaranteed good approximation.

5.5.1 Using GMM for computing core-sets

Let us consider first the remote-edge and remote-cycle problems. To build our composable core-sets for these problems we use the GMM k-center algorithm, that we described in Section 2.4.1. Note that we are interested only in the set of cluster centers found by GMM, and not in building the associated clustering. The following Lemma shows that if we run Algorithm GMM on each S_i, with $1 \leqslant i \leqslant \ell$, and then take the union of the outputs, the resulting set satisfies the hypotheses of Lemma 6.

Lemma 10. *For any $0 < \varepsilon' \leqslant 1$, let $k' = (8/\varepsilon')^D \cdot k$, and let $T = \bigcup_{i=1}^{\ell} \mathrm{GMM}(S_i, k')$. Then, $|T| = O(\ell k/(\varepsilon')^D)$, and given an arbitrary set $X \subseteq S$ with $|X| = k$, there exist a function $p : X \to T$ such that for any $x \in X$, $d(x, p(x)) \leqslant (\varepsilon'/2)\rho_k^*(S)$.*

Proof. Fix an arbitrary index i, with $1 \leqslant i \leqslant \ell$, and let $T_i = \{c_1, c_2, \ldots, c_{k'}\}$, where c_j denotes the point added to T_i at the j-th iteration of $\mathrm{GMM}(S_i, k')$. Let also $T_i(k) = \{c_1, c_2, \ldots, c_k\}$ and $d_k = d(c_k, T_i(k) \setminus \{c_k\})$. From the anticover property exhibited by GMM, which holds for any prefix of points selected by the algorithm, we have $r_{T_i(k)}(S_i) \leqslant d_k \leqslant \rho_{T_i(k)}(S_i) \leqslant \rho_k^*(S)$. Define $\varepsilon'' = \varepsilon'/8$. Since S_i can be covered with k balls of radius at most d_k, and the space has doubling dimension D, then there exist k' balls in the space (centered at nodes not necessarily in S_i) of radius at most $\varepsilon'' d_k$ that contain all the points in S_i. By choosing one arbitrary center in S_i in each such ball, we obtain a feasible solution to the k'-center problem for S_i with radius at most $2\varepsilon'' d_k$, which implies that the cost of the optimal solution to k'-center is at most $2\varepsilon' d_k$. As a consequence, since by Proposition 1 $\mathrm{GMM}(S_k, k')$ returns a 2-approximation T_i to the k'-center problem for S_i, we have tat $r_{T_i}(S_i) \leqslant 4\varepsilon'' d_k$, hence $r_{T_i}(S_i) \leqslant 4\varepsilon'' \rho_k^*(S)$. Let now $T = \bigcup_{i=1}^{\ell} T_i$. Clearly, we

Algorithm 12: GMM-EXT(S, k, k')

$T' \leftarrow \text{GMM}(S, k')$
Let $T' = \{c_1, c_2, \ldots, c_{k'}\}$
$T \leftarrow \emptyset$
for $j \leftarrow 1$ **to** k' **do**
  $C_j \leftarrow \{p \in S : c_j = \arg\min_{c \in T'} d(c, p) \wedge p \notin C_h \text{ with } h < j\}$
  $E_j \leftarrow \{c_j\} \cup \{\text{ arbitrary } \min\{|C_j| - 1, k - 1\} \text{ points in } C_j\}$
  $T \leftarrow T \cup E_j$
end
return T

have that $r_T(S) \leqslant \max_{1 \leqslant i \leqslant \ell} r_{T_i}(S_i) \leqslant 4\varepsilon'' \rho^*_k(S)$. Hence, for any set $X \subseteq S$, the desired proxy function $p(\cdot)$ is obtained by mapping each $x \in X$ to the closest point in T. By the above argument, we have $d(x, p(x)) \leqslant 4\varepsilon'' \rho^*_k(S) = (\varepsilon'/2)\rho^*_k(S)$. □

5.5.2 The GMM-EXT core-set algorithm

For the diversity problems considered in Lemma 7 (remote-cycle, remote-star, remote-bipartition, and remote-tree) the proxy function is required to be injective. Therefore, similarly to what we did with the streaming algorithms, we develop an extension of the GMM algorithm, dubbed GMM-EXT (see Algorithm 12). This extended algorithm first determines a kernel T' of $k' \geqslant k$ points by running $\text{GMM}(S, k')$ and then augments T' by first determining the clustering of S whose centers are the points of T' and then picking from each cluster its center and up to $k - 1$ delegate points. In this fashion, we ensure that each point of an optimal solution to the diversity problem under consideration will have a distinct close "proxy" in the returned set T.

As before, let $S_1, S_2, \ldots, S_\ell$ be disjoint subsets of a metric space of doubling dimension D. We have:

Lemma 11. *For any $0 < \varepsilon' \leqslant 1$, let $k' = (16/\varepsilon')^d \cdot k$, and let $T = \bigcup_{i=1}^{\ell} \text{GMM-EXT}(S_i, k, k')$. Then, $|T| = O(\ell k^2/(\varepsilon')^D)$ given an arbitrary set $X \subseteq S$, with $|X| = k$, there exist an injective function $p : X \to T$ such that for any $x \in X$, $d(x, p(x)) \leqslant (\varepsilon'/2)\rho^*_k(S)$.*

Proof. For any $1 \leqslant i \leqslant \ell$, let $T_i = \text{GMM-EXT}(S_i, k, k')$ be the result of the invocation of GMM-EXT on S_i. By defining $\varepsilon'' = \varepsilon'/16$ and by reasoning as in Lemma 10, we have that the radius of the set T'_i computed by the call to $\text{GMM}(S_i, k')$ within $\text{GMM-EXT}(S_i, k, k')$ is $r_{T'_i}(S_i) \leqslant 4\varepsilon'' \rho^*_k(S)$. Fix an arbitrary index i, with $1 \leqslant i \leqslant \ell$, and consider, for $1 \leqslant j \leqslant k'$, the sets $C_{i,j}$ and $E_{i,j}$ as determined by Algorithm $\text{GMM-EXT}(S_i, k, k')$, and define $X_{i,j} = X \cap C_{i,j}$. Since $|X_{i,j}| \leqslant \min\{k, |C_{i,j}|\} = |E_{i,j}|$, we can associate each point in $x \in X_{i,j}$ to a distinct proxy $p(x) \in E_{i,j}$. Since both x and $p(x)$ belong to $C_{i,j}$, by the triangle inequality we have that $d(x, p(x)) \leqslant 2r_{T'}(S) \leqslant 8\varepsilon'' \rho^*_k(S) = (\varepsilon'/2)\rho^*_k$. Since the input sets $S_1, S_2, \ldots, S_\ell$ are disjoint, then we have that all the $X_{i,j}$ are disjoint. This ensures that we can find a distinct proxy for each point of X in $T = \bigcup_{i=1}^{\ell} T_i$, hence, the proxy function is injective. □

The two lemmas above guarantee that the set of points obtained by invoking

GMM or GMM-EXT on the partitioned input complies with the hypotheses of Lemmas 6 and 7 of Section 5.3. Therefore, for metric spaces with bounded doubling dimension D, we have that GMM and GMM-EXT compute $(1+\varepsilon)$-composable core-sets for the problems listed in Table 5.1, as stated by the following two theorems.

Theorem 20. *For any $0 < \varepsilon \leqslant 1$, let ε' be such that $(1-\varepsilon') = 1/(1+\varepsilon)$, and let $k' = (8/\varepsilon')^D \cdot k$. For any $X \subseteq S$, GMM(X, k') computes a $(1+\varepsilon)$-composable core-set of size $O(k/(\varepsilon')^D)$ for the remote-edge and remote-cycle problems.*

Theorem 21. *For any $0 < \varepsilon \leqslant 1$, let ε' be such that $(1-\varepsilon') = 1/(1+\varepsilon)$, and let $k' = (16/\varepsilon')^D \cdot k$. For any $X \subseteq S$, GMM-EXT(X, k, k') computes a $(1+\varepsilon)$-composable core-set of size $O\left(k^2/(\varepsilon')^D\right)$ for the remote-clique, remote-star, remote-bipartition, and remote-tree problems.*

As for the streaming algorithms, we have that our MapReduce algorithms have different core-set sizes depending on the diversity problem: for a fixed approximation factor $(1+\varepsilon)$, the core-set for remote-edge and remote-cycle is smaller than the other four problems. This is due to the fact that remote-clique, remote-star, remote-bipartition and remote-tree all require an injective proxy function, which we provide by augmenting the core-set with delegates.

5.5.3 MapReduce deterministic algorithm

The composable core-sets discussed above can be used to obtain the following MapReduce algorithm for diversity maximization. Let S be the input set of n points and consider an arbitrary partition of S into ℓ subsets $S_1, S_2, \ldots, S_\ell$, each of size n/ℓ. In the first round, each S_i is assigned to a distinct reducer, which computes the corresponding core-set T_i, according to algorithms GMM, or GMM-EXT, depending on the problem. In the second round, the union of the ℓ core-sets $T = \bigcup_{i=1}^{\ell} T_i$ is concentrated within the same reducer, which runs a sequential approximation algorithm on T to compute the final solution. We have:

Theorem 22. *Let S be a set of n points of a metric space of doubling dimension D, and let $\mathcal{A}$ be a linear-space sequential approximation algorithm for any one of the problems of Table 5.1, returning a solution $S' \subseteq S$, with $\mathrm{div}_k(S) \leqslant \alpha\,\mathrm{div}(S')$, for some constant $\alpha \geqslant 1$. Then, for any $0 < \varepsilon \leqslant 1$, there is a 2-round MR algorithm for the same problem yielding an approximation factor of $\alpha + \varepsilon$, with $M_A = n$ and*

- $M_L = \Theta\left(\sqrt{(\alpha/\varepsilon)^D kn}\right)$ *for the remote-edge and the remote-cycle problems;*
- $M_L = \Theta\left(k\sqrt{(\alpha/\varepsilon)^D n}\right)$ *for the remote-tree, the remote-clique, the remote-star, and the remote-bipartition problems.*

Proof. Set ε' such that $1/(1-\varepsilon') = 1 + \varepsilon/\alpha$, and recall the remote-edge and the remote-cycle problems admit composable core-sets of size $k' = (8/\varepsilon')^D k$, while the problems remote-tree, remote-clique, remote-star, and remote-bipartition have core-sets of size kk', with $k' = (16/\varepsilon')^D k$. Suppose that the above MR algorithm is run with $\ell = \sqrt{n/k'}$ for the former group of two problems, and $\ell = \sqrt{n/(kk')}$ for the latter group of four problems. Observe that by the choice of ℓ we have that

both the size of each S_i and the size of the aggregate set $|T|$ are $O(M_L)$, therefore the stipulated bounds on the local memory of the reducers are met. The bound on the approximation factor of the resulting algorithm follows from the fact that the Theorems 20 and 21 imply that, for all problems, $\mathrm{div}_k(S) \leqslant (1+\varepsilon/\alpha)\,\mathrm{div}_k(T)$ and the properties of algorithm A yield $\mathrm{div}_k(T) \leqslant \alpha\,\mathrm{div}(S)$. □

Theorem 22 implies that on spaces of constant doubling dimension, we can get approximations to remote-edge and remote-cycle in 2 rounds of MapReduce which are almost as good as the best sequential approximations, with polynomially sublinear local memory $M_L = O\left(\sqrt{kn}\right)$, for values of k up to $n^{1-\delta}$, while for the remaining four problems, with polynomially sublinear local memory $M_L = O\left(k\sqrt{n}\right)$ for values of $k = O\left(n^{1/2-\delta}\right)$, for $0 \leqslant \delta < 1$. In fact, for these four latter problems and the same range of values for k, we can obtain substantial memory savings either by using randomization (in two rounds, as shown in the next subsection), or, deterministically with an extra round (as will be shown in Section 5.6.2).

5.5.4 MapReduce randomized algorithm

In this section, we show how to use randomization to get substantial memory savings for the diversity problems that require an injective proxy function. The basic idea is that if we randomly permute the input before computing the core-sets, it is unlikely that many points of the optimal solution are concentrated in the same reducer. Therefore we can afford to select fewer delegate points to ensure the injectivity of the proxy function, with high probability. The random permutation can be performed with an additional round. We have:

Theorem 23. *For the problems of remote-clique, remote-star, remote-bipartition, and remote-tree, we can obtain a randomized 3-round MR algorithm with the same approximation guarantees stated in Theorem 22 holding with high probability, and with*

$$M_L = \begin{cases} \Theta\left(\sqrt{(\alpha/\varepsilon)^D kn \log n}\right) \text{ for } k = O\left((\varepsilon^D n \log n)^{1/3}\right) \\ \Theta\left((\alpha/\varepsilon)^D k^2\right) \text{ for } k = \begin{cases} \Omega\left((\varepsilon^D n \log n)^{1/3}\right) \\ O\left(n^{1/2-\delta}\right) \ \forall \delta \in [0, 1/6) \end{cases} \end{cases}$$

where α is the approximation guarantee given by the current best sequential algorithms referenced in Table 5.1.

Proof. We fix ε' and k' as in the proof of Theorem 22, and, at the beginning of the first round, we use random keys to partition the n points of S among

$$\ell = \Theta\left(\min\{\sqrt{n/(k' \log n)}, n/(kk')\}\right)$$

reducers. Fix any of the four problems under consideration and let O be a given optimal solution. A simple balls-into-bins argument suffices to show that, with high probability, none of the ℓ partitions may contain more than $\Theta\left(\max\{\log n, k/\ell\}\right)$

out of the k points of O. Therefore, it is sufficient that, within each subset of the partition, GMM-EXT selects up to those many delegate points per cluster (rather than $k-1$). This suffices to establish the new space bounds. □

5.5.5 Recursive MapReduce algorithm

The deterministic strategy underlying the 2-round MR algorithm can be employed recursively to yield an algorithm with a larger (yet constant) number of rounds for the case of smaller local memory budgets. Specifically, if the union of the composable core-sets computed in each subset of the partition has size larger than M_L, we recursively call the algorithm using this union as input. The following theorem shows that this recursive strategy can still guarantee an approximation comparable to the sequential one as long as the local memory M_L is not too small.

Theorem 24. *Let S be a set of n points of a metric space of doubling dimension D, let and $\mathcal{A}$ be a linear-space sequential approximation algorithm for any one of the problems of Table 5.1, returning a solution $S' \subseteq S$, with $\mathrm{div}_k(S) \leqslant \alpha\,\mathrm{div}(S')$, for some constant $\alpha \geqslant 1$. Then, for any $0 < \varepsilon \leqslant 1$ and $0 < \gamma \leqslant 1/3$ there is an $O((1-\gamma)/\gamma)$-round MR algorithm for the same problem yielding an approximation factor of $\alpha + \varepsilon$, with $M_A = n$ and*

- $M_L = \Theta\left((\alpha 2^{(1-\gamma)/\gamma}/\varepsilon)^D k n^\gamma\right)$ *for the remote-edge and the remote-cycle problems;*
- $M_L = \Theta\left((\alpha 2^{(1-\gamma)/\gamma}\varepsilon)^D k^2 n^\gamma\right)$, *for some $\gamma > 0$ for the remote-clique, the remote-star, the remote-bipartition, and the remote-tree problems.*

Proof. Let ε' be such that $1/(1-\varepsilon') = 1 + \varepsilon/(\alpha(2^{(1-\gamma)/\gamma} - 1))$ and recall that the remote-edge and the remote-cycle problems admit composable core-sets of size $k' = (8/\varepsilon')^D k$, while the problems remote-tree, remote-clique, remote-star, and remote-bipartition, have core-sets of size kk', with $k' = (16/\varepsilon')^D$. We may apply the following recursive strategy. We partition the input set S into n/M_L sets of size M_L and compute the corresponding core-sets. Let T be the union of these core-sets. If $|T| > M_L$, then we recursively apply the same strategy using T as the new input set, otherwise, we send T to a single reducer where algorithm A is applied. By the choice of the parameters, it follows that in all cases $(1-\gamma)/\gamma$ rounds suffice to shrink the input set to a size at most M_L. The resulting approximation factor with respect to $\mathrm{div}_k(S)$ will then be at most

$$\alpha\left(1 + \frac{\varepsilon}{\alpha(2^{(1-\gamma)/\gamma} - 1)}\right)^{\frac{(1-\gamma)}{\gamma}} \leqslant \alpha\left(1 + \frac{\varepsilon(2^{(1-\gamma)/\gamma} - 1)}{\alpha(2^{(1-\gamma)/\gamma} - 1)}\right) = \alpha + \varepsilon,$$

where the last inequality follows from the known fact $(1+a)^b \leqslant (1 + (2^b - 1)a)$ for every $a \in [0, 1]$ and $b > 1$, and the observation that, by the choice of γ, we have $(1-\gamma)/\gamma \geqslant 2$. □

5.6 Saving memory: generalized core-sets

Consider the problems remote-clique, remote-star, remote-bipartition, and remote-tree. Our core-sets for these problems are obtained by exploiting the sufficient

conditions stated in Lemma 7, which require the existence of an injective proxy function that maps the points of an optimal solution into close points of the core-set. To ensure this property, our strategy so far has been to add more points to the core-sets, both in streaming and in MapReduce. More precisely, we saw how to build a core-set composed by a kernel of k' points, augmented by selecting, for each kernel point, a number of up to $k-1$ delegate points laying within a small range. This augmentation ensures that for each point o of an optimal solution O, there exists a distinct close proxy among the delegates of the kernel point closest to o, as required by Lemma 7.

In order to reduce the core-set size, hence saving memory, the augmentation can be done implicitly by keeping track only of the number of delegates that must be added for each kernel point. A set of pairs (p, m_p) is then returned, where p is a kernel point and m_p is the number of delegates for p (including p itself). The intuition behind this approach is the following. The set of pairs described above can be viewed as a compact representation of a multiset, where each point p of the kernel appears with multiplicity m_p. If, for a given diversity measure, we can solve a natural generalization of the maximization problem on the multiset, then we can transform the obtained multiset solution into a feasible solution for S by selecting, for each multiple occurrence of a kernel point, a distinct close enough point in S. In what follows we illustrate this idea in more detail.

Let S be a set of points. A *generalized core-set* T for S is a set of pairs (p, m_p) with $p \in S$ and m_p a positive integer, referred to as the *multiplicity* of p, where the first components of the pairs are all distinct. We define its *size* $s(T)$ to be the number of pairs it contains, and its *expanded size* as $m(T) = \sum_{(p,m_p)\in T} m_p$. Moreover, we define the *expansion* of a generalized core-set T as the multiset $\mathcal{T}$ formed by including, for each pair $(p, m_p) \in T$, m_p replicas of p in $\mathcal{T}$.

Given two generalized core-sets T_1 and T_2, we say that T_1 is a *coherent subset* of T_2, and write $T_1 \sqsubseteq T_2$, if for every pair $(p, m_p) \in T_1$ there exists a pair $(p, m'_p) \in T_2$ with $m'_p \geqslant m_p$. For a given diversity function div and a generalized core-set T for S, we define the *generalized diversity* of T, denoted by $\text{gen-div}(T)$, to be the value of div when applied to its expansion $\mathcal{T}$, where m_p replicas of the same point p are viewed as m_p distinct points at distance 0 from one another. We also define the *generalized* k*-diversity* of T as

$$\text{gen-div}_k(T) = \max_{T' \sqsubseteq T : m(T') = k} \text{gen-div}(T').$$

Let T be a generalized core-set for a set of points S. A set $I(T) \subseteq S$ with $|I(T)| = m(T)$ is referred to as a δ*-instantiation* of T if for each pair $(p, m_p) \in T$ it contains m_p distinct delegate points (including p), each at distance at most δ from p, with the requirement that the sets of delegates associated with any two pairs in T are disjoint. The following lemma ensures that the difference between the generalized diversity of T and the diversity of any of its δ-instantiations is bounded.

Lemma 12. *Let T be a generalized core-set for S with $m(T) = k$, and consider the remote-clique, remote-star, remote-bipartition, and remote-tree problems. For any δ-instantiation $I(T)$ of T we have that*

$$\mathrm{div}(I(T)) \geqslant \text{gen-div}(T) - f(k)2\delta.$$

where $f(k) = \binom{k}{2}$ *for remote-clique,* $f(k) = k - 1$ *for remote-star and remote tree, and* $f(k) = \lfloor k/2 \rfloor \cdot \lceil k/2 \rceil$ *for remote-bipartition.*

Proof. Recall that $\text{gen-div}(T)$ is defined over the expansion $\mathcal{T}$ of T where each pair $(p, m_p) \in T$ is represented by m_p occurrences of p. We create a 1-1 correspondence between $\mathcal{T}$ and $I(T)$ by mapping each occurrence of a point $p \in \mathcal{T}$ into a distinct proxy chosen among the delegates for (p, m_p) in $I(T)$. The lemma follows by noting both $\text{gen-div}(T)$ and $\text{div}(I(T))$ are expressed in terms of sums of $f(k)$ distances and that, by the triangle inequality, for any two points p_1, p_2 in the multiset (possibly two occurrences of the same point p) the distance of the corresponding proxies is at least $d(p_1, p_2) - 2\delta$. □

For any one of the four problems under considerations, given a generalized core-set T we intend to run an adaptation of the best sequential algorithm for the problem to compute a coherent subset $\hat{T}$ of T of exapanded size $m(\hat{T}) = k$ whose generalized diversity if a good approximation of $\text{gen-div}_k(T)$. A delta-instantiation of $\hat{T}$ will then provide the final solution. It is important to observe that the best sequential approximation algorithms for the remote-clique, remote-star, remote-bipartition, and remote-tree problems (see Table 5.1), which are essentially based on either finding a maximal matching or running GMM on the input set [HRT97; CH01; Hal+99], can be easily adapted to work on generalized core-sets.

Consider first the adaptation of the GMM algorithm, and its adaptation to a generalized core-set T. The algorithm keeps track of a set of pairs $\hat{T}$, which is initialized with a pair $(p, 1)$, where p is a point of T with some multiplicity m_p. Correspondingly, the pair (p, m_p) in T will be replaced by the pair $(p, m_p - 1)$ Then, the algorithm performs a sequence of iterations until the expanded size of $\hat{T}$ is k. In each iteration, the algorithm seeks among the pairs $(p, m_p) \in T$ such that $m_p > 0$ the one maximizing the distance $\text{dist}(p, c)$, for any $(c, m_c) \in \hat{T}$. Then, the pair $(p, m_p) \in T$ is replaced by a pair $(p, m_p - 1)$, and p is inserted into $\hat{T}$. This insertion operation is implemented as follows: if there is no pair (p, m_p) in $\hat{T}$, then we add to $\hat{T}$ the pair $(p, 1)$, otherwise we replace the original pair with $(p, m_p + 1)$.

For other diversity problems, the approximation algorithm is based on a maximal matching heuristic: the algorithm selects iteratively the pair of unselected points u, v maximizing $\text{dist}(u, v)$, until $\lfloor k/2 \rfloor$ pairs are selected. If k is odd, an extra arbitrary point is added to the solution. This heuristic can be easily applied to a generalized core-set T as follows. In each iteration, the algorithm seeks $(u, m_u) \in T$ and $(v, m_v) \in T$ with $m_u > 0$ and $m_u > 0$ such that $\text{dist}(u, v)$ is maximized. Then, the points u and v are inserted in the solution using the insertion procedure described above. If k is odd, an arbitrary point p such that $(p, m_p) \in T$ with $m_p > 0$ is added to the solution.

Given the above discussion, we have:

Fact 3. *The best existing sequential approximation algorithms for the remote-clique, remote-star, remote-bipartition, and remote-tree, can be adapted to obtain from a given generalized core-set* T *a coherent subset* $\hat{T}$ *with expanded size* $m(\hat{T}) = k$ *and* $\text{gen-div}(\hat{T}) \geqslant (1/\alpha)\,\text{gen-div}_k(T)$*, where* α *is the same approximation ratio achieved on the original problems. The adaptation works in space* $O(s(T))$.

We will now consider how to build generalized core-sets in both streaming and MapReduce.

5.6.1 Generalized core-sets for streaming algorithms

We can modify the streaming algorithm to compute generalized core-sets, so as to lower the memory requirements for the remote-tree, remote-clique, remote-star, and remote-bipartition problems to match the one of the other two problems, at the expense of an extra pass on the data. The idea behind this adaptation is that in a first pass on the data we can build a generalized core-set, using multiplicities instead of explicit delegates. Then, with a sequential approximation algorithm adapted as specified Fact 3 we identify a subset of points in the generalized core-set with expanded size k. Since the adapted sequential algorithms return a set of pairs (p, m_p), where p is a point and m_p is its multiplicity, for the algorithm to return a set of distinct points we need another pass on the data. In this second pass, we use these points with multiplicities to suitably select k distinct points that will make up the final solution. The details of this strategy are provided in the following theorem.

Theorem 25. *For the problems of remote-clique, remote star, remote-bipartition, and remote-tree, we can obtain a 2-pass streaming algorithm with approximation factor $\alpha + \varepsilon$ and memory $\Theta\left((\alpha^2/\varepsilon)^D k\right)$, for any $0 < \varepsilon < 1$, where α is the approximation guarantee given by the current best sequential algorithms referenced in Table 5.1.*

Proof. Let $\bar{\varepsilon}$ be such that $\alpha + \varepsilon = \alpha/(1-\bar{\varepsilon})$, and observe that $\bar{\varepsilon} = \Theta(\varepsilon/\alpha)$. In the first pass we determine a generalized core-set T of size $k' = (64\alpha/\bar{\varepsilon})^D \cdot k$ by suitably adapting the SMM-EXT algorithm to maintain counts rather than delegates for each kernel point. Let r_T denote the maximum distance of a point of S from the closest point x such that (x, m_x) is in T. Using the argument in the proof of Lemma 8, setting $\varepsilon' = \bar{\varepsilon}/(2\alpha)$, it is easily shown that $r_T \leqslant (\varepsilon'/2)\rho_k^*(S) = (\bar{\varepsilon}/(4\alpha))\rho_k^*(S)$. Therefore, we can establish an injective map $p(\cdot)$ from O to the expansion $\mathcal{T}$ of T. Let us focus on the remote-clique problem (the argument for the other three problems is virtually identical), and define $\bar{\rho} = \mathrm{div}(O)/\binom{k}{2}$. By reasoning as in the proof of Lemma 7, we can show that $\text{gen-div}_k(T) \geqslant \mathrm{div}(O)(1 - \bar{\varepsilon}/(2\alpha))$.

At the end of the first pass, the best sequential algorithm for the problem, adapted as stated in Fact 3, is used to compute in memory a coherent subset $\hat{T} \sqsubseteq T$ with $m(\hat{T}) = k$ and such that $\text{gen-div}(\hat{T}) \geqslant (1/\alpha)\,\mathrm{div}(O)(1 - \bar{\varepsilon}/(2\alpha))$.

The second pass computes an r_T-instantiation $I(\hat{T})$ of $\hat{T}$, that is a set containing, for each pair $(p, m_p) \in \hat{T}$, m_p distinct delegates at distance at most r_T from p. The algorithm works by maintaining $\hat{T}$ in memory, together with the set H of delegates. For each delegate in H, the algorithm keeps track of the pair (p, m_p) to which it was assigned. The set of delegates H is initialized to the set of points $\{p : \forall (p, m_p) \in \hat{T}\}$, assigning each p to its corresponding pair. We define a pair $(p, m_p) \in \hat{T}$ (and by extension the point p) to be *complete* if it has been assigned m_p delegates, and *incomplete* otherwise. The algorithm, for each point q of the stream, works as follows. If there is an incomplete pair for which q can be a delegate, then we add q to H, assigning it to that pair. Otherwise, we build the following auxiliary directed graph: there is a node v_p for each pair $(p, m_p) \in \hat{T}$, and there is a directed edge

from v_{p_1} to v_{p_2} if there is a delegate $\in H$ assigned to p_1 that can also be a delegate for p_2. Furthermore, there is a node v_q for q with a directed edge towards each node v_p such that q can be a delegate for p. We run on this graph a DFS starting from v_q, until we either find a node v_p such that (p, m_p) is incomplete or there are no more reachable nodes. In the former case, we can add q to H, with the caveat that we have to "make room" for it by performing a chain of swaps. Let $v_q, v_{p_1}, v_{p_2}, \ldots, v_{p_\gamma}$ be the directed path found by the DFS in the auxiliary graph from v_q to the first node v_{p_γ} such that p_γ is incomplete, and let $q_i \in H$ be a delegate assigned to p_i that can also be a delegate for p_{i+1}, for $1 \leqslant i < \gamma$ (we know that such a point exists because of how the auxiliary graph is defined). We assign q_i to p_{i+1}, for $1 \leqslant i \leqslant \gamma$. Note that we can assign $q_{\gamma-1}$ to p_γ because p_γ is an incomplete node. After this chain of swaps p_1 becomes incomplete and we can assign q to it. If the DFS does not find any v_p such that p is incomplete, then we discard q and proceed to the next point of the stream. Let $\bar{C}$ be the set of points $p \neq q$ such that v_p is traversed by the DFS rooted at v_q, and let $\bar{H} \subseteq H$ be the set of delegates associated to points of $\bar{C}$. There is no way to assign the delegates in $\bar{H} \cup \{q\}$ to the pairs satisfying the distance constraint and such that every pair (p, m_p) is assigned $\leqslant m_p$ delegates. Hence, in any r_T instantiation of $\hat{T}$ that contains q, at least one point of $\bar{H}$ is missing. The algorithm terminates when $|H| = k$.

Note that discarding points of the stream according to the above protocol does not prevent to find a solution, that is a r_T-instantiation of $\hat{T}$. Consider an iteration of the streaming algorithm, and let $\bar{S}$ be the set of points of the stream that have yet to be seen and H be the set of delegates selected so far. By induction on the number of iterations executed by the algorithm, we prove that there is a solution in $H \cup \bar{S}$. It is easy to see that in the first iteration, at the beginning of the stream with $H = \{p : \forall (p, m_p) \in \hat{T}\}$, the base case of the induction holds. Consider now an iteration in which a point q is discarded. Let $\bar{C}$ be the set of points traversed by the DFS as defined above, and let $\bar{H} \subseteq H$ be the set of delegates associated to points in $\bar{C}$. Note that all the points in $\bar{C}$ are complete, using points in $\bar{H}$, otherwise the algorithm would have included q in the solution. Assume by contradiction that all the possible solutions in $H \cup \bar{S}$ contain q, and consider one such solution, deemed Γ. Note that, as we discussed above, at least one point of H must be excluded from the solution to make room for q. Consider now the set $\Psi \subseteq \Gamma$ of delegates assigned to points in $\bar{C}$ in the solution, and observe that $q \in \Psi$. We have that the set $\Gamma \setminus \Psi \cup \bar{H}$ is also a feasible solution: $\bar{H}$ is a set of points that makes all points in $\bar{C}$ complete. Therefore, we can build a solution that does not contain q, contradicting the assumption that all solutions must include q. This ensures that the algorithm finds a feasible solution, that is, a valid r_T-instantiation of $\hat{T}$.

The above streaming algorithm can run using $O(k)$ space. In fact, the sets $\hat{T}$ and H only require $O(k)$ space, and there is no need to explicitly instantiate all the $O(k^2)$ edges of the auxiliary graph, since the algorithm can derive them on demand from the definition.

Now that we have a r_T-instantiation of $\hat{T}$, by applying Lemma 12 with $\delta = r_T \leqslant (\bar{\varepsilon}/(4\alpha))\bar{\rho}$, we get $\mathrm{div}(I(\hat{T})) \geqslant \mathrm{div}(O)/(\alpha + \varepsilon)$ and the approximation factor follows. Since $\bar{\varepsilon} = \Theta(\varepsilon/\alpha)$ and the second pass requires only $O(k)$ space, the space required is $\Theta\left((\alpha/\bar{\varepsilon})^D k\right) = \Theta\left((\alpha^2/\varepsilon)^D k\right)$. □

5.6.2 Generalized core-sets for MapReduce algorithms

In this section, we study how to adapt the MapReduce diversity maximization algorithm to employ generalized core-sets, for the problems of remote-clique, remote-star, remote-bipartition, and remote-tree. As in the case of the streaming algorithm, the aim of this adaptation to is save memory, bringing the memory requirement of the MapReduce algorithm for these four problems on par with the algorithm for remote-edge and remote-cycle. Similarly to what happened with the streaming algorithm, where the adoption of generalized core-sets implies an extra pass on the data, in MapReduce using generalized core-sets costs an extra round.

Before describing the adaptation of the algorithms, we need to extend the definition of composable core-set to generalized core-sets. Let div be a diversity function, k be a positive integer, and $\beta \geqslant 1$. A function $c(S)$ that maps a set of points S to a generalized core-set T for S computes a β-*composable generalized core-set* for div if, for any collection of disjoint sets $S_1, \ldots, S_\ell$, we have that

$$\text{gen-div}_k\left(\bigcup_{i=1}^{\ell} c(S_i)\right) \geqslant \frac{1}{\beta}\,\mathrm{div}_k\left(\bigcup_{i=1}^{\ell} S_i\right).$$

Consider a simple variant of GMM-EXT, which we refer to as GMM-GEN, which on input S, k and k' returns a generalized core-set T of S of size $s(T) = k'$ and extended size $m(T) \leqslant kk'$ as follows: for each point c_i of the kernel set $T' = \text{GMM}(S, k')$, algorithm GMM-GEN returns a pair (c_i, m_{c_i}) where m_{c_i} is equal to the size of the set E_i computed in the i-th iteration of the for loop of GMM-EXT.

Lemma 13. *For any $\varepsilon' > 0$, define $k' = (16\alpha/\varepsilon')^D k$. Algorithm* GMM-GEN *computes a β-composable generalized core-set for the remote-clique, remote-star, remote-bipartition, and remote-tree problems, with $1/\beta = 1 - \varepsilon'/(2\alpha)$.*

Proof. Given a collection of disjoint sets $S_1, \ldots, S_\ell$, let $T_i = \text{GMM-GEN}(S_i, k, k')$, and $T = \bigcup_{i=1}^{\ell} T_i$. Consider the expansion $\mathcal{T}$ of T. Let us focus on the remote-clique problem (the argument for the other three problems is virtually identical) and define $\bar{\rho} = \mathrm{div}(O)/\binom{k}{2}$. By reasoning along the lines of the proof of Theorem 25, we can establish an injective map $p : O \to \mathcal{T}$ such that, for any $o \in O$, $d(o, p(o)) \leqslant (\varepsilon'/(4\alpha))\bar{\rho}$. Let $\hat{T}$ be the generalized core-set whose expansion into a multiset yields the k points of the image of p. We have:

$$\text{gen-div}_k(T) \geqslant \text{gen-div}(\hat{T}) \geqslant \mathrm{div}(O)\left(1 - \frac{\varepsilon'}{2\alpha}\right) \qquad \square$$

We are now able to show that GMM-GEN computes a high-quality β-composable generalized core-set, which can then be employed in a 3-round MR algorithm to approximate the solution to the four problems under consideration with lower memory requirements. The basic idea of this MapReduce algorithm is the following. In the first two rounds, the algorithm computes a generalized core-set using the GMM-GEN algorithm. Then, we find a set with expanded size k using a sequential approximation algorithm for the diversity maximization problem at hand, adapted as described in Fact 3. A third round is needed to build the output, which is a suitable set of distinct points based on the multiplicities of the ones found by the sequential algorithm. This high level idea is detailed in the theorem below.

Theorem 26. *For the problems of remote-clique, remote-star, remote-bipartition, and remote-tree, we can obtain a 3-round MR algorithm with approximation factor $\alpha + \varepsilon$ and $M_L = \Theta\left(\sqrt{(\alpha^2/\varepsilon)^D kn}\right)$, for any $0 < \varepsilon < 1$, where α is the approximation guarantee given by the current best sequential algorithms referenced in Table 5.1.*

Proof. Consider the remote-clique problem (the argument for the other three problems is virtually identical) and define $\bar{\rho} = \mathrm{div}(O)/\binom{k}{2}$. Let ε' be such that $\alpha + \varepsilon = \alpha/(1-\varepsilon')$ and observe that $\varepsilon' = \Theta(\varepsilon/\alpha)$. Also, set $k' = (16\alpha/\varepsilon')^D \cdot k$. For $\ell = \sqrt{n/k'}$ consider a arbitrary partition of the input set S into ℓ subsets $S_1, S_2, \ldots, S_\ell$ each of size $M_L = n/\ell = \sqrt{nk'}$ each. In the first round, each reducer applies GMM-GEN to a distinct subset S_i to compute generalized core-sets of size k'. In the second round, these generalized core-sets are aggregated in a single generalized core-set T, whose size is $\ell k' = \sqrt{nk'} = M_L$ and such that the maximum distance of a point of S from the closest point x with $(x, m_x) \in T$ is $r_T \leqslant (\varepsilon'/(4\alpha))\bar{\rho}$. Then, one reducer applies to T the best sequential algorithm for the problem, adapted as stated in Fact 3, to compute a coherent subset $\hat{T} \sqsubseteq T$ with $m(\hat{T}) = k$ and such that

$$\text{gen-div}(\hat{T}) \geqslant \frac{1}{\alpha}\,\text{gen-div}_k(T) \geqslant \left(1 - \frac{\varepsilon'}{2\alpha}\right)\frac{1}{\alpha}\,\mathrm{div}(O),$$

where the last inequality follows by Lemma 13. In the third round, $\hat{T}$ is distributed to ℓ reducers which are able to compute an instantiation $I(\hat{T})$ of $\hat{T}$ as follows. For each pair $(p, m_p) \in \hat{T}$, such that $p \in S_i$, the i-th reducer selects m_p distinct delegates from S_i at distance at most $r_T \leqslant (\varepsilon'/(4\alpha))\bar{\rho}$ from p. By Lemma 12, we have that

$$\begin{aligned}
\mathrm{div}(I(\hat{T})) &\geqslant \left(1 - \frac{\varepsilon'}{2\alpha}\right)\frac{1}{\alpha}\,\mathrm{div}(O) - \frac{\varepsilon'}{2\alpha}\,\mathrm{div}(O) \\
&= \frac{1}{\alpha}\left(1 - \frac{\varepsilon'}{2\alpha} - \frac{\varepsilon'}{2}\right)\mathrm{div}(O) \\
&\geqslant \frac{1}{\alpha}(1-\varepsilon')\,\mathrm{div}(O) = \frac{1}{\alpha+\varepsilon}\,\mathrm{div}(O)
\end{aligned}$$

As for the memory bound, we have that $M_L = \sqrt{nk'} = \Theta\left(\sqrt{(\alpha^2/\varepsilon)^D kn}\right)$. □

5.7 Comparison with previous approaches

In the previous two sections we introduced streaming and MapReduce algorithms for all six diversity measures listed in Table 5.1. In this section, we compare their approximation factors and memory requirements with the ones that can be attained using the composable core-sets recently proposed in the literature [Ind+14; AFZ15].

For what concerns the streaming model, it is important to observe that while we devised ad-hoc streaming algorithms for the computation of our core-sets, to the best of our knowledge no other core-sets for diversity maximization computable by streaming algorithms are known. Hence, for comparison purposes we will consider the streaming approach proposed in [Ind+14] to derive core-sets from composable core-sets, which is applicable to the composable core-sets presented in the aforementioned previous works. Specifically, a stream of n input points is

	1 pass	2 passes
r-edge r-cycle	$\Theta\left((\alpha/\varepsilon)^D k\right)$	–
r-clique r-star r-bipartition r-tree	$\Theta\left((\alpha/\varepsilon)^D k^2\right)$	$\Theta\left((\alpha^2/\varepsilon)^D k\right)$

Table 5.3: Memory requirements of our streaming approximation algorithms. The approximation factor of each algorithm is reported in Table 5.5. The factor α is the best sequential approximation factor for a given problem, as reported in Table 5.1.

	2 rounds det.	3 rounds randomized	3 rounds det.
r-edge r-cycle	$\Theta\left(\sqrt{(\alpha/\varepsilon)^D kn}\right)$	–	–
r-clique r-star r-bipartition r-tree	$\Theta\left(k\sqrt{(\alpha/\varepsilon)^D n}\right)$	$\max\left\{\Theta\left((\alpha/\varepsilon)^D k^2\right), \Theta\left(\sqrt{(\alpha/\varepsilon)^D kn \log n}\right)\right\}$	$\Theta\left(\sqrt{(\alpha^2/\varepsilon)^D kn}\right)$

Table 5.4: Memory requirements of our MapReduce approximation algorithms. We report only the size of the local memory M_L since the aggregate memory M_A is always linear in n. The approximation factor of each algorithm is reported in Table 5.5. The factor α is the best sequential approximation factor for a given problem (see Table 5.1).

partitioned into blocks of size $\sqrt{kn}$ each. For each of these $\sqrt{n/k}$ blocks, a core-set of size k is computed and kept in memory. Then, the final solution is computed on the union of the core-sets, whose total size is $\sqrt{kn}$.

As for MapReduce, the application of the composable core-sets proposed in [Ind+14; AFZ15] is similar to ours, which we introduced in the previous section. In particular, the input set is partitioned among the reducers, that independently compute core-sets. These core-sets are then aggregated into a single reducer, which computes the final approximation.

In Tables 5.3 and 5.4 we report the memory requirements of our algorithms. These should be compared with the memory requirements of the approached proposed in [Ind+14; AFZ15]: $\Theta\left(\sqrt{kn}\right)$ for streaming, and $\Theta(\ell k)$ for MapReduce, where ℓ is the number of subsets in which the input is partitioned. Observe that in streaming (Table 5.3), the memory required by our algorithms is independent from the size of the stream n. This allows our algorithms to process streams of unlimited size. On the contrary, previous approaches require to maintain in main memory several core-sets: the consequent dependence on n of the memory required makes them applicable only to streams of size bounded by the available resources. To the best of our knowledge, our algorithms are the first to use core-sets for diversity-maximization of size independent of n in the streaming setting. As for MapReduce (Table 5.4) all our algorithms can adapt to the available local memory M_L thanks to

	Previous [Ind+14; AFZ15] General metric spaces	Our algorithms Bounded doubling dimension
remote-edge	6	$2+\varepsilon$
remote-clique	$12+\gamma$ †	$2+\varepsilon$
remote-star	24	$2+\varepsilon$
remote-bipartition	54	$3+\varepsilon$
remote-tree	16	$4+\varepsilon$
remote-cycle	9	$3+\varepsilon$

† γ is introduced in the description of the LocalSearch algorithm.

Table 5.5: Approximation factors obtained by our algorithmsg with suitable memory (the memory requirements are summarized in Tables 5.3 and 5.4). We compare with the best approximation obtained with the approaches devised in [Ind+14; AFZ15]. The approximation factors reported here assume the application of the best sequential algorithm for each problem, as reported in Table 5.1. D is the doubling dimension of the metric space of the input points.

the parameter ε, while previous approaches may not be able to exploit the available local memory. Our adaptability to the available memory, in both the streaming and MapReduce algorithms, has consequences also on the quality of the approximation.

In Table 5.5 we report the approximation factors obtained by our algorithms on spaces of bounded doubling dimension, with sufficient memory. The approximation factors are obtained by considering the application of the best-known sequential algorithm (see Table 5.1) for each problem on the final core-set built by our algorithms. Observe that, with respect to the previous approaches proposed in [Ind+14; AFZ15], on metric spaces with bounded doubling dimension we can get considerably better approximation factors. Most importantly, our approximation factors can be made as close as possible to the best factors attainable by sequential algorithms, as the amount of local memory increases. This means that by giving more resources to our algorithms, we are able to obtain better approximation qualities, compared to the constant-factor approximations of previous works.

5.8 Experimental evaluation

We present a suite of experiments with the aim of assessing the performance of our algorithms on different datasets, both real-world and synthetic. The main interest is in verifying the dependence of the approximation quality on the resources available to the algorithms. We also evaluate the performance and scalability of our implementations, and we compare with the performance of previous approaches [Ind+14; AFZ15]. To the best of our knowledge, ours is the first work on diversity maximization in the MapReduce and streaming settings which complements theoretical findings with an experimental evaluation.

We ran our experiments on the infrastructure described in Section 2.2.2. The MapReduce algorithm has been implemented within the Spark framework, whereas the streaming algorithm has been implemented in Scala, simulating a streaming

setting [Cec16a].

Synthetic datasets are generated randomly from the three-dimensional Euclidean space. We tested several random distributions of points, including: points picked uniformly at random within the unit-radius sphere; points with gaussian distance from the origin and a uniform random direction (in terms of the line going through the point and the origin); points placed uniformly at random in the space, each surrounded by a cloud of points at random gaussian distance and in a uniformly random direction. Among all these distribution, we verified that the most challenging for our algorithms was the following one. For a given k, k points are randomly picked on the surface of the unit radius sphere centered at the origin of the space. This way we ensure the existence of a set of far-away points. The other points are chosen uniformly at random in the concentric sphere of radius 0.8. Since this is the most challenging, and ultimately more interesting, input for our algorithms, we omit the results relative to other inputs.

We also tested our algorithms on real-world workloads, using the `musiXmatch` dataset [Ber+11]. This dataset contains the lyrics of 237,662 songs, each represented by its *bag-of-words* [LRU14]. In the bag-of-words model, the set of all the words occurring in the documents being processed defines a vector space, with each word being a dimension. A text, then, is represented in this space as the sparse vector of its word counts. In the case of the `musiXmatch` dataset, the set of words is limited to the most frequent 5,000 across the entire dataset, so the dimensionality of the corresponding vector space is 5,000. We performed the following preprocessing on the dataset. We filtered out songs represented by less than 10 frequent words, obtaining a dataset of 234,363 songs. The reason of this filtering is that one can build an optimal solution using songs with short, non overlapping word lists. Thus, removing these songs makes the dataset more challenging for our algorithm. As for the distance function, in the bag-of-words model, a commonly used one is the *cosine distance*. Given two vectors $\vec{u}$ and $\vec{v}$, it is defined as follows

$$\mathrm{dist}(\vec{u}, \vec{v}) = \frac{\arccos\left(\frac{\vec{u}\cdot\vec{v}}{\|\vec{u}\|\|\vec{v}\|}\right)}{\pi/2}$$

Since the components of all the vectors are always non-negative, the normalization factor $\pi/2$ makes the function take values $\in [0, 1]$. With this distance, two vectors orthogonal vectors — that is, two texts with no words in common — are at distance 1. This distance is closely related to the *cosine similarity* commonly used in Information Retrieval [LRU14].

Since optimal solutions are out of reach for the input sizes that we considered, for each dataset we computed approximation ratios with respect to the best solution found by many runs of our MapReduce algorithm with maximum parallelism and large local memory. In the case of the `musiXmatch` dataset, note that whenever we have a solution where the minimum cosine distance between two vectors is 1 such a solution is clearly optimal given the range of values taken by the cosine distance function.

In this section, we report results for the remote-edge problem. Preliminary experiments suggest that for other problems — such as remote-clique, remote-tree, and remote-star — our algorithms exhibit similar behaviour. A complete

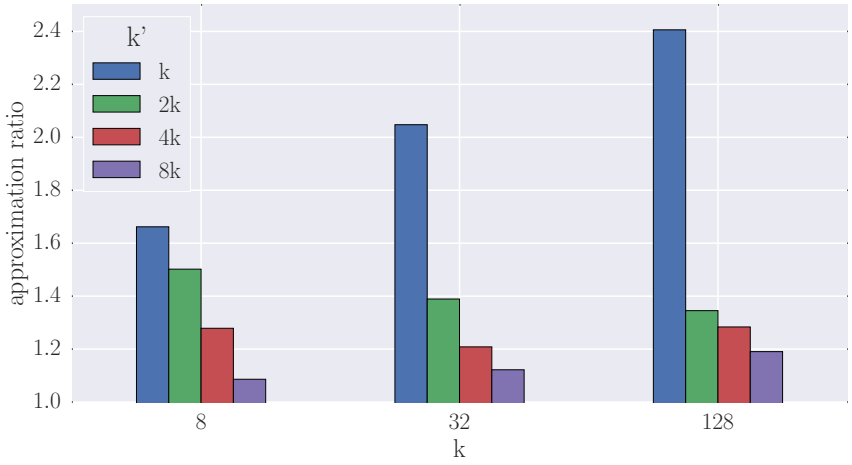

Figure 5.1: Approximation ratio attained by the streaming algorithm for different values of k and k′ on the `musiXmatch` dataset.

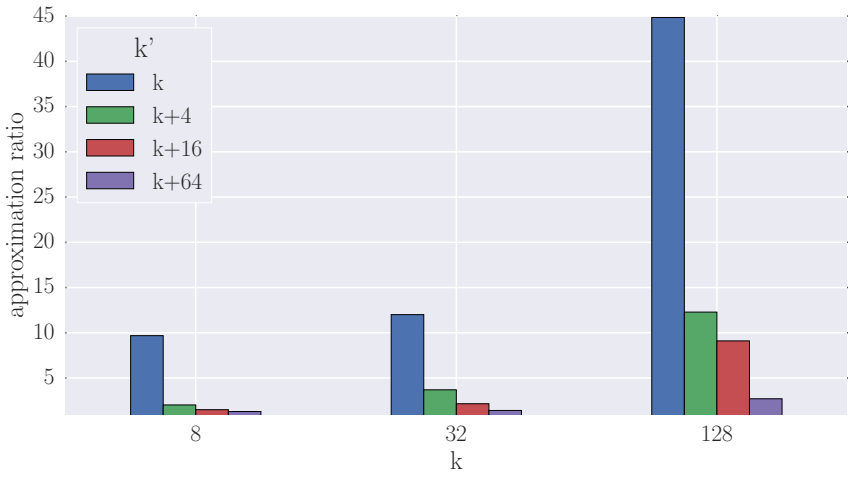

Figure 5.2: Approximation ratios attained by the streaming algorithm for different values of k and k′ on a synthetic dataset of 100 million points.

experimental evaluation of the performance of our algorithms on all the diversity measures is the object of ongoing work. All results reported in the following are obtained as averages over at least 10 runs: the input of the streaming algorithm is randomly shuffled before each run; similarly, the input of each MapReduce run is partitioned randomly.

5.8.1 Streaming algorithm

The first set of experiments investigates the behavior of the streaming algorithm for various values of k, as well as the impact of the core-set size, as controlled by the parameter k′, on the approximation quality. The results of these experiments are reported in Figure 5.1, for the `musiXmatch` dataset, and Figure 5.2. for a synthetic dataset of 100 million points, generated as explained above.

First, we observe that as k increases the remote-edge measure becomes harder

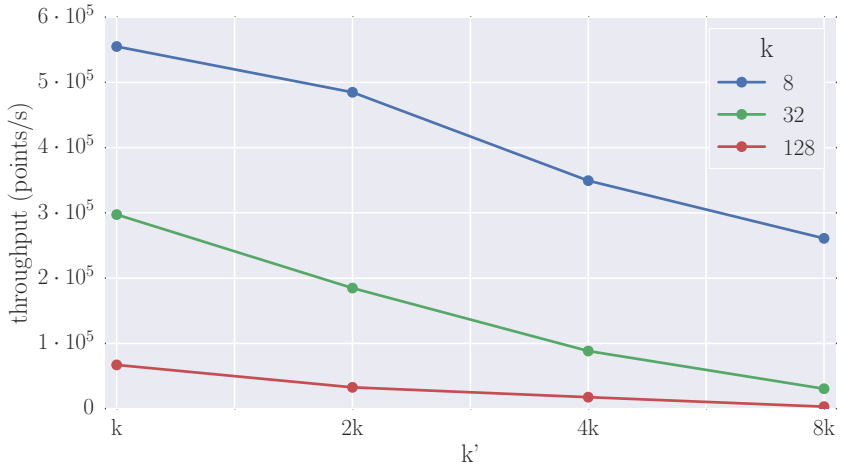

Figure 5.3: Throughput of the kernel of the streaming algorithm on the `musiXmatch` dataset.

to approximate: that is, finding a higher number of diverse elements appears more difficult. On the real-world dataset, because of the high dimensionality of its space, we tested the influence of k' on the approximation with a geometric progression of k' (Figure 5.1). On the synthetic datasets instead (Figure 5.2), since $\mathbb{R}^3$ has a smaller doubling dimension, the effect of k' is evident already with small values, therefore we use a linear progression. As expected, by increasing k' the accuracy of the algorithm increases in both datasets. Observe that although the theory suggests that good approximations require rather large values of $k' = \Omega(k/\varepsilon^D)$, in practice our experiments show that relatively small values of k', not much larger than k, already yield very good approximations, even for real-world datasets whose doubling dimension is unknown, such as the `musiXmatch` dataset.

In Figure 5.3, we consider the performance of the kernel of streaming algorithm, that is, we concentrate on the time required to process each point, ignoring the cost of data acquisition. The rationale is that data may be streamed from sources with very different throughput: our goal is to assess the maximum rate that can be sustained by our algorithm, independently of the source of the stream. We report results for the same combination of parameters shown in Figure 5.1. As expected, the throughput is inversely proportional to both k and k', with values ranging from 3,078 to 544,920 points/s. The throughput supported by our algorithm makes it amenable to be used in streaming pipelines: for instance, in 2013 Twitter[1] averaged at 5,700 tweets/s and peaked at 143,199 tweets/s. In this scenario, the bottleneck of the pipeline may become the data acquisition rather than our core-set construction.

As for the synthetic dataset, the throughput of the algorithm exhibits a behavior with respect to k and k' similar to the one reported in Figure 5.3, but with higher values ranging from 78,260 to 850,615 points/s since the distance function is cheaper to compute.

[1] `https://blog.twitter.com/2013/new-tweets-per-second-record-and-how`

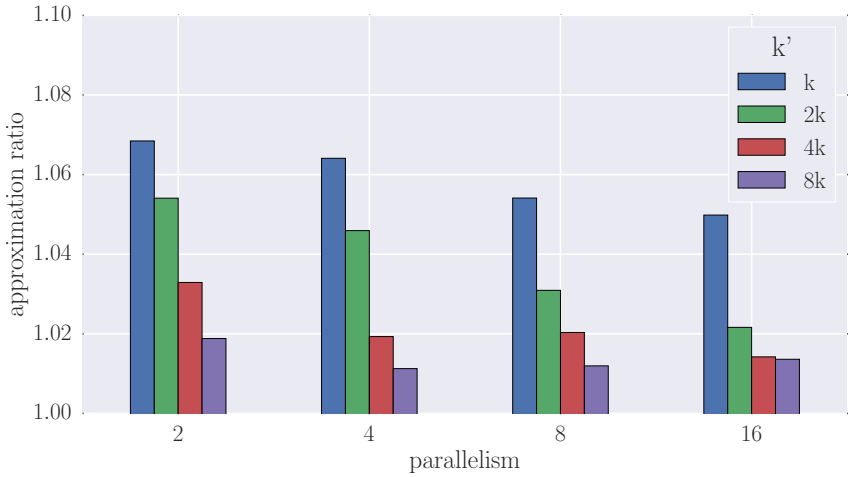

Figure 5.4: Approximation ratios attained by the MR algorithm for different values of k and k' on a synthetic dataset of 100 million points.

5.8.2 MapReduce algorithm

We demonstrate our MapReduce algorithm on the same datasets used in the previous section. For this set of experiments we fixed $k = 128$ and we varied two parameters: size of the core-sets, as controlled by k', and parallelism (i.e., the number of reducers). Because the solution returned by the MapReduce algorithm for $k' = k$ turns out to be already very good, we use a geometric progression for k' to highlight the dependency of the approximation factor on k'. For synthetic datasets, the results are reported in Figure 5.4. For a fixed level of parallelism, we observe that the approximation ratio decreases as k' increases, in accordance to the theory. Moreover, we observe that the approximation ratios are in general better than the ones attained by the streaming algorithm, plausibly because in MapReduce we use a 2-approximation k'-center algorithm to build the core-sets, while in streaming only a weaker 8-approximation k'-center algorithm is available.

Figure 5.4 also reveals that if we fix k' and increase the level of parallelism, the approximation ratio tends to decrease. This can be justified by the observation that the final core-set obtained by aggregating the ones produced by the individual reducers grows larger as the parallelism increases, thus containing more information on the input set. Instead, if we fix the product of k' and the level of parallelism, hence the size of the aggregate core-set, we observe that increasing the parallelism is mildly detrimental to the approximation quality. This is to be expected, since with a fixed space budget in the second round, in the first round each reducer is forced to build a smaller and less accurate core-set as the parallelism increases.

The experiments for the real-world `musiXmatch` dataset highlighted that the GMM k'-center algorithm returns very good core-sets on this high dimensional dataset, yielding approximation ratios very close to 1 even for low values of k'. (Because of this, we do not report a figure.) As remarked above, the more pronounced dependence on k' in the streaming case may be the result of the weaker approximation guarantees of its core-set construction.

Since in real scenarios the input might not be distributed randomly among the

	approximation		time (s)	
k	AFZ	CPPU	AFZ	CPPU
4	1.023	1.012	807.79	1.19
6	1.052	1.018	1,052.39	1.29
8	1.029	1.028	4,625.46	1.12

Table 5.6: Approximation ratios and running times attained by our MR algorithm (CPPU) and AFZ.

reducers as in the previous experiments, we also experimented with an "adversarial" partitioning of the input: each reducer was given points coming from a region of small volume, so as to obfuscate a global view of the pointset. With such adversarial partitioning, the approximation ratios worsen by up to 10%. On the other hand, as k' increases, the time required by a random shuffle of the points among the reducers becomes negligible with respect to the overall running time. Thus, randomly shuffling the points at the beginning may prove cost-effective if larger values of k' are affordable. Recall that in Section 5.5.4 we described how distributing the input randomly allows to save space by selecting less delegate points for the remote-clique, remote-star, remote-bipartition, and remote-tree problems. Ongoing work is focused on studying how such a reduction of the number of delegates impacts the quality of the approximation, when the input is distributed randomly.

5.8.3 Comparison with state of the art

In Table 5.6, we compare our MapReduce algorithm (dubbed CPPU) against its state of the art competitor based on composable core-sets presented in [AFZ15] (dubbed AFZ). Since no code was available for AFZ, we implemented it in MapReduce with the same optimizations used for CPPU. We remark that AFZ employs different core-set constructions for the various diversity measures, whereas our algorithm uses the same construction for all diversity measures. In particular, for remote-edge, AFZ is equivalent to CPPU with $k' = k$, hence the comparison is less interesting and can be derived from the behavior of CPPU itself. Instead, for remote-clique, the core-set construction used by AFZ is based on local search and may exhibit highly superlinear complexity. For remote-clique, we performed the comparison with various values of k, on datasets of 4 million points on the 2-dimensional Euclidean space, using 16 reducers (AFZ was prohibitively slow for higher dimensions and bigger datasets). The datasets were generated as described in the introduction to the experimental section. Also, we ran CPPU with $k' = 128$ in all cases, so as to ensure a good approximation ratio at the expense of a slight increase of the running time. As Table 5.6 shows, CPPU is in all cases at least three orders of magnitude faster than AFZ, while achieving a better quality at the same time.

5.8.4 Scalability

We report on the scalability of our MR algorithm on datasets drawn from $\mathbb{R}^3$, ranging from 100 million points (the same dataset used in subsections 5.8.1 and 5.8.2)

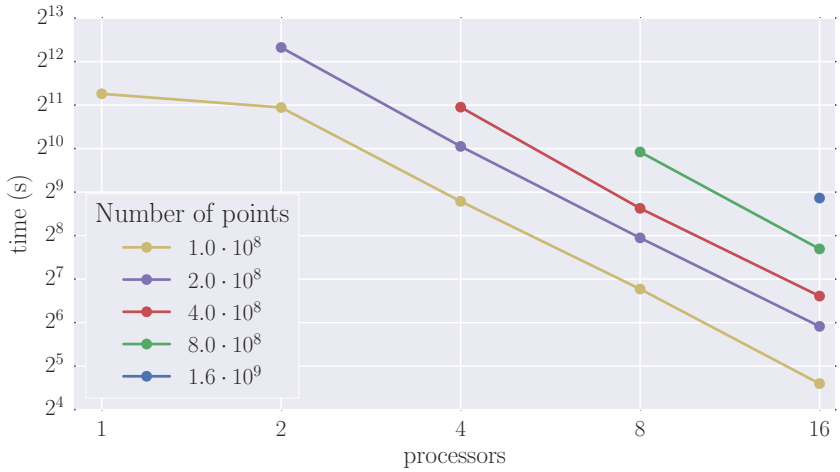

Figure 5.5: Scalability of our algorithms for different number of points and processors. The running time for one processor is obtained with the streaming algorithm.

up to 1.6 billion points. We fixed the size s of the memory required by the final reducer and varied the number of processors used. On a single machine, instead of running MapReduce, which makes little sense, we ran the streaming algorithm with $k' = 2048$, so as to have a final coreset of the same size as the ones found in MapReduce runs. For a given number of processors p and number of points n, we ran the corresponding experiment only if n/p points fit into the main memory of a single processor. As shown in Figure 5.5, for a fixed dataset size, our MapReduce algorithm exhibits super-linear scalability: doubling the number of processors results in a 4-fold gain in running time (at the expense of a mild worsening of the approximation ratio, as pointed out in Subsection 5.8.2). The reason is that each reducer performs $O\left(ns/(kp^2)\right)$ work to build its core-set, where p is the number of reducers, since the core-set construction involves $s/(kp)$ iterations, with each iteration requiring the scan of n/p points.

For the dataset with 100 million points, the MR algorithm outperformed the streaming algorithm in every processor configuration. It must be remarked that the running time reported in Figure 5.5 for the streaming algorithm takes into account also the time needed to stream data from main memory (unlike the throughput reported in Figure 5.3). This is to ensure a fair comparison with MapReduce, where we also take into account the time needed to shuffle data between the first and the second round, and the setup time of the rounds. Also, we note that the streaming algorithm appears to be faster than what the MR algorithm would be if executed on a single processor, and this is probably due to the fact that the former is more cache friendly. If we fix the number of processors, from Figure 5.5 we observe that our algorithm exhibits linear scalability in the number of points.

Chapter 6

Conclusions

In this thesis we considered three problems (estimation of the diameter of large weighted graphs, clustering of uncertain graphs, and diversity maximization) in the big data setting, where the sheer size of data often rules out the application of established algorithmic strategies. In fact, the size of the input data is such that the memory of a single machine is insufficient, therefore we have to resort to algorithms in either the MapReduce or the streaming model. Both computational models require algorithms to use only a limited amount of memory. In streaming, the limit is on the main memory that the processor can use, and algorithms aim at performing only a small number of sequential passes over the input. In MapReduce the aim is to minimize the number of rounds, with limits on both the local memory available to each reducer and on the total aggregate memory.

In this context, a powerful technique for the development of effective algorithms is k-center clustering, where the elements of the input are grouped according to their distance from a set of cluster centers. By doing so, we can build a concise summary of the input, where each cluster center represents all the other elements of its cluster. Therefore, k-center clustering can be used very effectively to reduce the input to a small summary that fits into the limited memory of a single processor, where it can be further processed. Furthermore, k-center clustering is an interesting problem in itself, as witnessed by the large body of literature on this problem.

For the problem of approximating the diameter of large weighted graphs, we saw in Chapter 3 how building a small-radius clustering of suitable size allows to pursue both a small number of MapReduce rounds and a good approximation quality. We obtained an algorithm that computes a polylogarithmic approximation to the input graph's diameter and, for the important class of graphs with bounded doubling dimension, executes in a number of rounds that can be made sublinear in the diameter. Moreover, we proved that our clustering approach can be used to provide a polylogarithmic approximation to the k-center problem on unweighted graphs. To the best of our knowledge, ours are the first parallel approximations for these problems to achieve parallel time sublinear in the diameter using only linear

space, for a relevant class of graphs. We complemented our theoretical findings with an extensive experimental evaluation of the performance of our algorithm. Our implementation of the algorithms is available in the public repository of spark libraries, so as to ease its integration in other software. The experiments we performed showed that our algorithm can scale up to graphs with billions of edges. We also found that the actual approximation factor is typically considerably lower than the one predicted by the theory. Our intuition is that a sharper analysis should be able to remove some logarithmic factor in the approximation, or even yield a constant approximation, and this will be object of future work. Moreover, another line of future research involves extending the proof of our MapReduce k-center approximation to the case of weighted graphs.

In Chapter 4 we studied the problem of clustering uncertain graphs so as to maximize the connection probability of nodes to cluster centers. By proving a triangle inequality-like relation for connection probabilities, which was unknown before, we have shown how our clustering problem can be cast in terms of k-center clustering. However, we saw how the difficulty of estimating low probabilities rules out the application of known k-center algorithms for this problem. We proposed two clustering strategies, a sequential and a concurrent one, aiming at growing clusters only considering nodes with high connection probability to the cluster centers. We then leveraged this behaviour when embedding a progressive sampling strategy into our algorithms, so as to use as few samples as possible, for efficiency. Our algorithms are the first to provide provable guarantees on the quality of the returned solution, contrary to other approaches proposed in the literature. A suite of preliminary experiments has shown that the performance of our algorithms is competitive with respect to previous works. The main line of future research involves the execution of an extensive suite of experiments on a variety of datasets, including synthetic graphs, Protein-Protein Interaction networks, and obfuscated social networks. One of the goals of these experiments will be to compare our sequential and concurrent clustering strategies, both from the perspective of accuracy and performance. We also plan to further assess the performance of our algorithms with respect to other works. Another line of research is about designing strategies to allow a better control on the final number of clusters in the concurrent clustering strategy, which currently returns more clusters than the required ones.

Finally, in Chapter 5 we considered the problem of diversity maximization. Here, we used k-center clustering as a means of building a succinct representation of the input, where each cluster center represents all of its cluster members. By doing so, we were able to develop efficient MapReduce and streaming algorithms for diversity maximization. In MapReduce, using modified versions of the well-known GMM k-center algorithm in each reducer allowed us to obtain constant-rounds algorithms. In streaming, adapting a recent streaming k-center algorithm provided us with means to build one-pass and two-passes algorithms. On spaces of bounded doubling dimension, for all the diversity problems we considered, our algorithms feature approximation factors that can be made arbitrarily close to the approximation factor of the best sequential algorithm for the same problem, if enough local memory is available. To the best of our knowledge, our algorithms are the first to expose a tradeoff between approximation quality and available resources, compared to previous works which provided constant-factor approximations. We demon-

strated the practicality of our approach on both real-world and synthetic datasets of up to billions of elements. These experiments highlighted the space/quality tradeoffs predicted by the theory, as well as the efficiency and scalability of our approach compared to previous works. Furthermore, we verified that our algorithms exhibit very good accuracy even on point sets for which the doubling dimension is not known. Our work is the first to study experimentally diversity-maximization algorithms in MapReduce and streaming, to the best of our knowledge. Future research will be focused on extending the experimental analysis to other diversity measures and datasets. We also plan to investigate experimentally the impact of the reduction of the number of delegates on both the performance and the accuracy of our algorithms, when the input is randomized. Another direction of future work is represented by the application of our core-set strategy to other problems.

References

Our publications

[Cec+15] Matteo Ceccarello, Andrea Pietracaprina, Geppino Pucci, and Eli Upfal. "Space and Time Efficient Parallel Graph Decomposition, Clustering, and Diameter Approximation." In: *Proc. SPAA*. ACM, 2015, pp. 182–191.

[Cec+16a] Matteo Ceccarello, Carlo Fantozzi, Andrea Pietracaprina, Geppino Pucci, and Fabio Vandin. "Clustering Uncertain Graphs". Manuscript. 2016.

[Cec+16b] Matteo Ceccarello, Andrea Pietracaprina, Geppino Pucci, and Eli Upfal. "A Practical Parallel Algorithm for Diameter Approximation of Massive Weighted Graphs". In: *Proc. IPDPS*. 2016, pp. 12–21. DOI: 10.1109/IPDPS.2016.61.

[Cec+17] Matteo Ceccarello, Andrea Pietracaprina, Geppino Pucci, and Eli Upfal. "MapReduce and Streaming Algorithms for Diversity Maximization in Metric Spaces of Bounded Doubling Dimension". In: *PVLDB* 10.5 (2017), pp. 469–480.

[CS15] Matteo Ceccarello and Francesco Silvestri. "Experimental Evaluation of Multi-Round Matrix Multiplication on MapReduce". In: *Proc. ALENEX*. 2015, pp. 119–132. DOI: 10.1137/1.9781611973754.11.

Software

[Cec15] Matteo Ceccarello. *GraDiaS: Graph Diameter in Spark*. http://crono.dei.unipd.it/gradias. 2015.

[Cec16a] Matteo Ceccarello. *DivMax: Diversity Maximization*. https://github.com/Cecca/diversity-maximization. 2016.

[Cec16b] Matteo Ceccarello. *graphx-diameter*. https://spark-packages.org/package/Cecca/graphx-diameter. 2016.

[Cec16c] Matteo Ceccarello. *UGraphC: Uncertain Graph Clustering*. https://github.com/Cecca/ugraph. 2016.

Other references

[Abr+06] Ittai Abraham, Cyril Gavoille, Andrew V. Goldberg, and Dahlia Malkhi. "Routing in Networks with Low Doubling Dimension". In: *Proc. ICDCS*. 2006, p. 75. DOI: 10.1109/ICDCS.2006.72.

[ABS10] Marcel R. Ackermann, Johannes Blömer, and Christian Sohler. "Clustering for metric and nonmetric distance measures". In: *ACM Trans. Algorithms* 6.4 (2010). DOI: 10.1145/1824777.1824779.

[AFZ15] Sepideh Aghamolaei, Majid Farhadi, and Hamid Zarrabi-Zadeh. "Diversity Maximization via Composable Coresets". In: *Proc. CCCG*. 2015.

[AHU74] Alfred V Aho, John E Hopcroft, and Jeffrey D Ullman. *The design and analysis of computer algorithms*. 1974.

[AHV05] Pankaj K. Agarwal, Sariel Har-Peled, and Kasturi R. Varadarajan. "Geometric approximation via coresets". In: *Combinatorial and computational geometry* 52 (2005), pp. 1–30.

[AK11] Albert Angel and Nick Koudas. "Efficient diversity-aware search". In: *Proc. SIGMOD*. 2011, pp. 781–792. DOI: 10.1145/1989323.1989405.

[Aki+15] Tyler Akidau et al. "The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing". In: *Proc. VLDB Endow.* 8.12 (2015), pp. 1792–1803.

[AMS99] Noga Alon, Yossi Matias, and Mario Szegedy. "The Space Complexity of Approximating the Frequency Moments". In: *J. Comput. Syst. Sci.* 58.1 (1999), pp. 137–147. DOI: 10.1006/jcss.1997.1545.

[AMT13] Zeinab Abbassi, Vahab S. Mirrokni, and Mayur Thakur. "Diversity maximization under matroid constraints". In: *Proc. KDD*. 2013, pp. 32–40. DOI: 10.1145/2487575.2487636.

[AMV12] Deepak Ajwani, Ulrich Meyer, and David Veith. "I/O-efficient hierarchical diameter approximation". In: *Proc. ESA*. Springer. 2012, pp. 72–83.

[AR07] Eytan Adar and Christopher Ré. "Managing Uncertainty in Social Networks". In: *IEEE Data Eng. Bull.* 30.2 (2007), pp. 15–22.

[Ast+04] Saurabh Asthana, Oliver D King, Francis D Gibbons, and Frederick P Roth. "Predicting protein complex membership using probabilistic network reliability". In: *Genome research* 14.6 (2004), pp. 1170–1175.

[Bad+14] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. "Streaming submodular maximization: massive data summarization on the fly". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*. New York, New York, USA: ACM Press, 2014, pp. 671–680. DOI: 10.1145/2623330.2623637.

[Bal86] M.O. Ball. "Computation Compexity of Network Reliability Analysis: An Overview". In: *IEEE Transactions on Reliability* R-35.3 (1986), pp. 230–239.

[Bat+14] MohammadHossein Bateni, Aditya Bhaskara, Silvio Lattanzi, and Vahab Mirrokni. "Distributed balanced clustering via mapping coresets". In: *Advances in Neural Information Processing Systems (NIPS '14)* (2014), pp. 2591–2599.

[BB06] Institute of Bioinformatics and Systems Biology. *The MIPS Comprehensive Yeast Genome Database*. `ftp://ftpmips.gsf.de/fungi/Saccharomycetes/CYGD/`. May 2006.

[BEL13] Maria-Florina Balcan, Steven Ehrlich, and Yingyu Liang. "Distributed k-means and k-median clustering on general communication topologies". In: *Proc. NIPS*. 2013, pp. 1995–2003.

[Ber+11] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. "The Million Song Dataset". In: *Proc. ISMIR*. 2011.

[BG09] Benjamin E. Birnbaum and Kenneth J. Goldman. "An Improved Analysis for a Greedy Remote-Clique Algorithm Using Factor-Revealing LPs". In: *Algorithmica* 55.1 (2009), pp. 42–59. DOI: 10.1007/s00453-007-9142-2.

[BGK12] Yair Bartal, Lee-Ad Gottlieb, and Robert Krauthgamer. "The traveling salesman problem: low-dimensionality implies a polynomial time approximation scheme". In: *Proc. STOC*. 2012, pp. 663–672. DOI: 10.1145/2213977.2214038.

[BGM11] Sayan Bhattacharya, Sreenivas Gollapudi, and Kamesh Munagala. "Consideration set generation in commerce search". In: *Proc. WWW*. 2011, pp. 317–326. DOI: 10.1145/1963405.1963452.

[BKL06] Alina Beygelzimer, Sham Kakade, and John Langford. "Cover trees for nearest neighbor". In: *Proc. ICML*. 2006, pp. 97–104. DOI: 10.1145/1143844.1143857.

[BLL09] Nader H. Bshouty, Yi Li, and Philip M. Long. "Using the doubling dimension to analyze the generalization of learning algorithms". In: *J. Comput. Syst. Sci.* 75.6 (2009), pp. 323–335. DOI: 10.1016/j.jcss.2009.01.003.

[BM05] Sanjit Biswas and Robert Morris. "ExOR: opportunistic multi-hop routing for wireless networks". In: *Proc. SIGCOMM*. 2005, pp. 133–144.

[Bol+12] Paolo Boldi, Francesco Bonchi, Aristides Gionis, and Tamir Tassa. "Injecting Uncertainty in Graphs for Identity Obfuscation". In: *Proc. VLDB Endow.* 5.11 (2012), pp. 1376–1387.

[Bru78] Peter Brucker. "On the complexity of clustering problems". In: *Optimization and operations research*. Springer, 1978, pp. 45–54.

[BRV11] Paolo Boldi, Marco Rosa, and Sebastiano Vigna. "HyperANF: Approximating the Neighbourhood Function of Very Large Graphs on a Budget". In: *Proc. WWW*. ACM, 2011, pp. 625–634. DOI: 10.1145/1963405.1963493.

[BS07] Surender Baswana and Sandeep Sen. "A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs". In: *Random Struct. Algorithms* 30.4 (2007), pp. 532–563. DOI: 10.1002/rsa.20130.

[BV13] Paolo Boldi and Sebastiano Vigna. "In-Core Computation of Geometric Centralities with HyperBall: A Hundred Billion Nodes and Beyond". In: *Proc. ICDM*. 2013, pp. 621–628. DOI: 10.1109/ICDMW.2013.10.

[CEZ16] Alfonso Cevallos, Friedrich Eisenbrand, and Rico Zenklusen. "Max-Sum Diversity Via Convex Programming". In: *Proc. SoCG*. 2016, 26:1–26:14. DOI: 10.4230/LIPIcs.SoCG.2016.26.

[CG98] Jaime G. Carbonell and Jade Goldstein. "The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries". In: *Pfoc SIGIR*. 1998, pp. 335–336. DOI: 10.1145/290941.291025.

[CH01] Barun Chandra and Magnús M. Halldórsson. "Approximation Algorithms for Dispersion Problems". In: *J. Algorithms* 38.2 (2001), pp. 438–465. DOI: 10.1006/jagm.2000.1145.

[Cha+04] Moses Charikar, Chandra Chekuri, Tomas Feder, and Rajeev Motwani. "Incremental Clustering and Dynamic Information Retrieval". In: *SIAM Journal on Computing* 33.6 (2004), pp. 1417–1440. DOI: 10.1137/S0097539702418498.

[Cha+99] Moses Charikar, Sudipto Guha, Éva Tardos, and David B Shmoys. "A constant-factor approximation algorithm for the k-median problem". In: *Proc. STOC*. ACM. 1999, pp. 1–10.

[CL07] Zhiyuan Chen and Tao Li. "Addressing Diverse User Preferences in SQL-query-result Navigation". In: *Proc. SIGMOD*. ACM, 2007, pp. 641–652.

[Coh00] Edith Cohen. "Polylog-time and near-linear work approximation scheme for undirected shortest paths". In: *Journal of the ACM* 47.1 (2000), pp. 132–166.

[Coh15] Edith Cohen. "All-Distances Sketches, Revisited: HIP Estimators for Massive Graphs Analysis". In: *IEEE Transactions on Knowledge and Data Engineering* 27.9 (Sept. 2015), pp. 2320–2334. DOI: 10.1109/TKDE.2015.2411606.

[Coh98] Edith Cohen. "Fast algorithms for constructing t-spanners and paths with stretch t". In: *SIAM Journal on Computing* 28.1 (1998), pp. 210–236.

[Col+07] S.R. Collins et al. "Toward a comprehensive atlas of the physical interactome of Saccharomyces cerevisiae". In: *Molecular & Cellular Proteomics* 6.3 (2007), pp. 439–450.

[Cre+12] Pierluigi Crescenzi, Roberto Grossi, Leonardo Lanzi, and Andrea Marino. "On Computing the Diameter of Real-World Directed (Weighted) Graphs". In: *Proc. SEA*. 2012, pp. 99–110. DOI: 10.1007/978-3-642-30850-5_10.

[Cre+13] Pilu Crescenzi, Roberto Grossi, Michel Habib, Leonardo Lanzi, and Andrea Marino. "On computing the diameter of real-world undirected graphs". In: *Theoretical Computer Science* 514 (2013). Graph Algorithms and Applications: in Honor of Professor Giorgio Ausiello, pp. 84–95. DOI: http://dx.doi.org/10.1016/j.tcs.2012.09.018.

[D+73] Richard O Duda, Peter E Hart, et al. *Pattern classification and scene analysis*. Vol. 3. Wiley New York, 1973.

[Dai+07] Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. "Co-clustering based classification for out-of-domain documents". In: *Proc. KDD*. 2007, pp. 210–219. DOI: 10.1145/1281192.1281218.

[DFR09] Camil Demetrescu, Irene Finocchi, and Andrea Ribichini. "Trading off Space for Passes in Graph Streaming Problems". In: *ACM Trans. Algorithms* 6.1 (Dec. 2009), 6:1–6:17. DOI: 10.1145/1644015.1644021.

[DG08] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[Don08] Stijn van Dongen. "Graph Clustering Via a Discrete Uncoupling Process". In: *SIAM J. Matrix Analysis Applications* 30.1 (2008), pp. 121–141. DOI: 10.1137/040608635.

[Dwa69] Meyer Dwass. "The total progeny in a branching process and a related random walk". In: *Journal of Applied Probability* 6.3 (1969), pp. 682–686.

[EIM11] Alina Ene, Sungjin Im, and Benjamin Moseley. "Fast Clustering Using MapReduce". In: *Proc. KDD*. ACM, 2011, pp. 681–689. DOI: 10.1145/2020408.2020515.

[EN89] Erhan Erkut and Susan Neuman. "Analytical models for locating undesirable facilities". In: *European Journal of Operational Research* 40.3 (1989), pp. 275–291.

[EN91] Erhan Erkut and Susan Neuman. "Comparison of 4 models for dispersing facilities". In: *Infor* 29.2 (1991), pp. 68–86.

[Erk90] Erhan Erkut. "The discrete p-dispersion problem". In: *European Journal of Operational Research* 46.1 (1990), pp. 48–60.

[FB13] Wei Fan and Albert Bifet. "Mining big data: current status, and forecast to the future". In: *ACM SIGKDD Explorations Newsletter* 14.2 (2013), pp. 1–5.

[Fei+02] Joan Feigenbaum, Sampath Kannan, Martin Strauss, and Mahesh Viswanathan. "An Approximate L1-Difference Algorithm for Massive Data Streams". In: *SIAM J. Comput.* 32.1 (2002), pp. 131–151. DOI: 10.1137/S0097539799361701.

[FFF99] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. "On Power-law Relationships of the Internet Topology". In: *Proc. SIGCOMM*. 1999, pp. 251–262. DOI: 10.1145/316188.316229.

[Fla+08] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. "HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm". In: *DMTCS Proceedings* 1 (2008).

[FM03] Sándor P. Fekete and Henk Meijer. "Maximum Dispersion and Geometric Maximum Weight Cliques". In: *Algorithmica* 38.3 (2003), pp. 501–511. DOI: 10.1007/s00453-003-1074-x.

[FM10] S.A. Friedler and D.M. Mount. "Approximation algorithm for the kinetic robust K-center problem". In: *Computational Geometry* 43.6 (2010), pp. 572–586.

[FM83] Philippe Flajolet and G.Nigel Martin. "Probabilistic Counting". In: *Proc. FOCS*. Nov. 1983, pp. 76–82. DOI: 10.1109/SFCS.1983.46.

[Gar+15] Kiran Garimella, Gianmarco De Francisci Morales, Aristides Gionis, and Mauro Sozio. "Scalable Facility Location for Massive Graphs on Pregel-like Systems". In: *Proc. CIKM*. 2015, pp. 273–282. DOI: 10.1145/2806416.2806508.

[Gav+06] A. C. Gavin et al. "Proteome survey reveals modularity of the yeast cell machinery". In: *Nature* 440.7084 (2006), pp. 631–636.

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system". In: *Proc. SIGOPS*. Vol. 37. 5. ACM. 2003, pp. 29–43.

[Gho+07] Joy Ghosh, Hung Q Ngo, Seokhoon Yoon, and Chunming Qiao. "On a Routing Problem Within Probabilistic Graphs and its Application to Intermittently Connected Networks". In: *Proc. INFOCOM*. 2007, pp. 1721–1729.

[Gil+02] Anna C. Gilbert et al. "Fast, small-space algorithms for approximate histogram maintenance". In: *Proc. STOC*. 2002, pp. 389–398. DOI: 10.1145/509907.509966.

[GK13] Lee-Ad Gottlieb and Robert Krauthgamer. "Proximity Algorithms for Nearly Doubling Spaces". In: *SIAM J. Discrete Math.* 27.4 (2013), pp. 1759–1769. DOI: 10.1137/120874242.

[GKK14] Lee-Ad Gottlieb, Aryeh Kontorovich, and Robert Krauthgamer. "Efficient Classification for Metric Data". In: *IEEE Trans. Information Theory* 60.9 (2014), pp. 5750–5759. DOI: 10.1109/TIT.2014.2339840.

[GKL03] Anupam Gupta, Robert Krauthgamer, and James R. Lee. "Bounded Geometries, Fractals, and Low-Distortion Embeddings". In: *Proc. FOCS*. 2003, pp. 534–543. DOI: 10.1109/SFCS.2003.1238226.

[Gon85] Teofilo F. Gonzalez. "Clustering to minimize the maximum intercluster distance". In: *Theoretical Computer Science* 38 (1985), pp. 293–306. DOI: http://dx.doi.org/10.1016/0304-3975(85)90224-5.

[GS09] Sreenivas Gollapudi and Aneesh Sharma. "An axiomatic approach for result diversification". In: *Proc. WWW*. 2009, pp. 381–390. DOI: 10.1145/1526709.1526761.

[GSZ11] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. "Sorting, Searching, and Simulation in the MapReduce Framework". In: *Proc. ISAAC*. 2011, pp. 374–383. DOI: 10.1007/978-3-642-25591-5_39.

[Gu+14] Yu Gu, Chunpeng Gao, Gao Cong, and Ge Yu. "Effective and Efficient Clustering Methods for Correlated Probabilistic Graphs". In: *IEEE Trans. Knowl. Data Eng.* 26.5 (2014), pp. 1117–1130.

[Hal+99] Magnús M. Halldórsson, Kazuo Iwano, Naoki Katoh, and Takeshi Tokuyama. "Finding Subsets Maximizing Minimum Structures". In: *SIAM Journal on Discrete Mathematics* 12.3 (1999), pp. 342–359. DOI: 10.1137/S0895480196309791.

[HK07] Sariel Har-Peled and Akash Kushal. "Smaller coresets for k-median and k-means clustering". In: *Discrete and Computational Geometry* 37.1 (Jan. 2007), pp. 3–19. DOI: 10.1007/s00454-006-1271-x.

[HM04] Sariel Har-Peled and Soham Mazumdar. "On coresets for k-means and k-median clustering". In: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing - STOC '04*. New York, New York, USA: ACM Press, 2004, pp. 291–300. DOI: 10.1145/1007352.1007400.

[HRR98] Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. "Computing on data streams". In: *Proc. DIMACS*. 1998, pp. 107–118.

[HRT97] Refael Hassin, Shlomi Rubinstein, and Arie Tamir. "Approximation algorithms for maximum dispersion". In: *Oper. Res. Lett.* 21.3 (1997), pp. 133–137. DOI: 10.1016/S0167-6377(97)00034-5.

[Ind+14] Piotr Indyk, Sepideh Mahabadi, Mohammad Mahdian, and Vahab S. Mirrokni. "Composable core-sets for diversity and coverage maximization". In: *Proc. PODS*. 2014, pp. 100–108. DOI: 10.1145/2594538.2594560.

[Isa+07] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. "Dryad: distributed data-parallel programs from sequential building blocks". In: *ACM SIGOPS Operating Systems Review*. Vol. 41. 3. ACM. 2007, pp. 59–72.

[Jai10] Anil K. Jain. "Data clustering: 50 years beyond K-means". In: *Pattern Recognition Letters* 31.8 (2010), pp. 651–666. DOI: 10.1016/j.patrec.2009.09.011.

[Jin+11] Ruoming Jin, Lin Liu, Bolin Ding, and Haixun Wang. "Distance-Constraint Reachability Computation in Uncertain Graphs". In: *Proc. VLDB Endow.* 4.9 (2011), pp. 551–562.

[Kan+11] U. Kang, Charalampos E. Tsourakakis, Ana Paula Appel, Christos Faloutsos, and Jure Leskovec. "HADI: Mining Radii of Large Graphs". In: *ACM Trans. Knowl. Discov. Data* 5.2 (Feb. 2011), 8:1–8:24. DOI: 10.1145/1921632.1921634.

[Ken75] David G Kendall. "The genealogy of genealogy branching processes before (and after) 1873". In: *Bulletin of the London Mathematical Society* 7.3 (1975), pp. 225–253.

[KL04] Robert Krauthgamer and James R. Lee. "Navigating nets: simple algorithms for proximity search". In: *Proc. SODA*. 2004, pp. 798–807.

[KPT13] George Kollios, Michalis Potamias, and Evimaria Terzi. "Clustering Large Probabilistic Graphs". In: *IEEE Trans. Knowl. Data Eng.* 25.2 (2013), pp. 325–336.

[Kro+06] Nevan J Krogan et al. "Global landscape of protein complexes in the yeast Saccharomyces cerevisiae". In: *Nature* 440.7084 (2006), pp. 637–643.

[KRX08] Goran Konjevod, Andréa W. Richa, and Donglin Xia. "Dynamic Routing and Location Services in Metrics of Low Doubling Dimension". In: *Proc. DISC*. 2008, pp. 379–393. DOI: 10.1007/978-3-540-87779-0_26.

[KSI06] Georgia Koutrika, Alkis Simitsis, and Yannis E. Ioannidis. "Précis: The Essence of a Query Answer". In: *Proc. ICDE*. 2006, pp. 69–78. DOI: 10.1109/ICDE.2006.114.

[KSV10] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. "A Model of Computation for MapReduce". In: *Proc. SODA*. 2010, pp. 938–948. DOI: 10.1137/1.9781611973075.76.

[KSW04] Jon M. Kleinberg, Aleksandrs Slivkins, and Tom Wexler. "Triangulation and Embedding Using Small Sets of Beacons". In: 2004, pp. 444–453. DOI: 10.1109/FOCS.2004.70.

[Kub87] Michael J Kuby. "Programming Models for Facility Dispersion: The p-Dispersion and Maxisum Dispersion Problems". In: *Geographical Analysis* 19.4 (1987), pp. 315–329.

[Lan01] Doug Laney. "3D data management: Controlling data volume, velocity and variety". In: *META Group Research Note* 6 (2001), p. 70.

[Lat+11] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. "Filtering: a method for solving graph problems in mapreduce". In: *Proc. SPAA*. ACM. 2011, pp. 85–94.

[Lin+12] Liu Lin, Jin Ruoming, Charu Aggarwal, and Shen Yelong. "Reliable clustering on uncertain graphs". In: *Proc. ICDM*. Dec. 2012, pp. 459–468.

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed*. Cambridge University Press, 2014.

[Mal+15] Gustavo Malkomes, Matt J Kusner, Wenlin Chen, Kilian Q Weinberger, and Benjamin Moseley. "Fast Distributed k-Center Clustering with Outliers on Massive Data". In: *Proc. NIPS*. Curran Associates, Inc., 2015, pp. 1063–1071.

[MB08] Michael Masin and Yossi Bukchin. "Diversity Maximization Approach for Multiobjective Optimization". In: *Operations Research* 56.2 (2008), pp. 411–424. DOI: 10.1287/opre.1070.0413.

[MBE06] Vishal Monga, Arindam Banerjee, and Brian L. Evans. "A clustering based approach to perceptual image hashing". In: *IEEE Trans. Information Forensics and Security* 1.1 (2006), pp. 68–79. DOI: 10.1109/TIFS.2005.863502.

[Mew+04] Hans-Werner Mewes et al. “MIPS: analysis and annotation of proteins from whole genomes”. In: *Nucleic acids research* 32.suppl 1 (2004), pp. D41–D44.

[Mey08] Ulrich Meyer. “On Trade-Offs in External-Memory Diameter-Approximation”. English. In: *Algorithm Theory – SWAT 2008*. Vol. 5124. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 426–436. DOI: 10.1007/978-3-540-69903-3_38.

[MLH09] Clémence Magnien, Matthieu Latapy, and Michel Habib. “Fast computation of empirically tight bounds for the diameter of massive graphs”. In: *Journal of Experimental Algorithmics (JEA)* 13 (2009), p. 10.

[MPX13] Gary L. Miller, Richard Peng, and Shen Chen Xu. “Parallel graph decompositions using random shifts”. In: *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada - July 23 - 25, 2013*. 2013, pp. 196–203. DOI: 10.1145/2486159.2486180.

[MRK06] Sean M. McNee, John Riedl, and Joseph A. Konstan. “Being accurate is not enough: how accuracy metrics have hurt recommender systems”. In: *Proc. CHI*. 2006, pp. 1097–1101. DOI: 10.1145/1125451.1125659.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[MS03] Ulrich Meyer and Peter Sanders. “Δ-stepping: a parallelizable shortest path algorithm”. In: *Journal of Algorithms* 49.1 (2003). 1998 European Symposium on Algorithms, pp. 114–152. DOI: http://dx.doi.org/10.1016/S0196-6774(03)00076-2.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[Mur+10] Richard C Murphy, Kyle B Wheeler, Brian W Barrett, and James A Ang. “Introducing the graph 500”. In: *Cray User's Group (CUG)* (2010).

[Mut05] S. Muthukrishnan. “Data Streams: Algorithms and Applications”. In: *Foundations and Trends in Theoretical Computer Science* 1.2 (2005). DOI: 10.1561/0400000002.

[MZ15] Vahab Mirrokni and Morteza Zadimoghaddam. “Randomized Composable Core-sets for Distributed Submodular Maximization”. In: *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing - STOC '15*. New York, New York, USA: ACM Press, 2015, pp. 153–162. DOI: 10.1145/2746539.2746624.

[MZR09] Sean A. Munson, Daniel Xiaodan Zhou, and Paul Resnick. “Sidelines: An Algorithm for Increasing Diversity in News and Opinion Aggregators”. In: *Proc. ICWSM*. 2009.

[NYP12] Tamás Nepusz, Haiyuan Yu, and Alberto Paccanaro. “Detecting overlapping protein complexes in protein-protein interaction networks”. In: *Nature methods* 9.5 (2012), pp. 471–472.

[Par+15] Panos Parchas, Francesco Gullo, Dimitris Papadias, and Francesco Bonchi. "Uncertain Graph Processing through Representative Instances". In: *ACM Trans. Database Syst.* 40.3 (2015), p. 20. DOI: 10.1145/2818182.

[Pen+05] Yi Peng, Gang Kou, Yong Shi, and Zhengxin Chen. "Improving Clustering Analysis for Credit Card Accounts Classification". In: *Proc. ICCS*. 2005, pp. 548–553. DOI: 10.1007/11428862_75.

[Pet04] Seth Pettie. "A new approach to all-pairs shortest paths on real-weighted graphs". In: *Theoretical Computer Science* 312.1 (2004), pp. 47–74. DOI: http://dx.doi.org/10.1016/S0304-3975(03)00402-X.

[PGF02] Christopher R. Palmer, Phillip B. Gibbons, and Christos Faloutsos. "ANF: A Fast and Scalable Tool for Data Mining in Massive Graphs". In: *Proc. KDD*. ACM, 2002, pp. 81–90. DOI: 10.1145/775047.775059.

[Pie+12] Andrea Pietracaprina, Geppino Pucci, Matteo Riondato, Francesco Silvestri, and Eli Upfal. "Space-round tradeoffs for MapReduce computations". In: *Proc. ICS*. ACM. 2012, pp. 235–244.

[Pot+10] Michalis Potamias, Francesco Bonchi, Aristides Gionis, and George Kollios. "k-Nearest Neighbors in Uncertain Graphs." In: *Proc. VLDB Endow.* 3.1 (2010), pp. 997–1008.

[PR02] Seth Pettie and Vijaya Ramachandran. "Computing Shortest Paths with Comparisons and Additions". In: *Proc. SODA*. Society for Industrial and Applied Mathematics, 2002, pp. 267–276.

[RG06] Cole R and L.A. Gottlieb. "Searching Dynamic Point Sets in Spaces with Bounded Doubling Dimension". In: *Proc. STOC*. 2006, pp. 574–583.

[RRT07] Daniel J. Rosenkrantz, S. S. Ravi, and Giri Kumar Tayi. "Approximation Algorithms for Facility Dispersion". In: *Handbook of Approximation Algorithms and Metaheuristics*. 2007. DOI: 10.1201/9781420010749.ch38.

[RRT94] S. S. Ravi, Daniel J. Rosenkrantz, and Giri Kumar Tayi. "Heuristic and Special Case Algorithms for Dispersion Problems". In: *Operations Research* 42.2 (1994), pp. 299–310. DOI: 10.1287/opre.42.2.299.

[Sch07] Satu Elisa Schaeffer. "Graph clustering". In: *Computer Science Review* 1.1 (2007), pp. 27–64.

[Shv+10] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. "The Hadoop Distributed File System". In: *Proc. MSST*. IEEE Computer Society, 2010, pp. 1–10. DOI: 10.1109/MSST.2010.5496972.

[Sli07] Aleksandrs Slivkins. "Distance estimation and object location via rings of neighbors". In: *Distributed Computing* 19.4 (2007), pp. 313–333. DOI: 10.1007/s00446-006-0015-8.

[Tal04] Kunal Talwar. "Bypassing the embedding: algorithms for low dimensional metrics". In: *Proc. STOC*. 2004, pp. 281–290. DOI: 10.1145/1007352.1007399.

[Tam91] Arie Tamir. "Obnoxious Facility Location on Graphs". In: *SIAM J. Discrete Math.* 4.4 (1991), pp. 550–567. DOI: 10.1137/0404048.

[Val79] Leslie G. Valiant. "The Complexity of Enumeration and Reliability Problems". In: *SIAM J. Comput.* 8.3 (1979), pp. 410–421.

[Whi15] Tom White. *Hadoop - The Definitive Guide: Storage and Analysis at Internet Scale (4. ed., revised & updated)*. O'Reilly, 2015.

[Wil12] Virginia Vassilevska Williams. "Multiplying matrices faster than Coppersmith-Winograd". In: *Proc. STOC*. ACM. 2012, pp. 887–898.

[Wu13] Yong Cheng Wu. "Active Learning Based on Diversity Maximization". In: *Applied Mechanics and Materials* 347.10 (2013), pp. 2548–2552.

[Xin+06] Dong Xin, Hong Cheng, Xifeng Yan, and Jiawei Han. "Extracting redundancy-aware top-k patterns". In: *Proc. KDD*. 2006, pp. 444–453. DOI: 10.1145/1150402.1150452.

[Yan+08] Allen Y. Yang, John Wright, Yi Ma, and Shankar S. Sastry. "Unsupervised segmentation of natural images via lossy data compression". In: *Computer Vision and Image Understanding* 110.2 (2008), pp. 212–225. DOI: 10.1016/j.cviu.2007.07.005.

[Yan+15] Yi Yang, Zhigang Ma, Feiping Nie, Xiaojun Chang, and Alexander G. Hauptmann. "Multi-Class Active Learning by Uncertainty Sampling with Diversity Maximization". In: *International Journal of Computer Vision* 113.2 (2015), pp. 113–127. DOI: 10.1007/s11263-014-0781-x.

[YLA09] Cong Yu, Laks V. S. Lakshmanan, and Sihem Amer-Yahia. "Recommendation Diversification Using Explanations". In: *Proc. ICDE*. 2009, pp. 1299–1302. DOI: 10.1109/ICDE.2009.225.

[Zah+10] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. "Spark: Cluster Computing with Working Sets." In: *Proc. HotCloud*. USENIX Association, 2010.