algorithms

MDPI

Article

Distributed Graph Diameter Approximation

Matteo Ceccarello [1,*], Andrea Pietracaprina [2], Geppino Pucci [2] and Eli Upfal [3]

[1] Faculty of Computer Science, Free University of Bozen, 39100 Bolzano, Italy
[2] Department of Information Engineering, University of Padova, 35131 Padova, Italy; andrea.pietracaprina@unipd.it (A.P.); geppino.pucci@unipd.it (G.P.)
[3] Department of Computer Science, Brown University, Providence, RI 02912, USA; eli@cs.brown.edu
* Correspondence: mceccarello@unibz.it

Received: 30 July 2020; Accepted: 29 August 2020; Published: 1 September 2020

Abstract: We present an algorithm for approximating the diameter of massive weighted undirected graphs on distributed platforms supporting a MapReduce-like abstraction. In order to be efficient in terms of both time and space, our algorithm is based on a decomposition strategy which partitions the graph into disjoint clusters of bounded radius. Theoretically, our algorithm uses linear space and yields a polylogarithmic approximation guarantee; most importantly, for a large family of graphs, it features a round complexity asymptotically smaller than the one exhibited by a natural approximation algorithm based on the state-of-the-art Δ-stepping SSSP algorithm, which is its only practical, linear-space competitor in the distributed setting. We complement our theoretical findings with a proof-of-concept experimental analysis on large benchmark graphs, which suggests that our algorithm may attain substantial improvements in terms of running time compared to the aforementioned competitor, while featuring, in practice, a similar approximation ratio.

Keywords: graph analytics; parallel graph algorithms; weighted graph decomposition; weighted diameter approximation; MapReduce

1. Introduction

The analysis of very large (typically sparse) graphs is becoming a central tool in numerous domains, including geographic information systems, social sciences, computational biology, computational linguistics, semantic search and knowledge discovery, and cyber security [1]. A fundamental primitive for graph analytics is the estimation of a graph's diameter, defined as the maximum (weighted) distance between nodes in the same connected component. Computing exactly the diameter of a graph is essentially as expensive as computing All-Pairs Shortest Paths [2], which is too costly for very large graphs. Therefore, when dealing with these graphs, we need to target approximate solutions in order to ensure reasonable running times. Furthermore, large graphs might be too large to fit in the memory of a single machine: in such cases, resorting to distributed algorithms running on large clusters of machines is key to the solution of the problem. Unfortunately, state-of-the-art parallel strategies for diameter estimation are either space inefficient or incur long critical paths. Thus, these strategies are unfeasible for dealing with huge graphs, especially on distributed platforms characterized by limited local memory and high communication costs, (e.g., clusters of loosely coupled commodity servers supporting a MapReduce-like abstraction), which are widely used for big data tasks, and represent the target computational scenario of this paper. In this setting, the challenge is to minimize the number of communication rounds while using linear aggregate space and small (i.e., substantially sublinear) space in the individual processors.

To cope with the above issues, we develop a novel parallel strategy for approximating the diameter of large undirected graphs, both weighted and unweighted, which is suitable for implementation on MapReduce-like computing frameworks.

1.1. Related Work

1.1.1. Exact Algorithms

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 classical sequential algorithms to solve this problem. On an n-node weighted graph, Floyd–Warshall's algorithm runs in $\Theta(n^3)$ time and requires $\Theta(n^2)$ space. On graphs with non-negative weights, running Dijkstra's algorithm (with Fibonacci heaps) from each node requires $O(mn + n^2 \log n)$ time, which is better than $O(n^3)$ for sparse graphs. Also, there are some works improving on this latter $O(mn + n^2 \log n)$ bound. For undirected graphs, a faster algorithm with running time $O(mn \log \alpha(m, n))$, where $\alpha(m, n)$ is the inverse-Ackermann function, is presented in [3]. On unweighted graphs, Dijkstra's algorithm can be replaced by a simple Breadth First Search, with an overall running time of $O(mn)$. However, Dijkstra's algorithm and the BFS are difficult to parallelize. In fact, one may run the n instances of Dijkstra's algorithm or BFS (one from each node of the graph) on separate processors in parallel, however this approach is not applicable when the graph does not fit in the memory available to a single processor.

An alternative 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 $O(n^{2.3727} \log n)$ time [4]. By applying a clever recursive decomposition of the problem, we can drop the repeated squaring, saving a logarithmic factor in the time complexity [5] (pp. 201–206). The space requirement in any case is $O(n^2)$, which rules out matrix-based approaches in the context of large graphs.

The drawback of approaches that solve the APSP problem is that they are either space inefficient or inherently sequential with at least quadratic running times, making them not applicable to very large graphs.

1.1.2. Approximation Algorithms

A very simple approximation algorithm, in both weighted and unweighted cases, 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 has spurred a line of research in which a few SSSP instances are solved, starting from carefully selected nodes, to get a good approximation of the diameter. Magnien et al. [6] consider the case of unweighted graphs, and study 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. develop IFUB algorithm for unweighted graphs [7], and the DIFUB algorithm for weighted graphs [8], both of which compute the diameter of a given graph exactly. In the worst case, these algorithms require to solve the APSP problem. However, with an extensive experimental evaluation the authors show 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.

Another line of research investigates the computation of the *neighbourhood function* of graphs which can be used, among other things, to approximate the diameter of unweighted graphs. The neighbourhood function $N_G(h)$ of a graph G, also called the *hop plot* [9], is the number of pairs of nodes that are within distance h, for every $h \geq 0$ [10]. 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. [10] introduces the *Approximate Neighbourhood Function* algorithm (abbreviated ANF), where Flajolet–Martin probabilistic counters [11] are used to count the nodes reachable within distance h: each counter requires only $O(\log n)$ bits, therefore lowering to $O(n \log n)$ the overall memory requirement. This result is further improved by Boldi et al. [12] by replacing Flajolet–Martin counters with HyperLogLog counters [13]. HyperLogLog counters require only $O(\log \log n)$ bits to count the number of reachable nodes, so the overall memory requirement of the algorithm (called HyperANF) drops to $O(n \log \log n)$. Boldi et al. [14] further extend the algorithm to support the computation of many graph centrality measures.

These neighbourhood function approximation algorithms were originally developed for shared memory machines. A MapReduce implementation of ANF, called HADI, is presented by Kang et al. [15]. 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. Most importantly, all three of ANF, HyperANF and HADI work *exclusively* on graphs with unweighted edges. In [14], the HyperANF approach is extended to work for graphs with integer weights on the *nodes*, which are arguably less common than edge-weighted graphs.

In the external memory model, where performing even a single BFS has a very high I/O complexity, Meyer [16] proposes an approximation algorithm to compute the diameter in the unweighted case in terms of the diameter of an auxiliary graph derived from a clustering of the original graph built around random centers. However, the algorithm features a high approximation factor and might suffer from a long critical path if implemented in a distributed environment. Also, no analytical guarantees on the approximation factor are known in the presence of weights. A similar diameter-approximation strategy could be devised by using the parallel clusterings presented in [17,18]. However, these strategies apply again only to unweighted graphs and, moreover, they aim at minimizing the number of inter-cluster edges but cannot provide tight guarantees on the clustering radius, which in turn could result in poor approximations to the diameter. In fact, in this work we adopt a similar approach but devise a novel distributed clustering strategy which is able to circumvent all of the aforementioned limitations.

Clustering is also used to approximate shortest paths between nodes, which can indirectly be used to approximate the diameter. In particular, Cohen [19] presents a PRAM algorithm which uses clustering to approximate shortest-path distances. For sparse graphs with $m \in \Theta(n)$, this algorithm features $O(n^\delta)$ depth, for any fixed constant $\delta \in (0, 1)$, but incurs a polylogarithmic space blow-up. The algorithm is 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.

1.1.3. Δ-Stepping

In a seminal work, Meyer and Sanders [20] propose a PRAM algorithm, called Δ-*stepping*, for the SSSP problem. This algorithm exercises a tradeoff between work-efficiency and parallel time through a parameter Δ, and can be used to obtain a 2-approximation to a weighted graph's diameter. The analysis of the original paper can be easily adapted to show that the algorithm runs in a number of rounds at least linear in the diameter, if implemented in MapReduce. Therefore, it suffers from the same drawbacks of the BFS for the unweighted case.

The authors also propose a preprocessing of the graph to speed up the execution [20]. This preprocessing involves adding so-called *shortcuts*: the graph is augmented with new edges between nodes at distance $\leq \Delta$. While this preprocessing can be shown to improve the running time, 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 feasible.

1.2. Our Contribution

This paper extends and improves on earlier conference papers on unweighted [21] and weighted [22] graphs. In this work, we present a simpler algorithm that can handle both weighted and

unweighted cases. The high level idea of our algorithm is the following. Given a (weighted) graph G, we construct in parallel a smaller *auxiliary graph*, whose size is tuned to fit into the local memory of a single machine. To this purpose, first the input graph is partitioned into clusters grown around suitably chosen *centers*. Then, the auxiliary graph is built by associating each node with a distinct center, and defining edges between nodes corresponding to centers of adjacent cluster, with weights upper bounding the distance between the centers. The approximation of the diameter of G 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.

For a (weighted) graph G with n nodes and m edges we prove that our algorithm attains an $O(\log^3 n)$ approximation ratio, with high probability. Moreover, we show how the algorithm can be implemented in the MapReduce model with $O(n^\epsilon)$ local memory available at each machine, for any fixed $\epsilon \in (0,1)$, and $\Theta(m)$ aggregate memory. The number of rounds is expressed as a function of ϵ, n and the maximum number of edges in the simple paths that are traversed to connect the nodes of G to the centers of their respective clusters. We also analyze the round complexity in terms of the doubling dimension D of G [23], a notion which broadly corresponds to the Euclidean dimension with respect to the metric space on the nodes induced by the shortest-path distances in the graph (see Section 2.2 for a formal definition). In particular, we show that on graphs of bounded doubling dimension (an important family including, for example, multidimensional arrays) and with random edge weights, if the local memory available to each processor is $\Theta(n^\epsilon)$, then the round complexity of our algorithm becomes asymptotically smaller than the unweighted diameter of the graph by a factor $\Theta(n^{\epsilon/(2D)}/\text{polylog}(n))$. This is a substantial improvement with respect to the MapReduce implementation of Δ-stepping whose round complexity, as observed before, is linear in the diameter when linear overall space is targeted.

As a proof of concept, we complement our theoretical findings with an experimental analysis on large benchmark graphs. The experiments demonstrate that when the input graph is so large that does not fit into the memory of a single machine, our algorithm attains substantial improvements in terms of the running time compared to Δ-stepping, while featuring, in practice, a similar approximation ratio, which turns out to be much better than what predicted by the analysis. The experiments also show that our algorithm features a good scalability with respect to the graph size.

The novelties of this paper with respect to our two conference papers [21,22] are the following: the presentations of the algorithms for the unweighted and weighted cases have been unified; a novel and simpler clustering strategy has been adopted (adapting the analysis accordingly); and a novel set of experiments has been carried out using a more efficient implementation of our algorithm devised on the Timely Dataflow framework [24].

1.3. Structure of the Paper

The rest of the paper is structured as follows. Section 2 introduces the basic concepts and defines our reference computational model. Section 3 illustrates the clustering-based graph decomposition at the core of the diameter approximation algorithm, which is described in the subsequent Section 4. Section 5 shows how to implement the algorithm in the MapReduce model and analyzes its space and round complexities, while Section 6 specializes the result to the case of unweighted graphs. The experimental assessment of our diameter approximation algorithm is finally presented in Section 7.

2. Preliminaries

2.1. Graph-Theoretic Concepts

We introduce some notation that will be used throughout the rest of the paper. Let $G = (V, E, w)$ be a connected undirected weighted graph with n nodes, m edges, and a function w which assigns a positive integer weight $w(e)$ to each edge $e \in E$. We make the reasonable assumption that the ratio w_{max}/w_{min} between the maximum (w_{max}) and the minimum (w_{min}) weight is polynomial in

n. The *distance* between two nodes $u, v \in V$, denoted by $d(u, v)$, is defined as the weight of a minimum-weight path between u and v in G. Moreover, we let $\Phi(G)$ denote the *diameter* of the graph defined as the maximum distance between any two nodes. The following definition introduces the concept of k-clustering in weighted graphs.

Definition 1. *For any positive integer $k \leq n$, a* k-clustering *of G is a partition $\mathcal{C} = \{C_1, C_2 \ldots, C_k\}$ of V into k subsets called* clusters*. Each cluster C_i has a distinguished node $c_i \in C_i$ called* center*, and a* radius $r(C_i) = \max_{v \in C_i}\{d(c_i, v)\}$*. The* radius of a k-clustering $\mathcal{C}$ *is* $r(\mathcal{C}) = \max_{1 \leq i \leq k}\{r(C_i)\}$.

2.2. Doubling Dimension

We now introduce the concept of doubling dimension, that a number of works have shown to be useful in relating algorithm's performance to graph properties [23], as well as clustering [25,26] and diversity maximization [27,28]. In this work, for both weighted and unweighted graphs we define the doubling dimension in terms of paths that ignore edge weights.

Definition 2. *For an undirected graph $G = (V, E)$, define the ball of radius R centered at node v as the set of nodes reachable through paths of at most R edges from v. The doubling dimension of G is the smallest integer $b > 0$ such that for any $R > 0$, any ball of radius $2R$ is contained in the union of at most 2^b balls of radius R.*

2.3. The MapReduce Framework

Created to simplify the implementation of distributed algorithms, MapReduce [29] is a very popular framework. Our algorithms adhere to the MapReduce computational model as formalized by Pietracaprina et al. in [30]. A MapReduce algorithm operates on a multiset of key-value pairs, which is transformed through a sequence of *rounds*. In each round, the current multiset X of key-value pairs is transformed into a new multiset Y of pairs by applying the same, given function, called a *reducer*, independently to each subset of pairs of X having the same key. Operationally, each reducer instance is exectued by a given worker of the underlying distributed platform. Therefore, each round entails a data shuffle step, where each key-value pair is routed to the appropriate worker. (See Figure 1 for a pictorial representation of a round). The model is parametrized in terms of two parameters M_L and M_A, and is therefore called MR(M_L, M_A), and an algorithm for the MR(M_L, M_A) model is called an *MR-algorithm*. The two parameters constrain the memory resources available: M_L is the maximum amount of memory locally available to each reducer, whereas M_A is the *aggregate* memory available to the computation. 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 work are sorting and prefix operations. We report a fundamental result from [30].

Theorem 1. *The sorting and prefix operation primitives for inputs of size n can be performed in the MR(M_L, M_A) model with $M_A = \Theta(n)$, using $O(\log_{M_L} n)$ rounds. In particular, if $M_L = O(n^\epsilon)$, for some $\epsilon \in (0, 1)$, the round complexity becomes $O(1)$.*

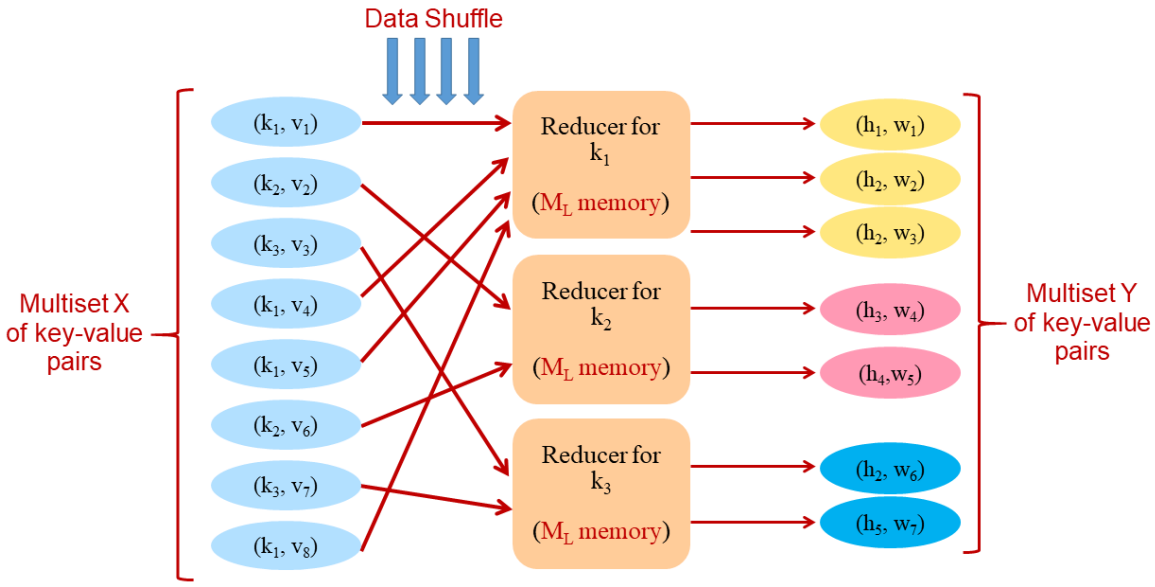

Figure 1. An example of a MapReduce round, transforming a multiset of 8 key-value pairs (over 3 keys) into a multiset of 7 key-value pairs (over 5 keys). At most M_A memory overall can be employed during the execution.

3. Clustering Algorithm

Our algorithm for approximating the diameter of a weighted graph uses, as a crucial building block, a clustering strategy which partitions the graph nodes into subsets, called clusters, of bounded (weighted) radius. The strategy grows the clusters around suitable "seed" nodes, called centers, which are selected progressively in batches throughout the algorithm. The challenge is to perform cluster growth by exploiting parallelism while, at the same time, limiting the weight of the edges involved in each growing step, so to avoid increasing excessively the radius of the clusters, which, in turn, directly influences the quality of the subsequent diameter approximation.

Specifically, we grow clusters in stages, where in each stage a new randomly selected batch of centers is added, and clusters are grown around all centers (old and new ones) for a specified radius, which is a parameter of the clustering algorithm. The probability that a node is selected as center increases as the algorithm progresses. The idea behind such a strategy is to force more clusters to grow in regions of the graph which are either poorly connected or populated by edges of large weight, 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, thus resulting in an increase of the weighted cluster radius which could be too large for our purposes. To tackle this challenge, we use ideas akin to those employed in the Δ-stepping parallel SSSP algorithm proposed in [20].

In what follows, we describe an algorithm that, given a weighted graph $G = (V, E, w)$ and a radius parameter r, computes a *k-clustering* of G with radius slightly larger than r, and k slightly larger than the *minimum* number of centers of any clustering of G of radius at most r. For each node $u \in V$ the algorithm maintains a four-variable *state* (c_u, d_u, g_u, s_u). Variable c_u is the center of the cluster to which u currently belongs to, and d_u is an upper bound to the distance between u and c_u. Variable g_u is the *generation* of the cluster to which u belongs to, that is, the iteration at which the cluster centerd at c_u started growing. Finally, variable s_u is a Boolean flag that marks whether node u is *stable*, that is, if the assignment of node u to the cluster centered at c_u is final. Initially, c_u and g_u are undefined, $d_u = \infty$ and s_u is *false*. A node $u \in V$ is said to be *uncovered* if c_u is undefined, and *covered* otherwise.

For a given graph $G = (V, E, w)$ and integer parameter r, Algorithm RandCluster(G, r), whose pseudocode is given in Algorithm 1, builds the required clustering in $\log n$ iterations (Unless

otherwise specified, throughout the paper all logarithms are taken to the base 2. Also, to avoid cluttering the mathematical derivations with integer-rounding notation, we assume n to be a power of 2, although the results extend straightforwardly to arbitrary n). In each iteration, all clusters are grown for an extra weighted radius of $2r$, with new centers also being added at the beginning of the iteration. More specifically, initially all nodes are uncovered. In iteration i, with $1 \leq i \leq \log n$, the algorithm selects each uncovered node as a new cluster center with probability $2^i/n$ uniformly at random (line 4), and then grows both old and new clusters using a sequence of *growing steps*, defined as follows. In a growing step, for each edge (u, v) of weight $w(u, v) \leq 2r$ (referred to as a *light edge*) such that c_u is defined (line 7), the algorithm checks two conditions: whether v is not stable and whether assigning v to the cluster centered at c_u would result in a radius compatible with the one stipulated for c_u at the current iteration, calculated using the *generation* variable g_u (line 8). If both checks succeed, and if $d_u + w(u, v) < d_v$, then the state of v is updated, assigning it to the cluster centered at c_u, updating the distance, and setting g_u to the generation of c_u (line 10). In case two edges $\{u, v\}$ and $\{u', v\}$ trigger a concurrent state update for v, we let an arbitrary one succeed. When a growing step does not update any state, then all the covered nodes are marked as stable (line 13) and the algorithm proceeds to the next iteration. An example of one execution of RandCluster is described pictorially in Figure 2.

Algorithm 1: RandCluster$(G = (V, E, w), r)$

```
1  foreach u ∈ V parallel do ..... ▷ initialize the state of each node
2      (c_u, d_u, g_u, s_u) ← (nil, ∞, nil, false)
3  for i ← 1 to log n do
4      foreach u ∈ V : c_u = nil parallel do ..... ▷ for each uncovered node
5          With probability 2^i/n do (c_u, d_u, g_u, s_u) ← (u, 0, i, true) ..... ▷ select as a new center
6      repeat ..... ▷ perform the growing steps
7          foreach {u, v} ∈ E : c_u ≠ nil ∧ w(u, v) ≤ 2r parallel do
8              if ¬s_v ∧ d_u + w(u, v) ≤ (i − g_u + 1) · 2r then ..... ▷ check if we should use this edge
9                  if d_u + w(u, v) < d_v then ..... ▷ check if we should update the node
10                     (c_v, d_v, g_v, s_v) ← (c_u, d_u + w(u, v), g_u, false)
11     until No state is updated
12     foreach u ∈ V parallel do ..... ▷ mark all covered nodes as stable
13         if c_u ≠ nil then
14             s_u ← true
15 return {(u, c_u, d_u) : u ∈ V}
```

For a weighted graph $G = (V, E, w)$ and a positive integer x, define ℓ_x as the maximum number of edges in any path in G of total weight at most x. We have:

Theorem 2. *Let $G = (V, E, w)$ be a weighted graph, r be a parameter, and let k^* be the minimum integer such that there exists a k^*-clustering of G of radius at most r. Then, with high probability, Algorithm* RandCluster(G, r) *performs $O(\ell_{2r} \log n)$ growing steps and returns an $O(k^* \log^2 n)$-clustering of G of radius at most $2r \log n$.*

Proof. The bounds on the number of growing steps and on the radius of the clustering follows straightforwardly from the fact that in each of the $\log n$ iterations of the for loop, the radius of the current (partial) clustering can grow by at most an additive term $2r$, hence the growth is attained along paths of distance at most $2r$ which are covered by traversing at most ℓ_{2r} edges in consecutive growing steps.

We now bound the number of centers selected by the algorithm. We need to consider only the case $k^* \leq n/\log^2 n$ since, for larger values of k^*, the claimed bound is trivial. The argument is structured as follows. We divide the iterations of the for loop into two groups based on a suitable index h (determined below as a function of k^* and n). The first group comprises iterations with index $i \leq h$, where the low selection probability ensures that few nodes are selected as centers. The second group comprises iterations with $i > h$, where, as we will prove, the number of uncovered nodes decreases geometrically in such a way to counterbalance the increase in the selection probability, so that, again, not too many centers are selected.

Let $\gamma = 4/\log e$ and define h as the smallest integer such that

$$2^h \geq \gamma \cdot k^* \cdot \log n. \tag{1}$$

Since $k^* \leq n/\log^2 n$, we can safely assume that $1 \leq h < \log n$ and define $t = \log n - h$. For $i \in [0, t)$, we define the event $E_i =$ *At the end of iteration* $h + i$ *of the for loop, at most* $n/2^i$ *nodes are still uncovered*. We now prove that the event $\cap_{i=0}^{t-1} E_i$ occurs with high probability. Observe that

$$\Pr[\cap_{i=0}^{t-1} E_i] = \Pr[E_0] \cdot \prod_{i=1}^{t-1} \Pr[E_i | E_0 \cap \cdots \cap E_{i-1}] = \prod_{i=1}^{t-1} \Pr[E_i | E_0 \cap \cdots \cap E_{i-1}], \tag{2}$$

where the first equality comes from the definition of conditional probability, and the second is due to the fact that $\Pr[E_0]$ clearly holds with probability 1.

Consider an arbitrary $i \in [1, t)$, and assume that the event $E_0 \cap \cdots \cap E_{i-1}$ holds. We prove that, conditioned on this event, E_i holds with high probability. Let V_i be the set of nodes already covered at the beginning of iteration $h + i$. By hypothesis, we have that $|V \setminus V_i| \leq n/2^{i-1}$. We distinguish two cases. If $|V \setminus V_i| \leq n/2^i$, then E_i clearly holds with probability 1. Otherwise, it must be

$$\frac{n}{2^i} \leq |V \setminus V_i| \leq \frac{n}{2^{i-1}}. \tag{3}$$

Let $\mathcal{C}^*$ be a k^*-clustering of G with radius r, whose existence is implied by the definition of k^*, and divide its clusters into two groups: a cluster $C \in \mathcal{C}^*$ is called *small* if $|C \cap (V \setminus V_i)| < |V \setminus V_i|/(2k^*)$, and *large* otherwise. Let $\mathcal{C}^s$ be the set of small clusters. We can bound the number of uncovered vertices contained in the small clusters as follows:

$$\sum_{C \in \mathcal{C}^s} |C \cap (V \setminus V_i)| \quad < \quad \sum_{C \in \mathcal{C}^s} \frac{|V \setminus V_i|}{2k^*} \quad \leq \quad \frac{|V \setminus V_i|}{2}$$

It then follows that, overall, the *large clusters* contain at least half of the nodes that the algorithm has yet to cover at the beginning of iteration $h + i$. We now prove that, with high probability, at least one center will be selected in each large cluster. Consider an arbitrary large cluster C. By Equations (1) and (3), we have that the number of uncovered nodes in C is at least

$$\frac{|V \setminus V_i|}{2k^*} \geq \frac{n \cdot \gamma \cdot \log n}{2^{h+i+1}}.$$

Since in iteration $h + i$, an uncovered node becomes a center with probability $2^{h+i}/n$, the probability that no center is selected from C is at most

$$\left(1 - \frac{2^{h+i}}{n}\right)^{|V \setminus V_i|/2k^*} \leq \left(1 - \frac{2^{h+i}}{n}\right)^{\frac{n\gamma \log_2 n}{2^{h+i+1}}} \leq \exp\left(-\frac{\gamma \log_2 n}{2}\right) = \frac{1}{n^2}$$

By applying the union bound over all large clusters and taking the complement of the probability, it immediately follows that the probability that each large cluster has at least one center being selected

among its uncovered nodes is at least $1 - k^*/n^2$. Now, it is easy to see that in each iteration, in particular, in iteration $h + i$, all nodes at distance at most $2r$ from the newly selected centers will be covered by some cluster (either new or old). Consequently, since the radius of any cluster of $\mathcal{C}$ is $\leq r$, we have that with probability at least $1 - k^*/n^2$, at the end of iteration $h + i$ all nodes in large clusters are covered. Thus, the nodes still uncovered at the of the iteration can belong only to small clusters in $\mathcal{C}^s$, and, from what was observed before, there are at most $|V \setminus V_i|/2 \leq n/2^i$ of them.

By multiplying the probabilities of the $t < \log n$ conditional events $(E_i | E_0 \cap \cdots \cap E_{i-1})$, we conclude that $\cap_{i=1}^{t-1} E_i$ happens with probability at least $(1 - k^*/n^2)^t \geq 1 - 1/n$, where the last bound follows from Bernoulli's inequality and the fact that $k^* \leq n/\log^2 n$.

We can finally bound the number of centers selected during the execution of the algorithm. We do so by partitioning the iterations into three groups based on the iteration index j, and by reasoning on each group as follows:

- Iterations j, with $1 \leq j \leq h$: since, at the beginning of each such iteration j, the number of uncovered nodes is clearly $\leq n$ and the selection probability for each uncovered node is $2^j/n$, by the Chernoff bound we can easily show that $O(2^j \log n)$ centers are selected, with probability at least $1 - 1/n^2$.
- Iterations j, with $h < j < \log n$: by conditioning on the event $\cap_{i=0}^{t-1} E_i$, we have that at the beginning of each such iteration j the number of uncovered nodes does not exceed $n/2^{j-h-1}$ and the selection probability for an uncovered node is $2^j/n$. Since $2^h = \Theta(k^* \log n)$, by the Chernoff bound we can easily show that in iteration j, $O(2^h)$ centers are selected, with probability at least $1 - 1/n^2$.
- Iteration $\log n$: by conditioning on the event $\cap_{i=0}^{t-1} E_i$ we have that at the beginning of iteration $\log n$ the number of uncovered nodes is $O(2^h)$, and since in this iteration the selection probability is 1, all uncovered nodes will be selected as centers.

Putting it all together we have that, by conditioning on $\cap_{i=0}^{t-1} E_i$ and by Bernoulli's inequality, with probability at least $1 - 1/n$ the total number of centers selected by the algorithm is

$$O\left(\sum_{j=1}^{h} 2^j \log n + \sum_{j=h}^{\log n - 1} 2^h + 2^h\right) = O\left(2^h \log n\right) = O\left(k^* \log^2 n\right).$$

The theorem follows. □

Algorithm RANDCLUSTER exhibits the following important property, which will be needed in proving the diameter approximation. Define the *light distance* between two nodes u and v of G as the weight of the shortest path from u to v consisting only of light edges, that is, edges of weight at most $2r$ (note that this distance is not necessarily defined for every pair of nodes). We have the following.

Observation 1. *For a specific execution of* RANDCLUSTER$(G = (V, E, w), r)$*, given a center $c \in V$ selected at iteration i, and a node $v \in V$ at light distance d from c, v cannot be covered by c (i.e., c_v cannot be set to c by the algorithm) in* less *than $\lceil d/2r \rceil$ iterations. Also, by the end of iteration $i + \lceil d/2r \rceil - 1$, v will be covered by some cluster center (possibly c).*

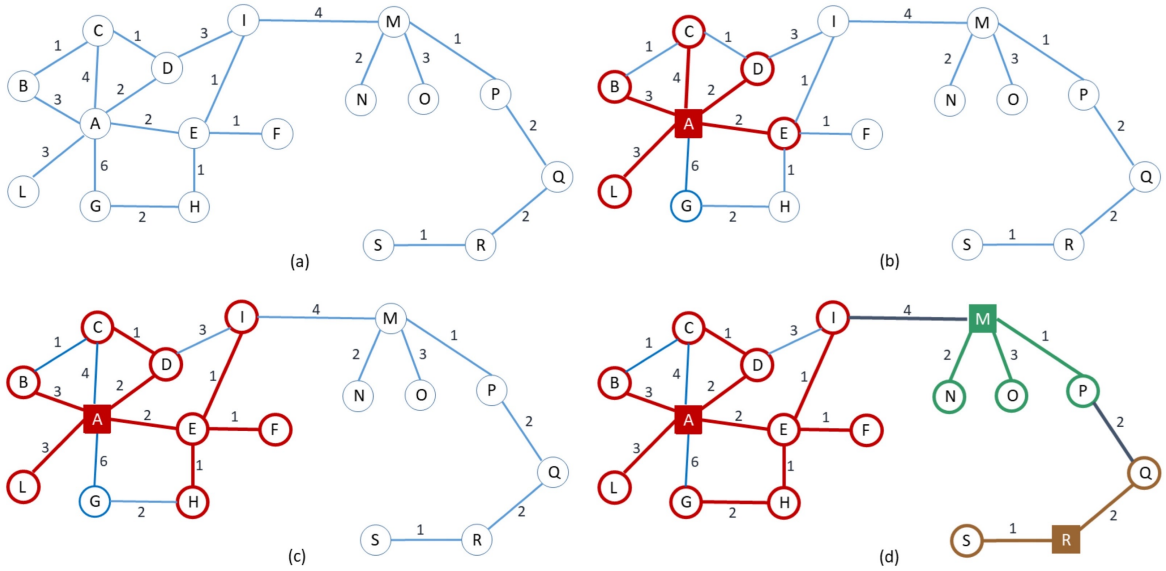

Figure 2. A specific execution of RANDCLUSTER on the weighted graph depicted in (**a**), with $r = 2$. In Iteration $i = 1$ of the **for** loop, only node A is (randomly) selected, and the two growing steps (**b**), (**c**) are performed. The resulting partial cluster centered at A is highlighted in red. In Iteration 2, two further centers, M and R, are selected, and only one growing step (**d**) is executed. The resulting final clusters are highlighted in red, green, and brown.

Proof. Let π be a path of light edges of total weight d between c and v. In each iteration of the algorithm, the cluster centered at c is allowed to grow for a length at most $2r$ along π. Also, an easy induction suffices to show that any node at light distance at most $2rs$ from c on the path will surely be covered (by c or another center) by the end of iteration s. □

An immediate consequence of the above observation is the following. Let $c \in V$ be a center selected at iteration i, and $v \in V$ a node at light distance d from c. No center c' selected at an iteration $j > i$ and at distance at least d from v is able to cover v. The reason is that any such center c' would require the same number of iterations to reach v as c, and by the time it is able to reach v, v would have already been reached by c (or some other center) and marked as stable, which would prevent the reassignment to other centers.

4. Diameter Approximation Strategy

We are now ready to describe our clustering-based strategy to approximate the diameter of a graph G. For any fixed value $r > 0$, we run RANDCLUSTER(G, r) to obtain a clustering $\mathcal{C}$. By Theorem 2, we know that, with high probability, $\mathcal{C}$ has radius $r_{\mathcal{C}} \leq 2r \log n$ and consists of $O(k^* \log^2 n)$ clusters, where k^* is the minimum number of clusters among all the clusterings of G with radius r. Also, we recall that $\mathcal{C}$ is represented by the tuples (u, c_u, d_u), for every $u \in V$, where c_u is the center of the cluster of u, and $d_u \leq r_{\mathcal{C}}$ is an upper bound to $d(u, c_u)$.

As in [16], we define a weighted auxiliary graph $G_{\mathcal{C}}$ associated with $\mathcal{C}$ where the nodes correspond to the cluster centers and, for each edge (u, v) of G with $c_u \neq c_v$, there is an edge (c_u, c_v) in $G_{\mathcal{C}}$ of weight $w(u, v) + d_u + d_v$. In case of multiple edges between clusters, we retain only the one yielding the minimum weight. (See Figure 3 for the auxiliary graph associated to the clustering described in Figure 2).

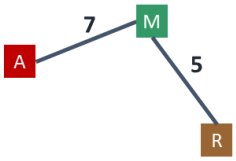

Figure 3. Auxiliary graph G_C associated to the clustering C of radius $r_C = 5$ described in Figure 2. Observe that the diameter $\Phi(G)$ of the original graph G (Figure 2a) is realized by the path $\langle L, A, E, I, M, P, Q, R, S \rangle$ of weight 16, while the algorithm returns the value $\Phi_{\text{approx}}(G) = \Phi(G_C) + 2 \cdot r_C = 12 + 2 \times 5 = 22$.

Let $\Phi(G)$ (resp., $\Phi(G_C)$) be the diameter of G (resp., G_C). We approximate $\Phi(G)$ with

$$\Phi_{\text{approx}}(G) = \Phi(G_C) + 2 \cdot r_C \tag{4}$$

It is easy to see that $\Phi_{\text{approx}}(G) \geq \Phi(G)$. The following theorem provides an upper bound to $\Phi_{\text{approx}}(G)$.

Theorem 3. *If* $r = O(\Phi(G))$, *then*

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

with high probability.

Proof. Since $r_C \leq 2r \log n$, and, by hypothesis, $r = O(\Phi(G))$, it is sufficient to show that $\Phi(G_C) = O(\Phi(G) \log^3 n)$. To this purpose, let us fix an arbitrary pair of distinct clusters C_1, C_2 and a shortest path π in G between their centers of (minimum) weight w_π . Let π_C be the (not necessarily simple) path of clusters in G_C traversed by π, and observe that, by shortcutting all cycles in π_C, we have that the distance between the nodes associated to the centers of C_1 and C_2 in G_C is upper bounded by w_π plus twice the sum of the radii of the distinct clusters encountered by π_C. We now show that, with probability at least $1 - 1/n^3$, this latter sum is $O(\Phi(G) \log^3 n)$. Then the theorem will immediately follow by applying the union bound over all pairs of distinct clusters. We distinguish the following two cases.

Case 1. Suppose that $2r > \Phi(G)$ (hence, $r = \Theta(\Phi(G))$, since we assumed $r = O(\Phi(G))$) and let $i \geq 1$ be the index of the first iteration where some centers are selected. By applying a standard Chernoff bound ([31], Corollary 4.6) we can easily show that $O(\log n)$ centers are selected in Iteration i, with probability at least $1 - 1/n^3$. Moreover, since all nodes in G are at light distance at most $\Phi(G)$ from the selected centers, we have that, by virtue of Observation 1, all nodes will be covered by the end of Iteration i. Therefore, with probability at least $1 - 1/n^3$, π_C contains $O(\log n)$ clusters and it easily follows that its weight is $O(w_\pi + r \log n) = O(w_\pi + \Phi(G) \log n) = O(\Phi(G) \log^3 n)$.

Case 2. Suppose now that $2r \leq \Phi(G)$. We show that, with high probability, at most $O(\lceil w_\pi / r \rceil \log^2 n)$ clusters intersect π (i.e., contain nodes of π). Observe that π can be divided into $O(\lceil w_\pi / r \rceil)$ subpaths, where each subpath is either an edge of weight $> r$ or a segment of weight $\leq r$. We next show that the nodes of each of the latter segments belong to $O(\log^2 n)$ clusters, with high probability. Consider one such segment S. Clearly, all clusters containing nodes of S must have their centers at light distance at most $2r \log n$ from S (i.e., light distance at most $2r \log n$ from the closest node of S).

For $1 \leq j \leq 2 \log n + 1$, let $C(S, j)$ be the set of nodes whose light distance from S is between $(j-1)r$ and $j \cdot r - 1$, 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 $1 \leq j \leq 2 \log n + 1$, and let i_j

be the first iteration of the for loop of RANDCLUSTER in which some center is selected from $C(S,j)$. By Observation 1, $\lceil((j+1)r-1)/(2r)\rceil$ 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 $\lceil(j-1)r/(2r)\rceil$ iterations to touch the segment. Hence, we have that no center selected from $C(S,j)$ at Iteration i_j+2 or higher can cover a node of S, since by the time it reaches S, the nodes of the segment are already covered and stable.

We now show that the number of centers being selected from $C(S,j)$ in Iterations i_j and i_j+1 is $O(\log n)$. For a suitable constant c, we distinguish two cases.

1. $|C(S,j)| \leq c \log n$. Then the bound trivially holds.
2. $|C(S,j)| > c \log n$. Let h be an index such that in Iteration h the center selection probability $p_h = 2^h/n$ is such that $p_h \cdot |C(S,j)| = c \log n$. Therefore, in Iteration $h+1$ the expected number of centers selected from $C(S,j)$ is $2c \log n$. By applying the Chernoff bound, we can prove that, with high probability, the number of centers selected from $C(S,j)$ in Iteration $h+1$ is $\Theta(\log n)$ and that the number of centers selected in any iteration $\leq h+1$ is $O(\log n)$. Furthermore, observe that, with high probability, $i_j \leq h$: indeed, if no center is selected in iterations up to the $h-1$, with high probability at least one is chosen at iteration h, since we just proved that $\Theta(\log n)$ are selected in iteration h.

Therefore, we have that the total number of centers selected from $C(S,j)$ in Iterations i_j and i_j+1 (and thus the only centers from $C(S,j)$ which may cover nodes of S) is $O(\log n)$. Overall, considering all $C(S,j)$'s, the nodes of segment S will belong to $O(\log^2 n)$ clusters, with high probability. By applying the union bound over all segments of π, we have that $O(\lceil w_\pi/r\rceil \log^2 n)$ clusters intersect π, with high probability. It is easy to see that by suitably selecting the constants in all of the applications of the Chernoff bound, the high probability can be made at least $1-1/n^3$. Therefore, with such probability we have:

$$\begin{aligned}\pi_C &= O\left(w_\pi + r_{\mathcal{C}} \left\lceil \frac{w_\pi}{r} \right\rceil \log^2 n\right) \\ &= O\left(\Phi(G) + r \left\lceil \frac{\Phi(G)}{r} \right\rceil \log^3 n\right) \\ &= O\left(\Phi(G) \log^3 n\right)\end{aligned}$$

where last equality follows from the hypothesis $r = O(\Phi(G))$. □

An important remark regarding the above result is in order at this point. While the upper bound on the approximation ratio does not depend on the value of parameter r (as long as $r \in O(\Phi(G))$), this value affects the efficiency of the strategy. In broad terms, a larger r yields a clustering $\mathcal{C}$ with fewer clusters, hence a smaller auxiliary graph $G_{\mathcal{C}}$ whose diameter can be computed efficiently with limited main memory, but requires a larger number of growing steps in the execution of RANDCLUSTER, hence a larger number of synchronizations in a distributed execution. In the next section we will show how to make a judicious choice of r which strikes a suitable space-round tradeoff in MapReduce.

5. Implementation in the MapReduce Model

In this section we describe an algorithm for diameter approximation in MapReduce, which uses sublinear local memory and linear aggregate memory and, for an important class of graphs which includes well-known instances, features a round complexity asymptotically smaller than the one required to obtain a 2-approximation through the state-of-the-art SSSP algorithm by [20]. The algorithm runs the clustering-based strategy presented in the previous section for geometrically increasing guesses of the parameter r, until a suitable guess is identified which ensures that the computation can be. carried out within a given local memory budget.

Recall that the diameter-approximation strategy presented in the previous section is based on the use of Algorithm RANDCLUSTER to compute a clustering of the graph, and that, in turn, RANDCLUSTER entails the execution of sequences of growing steps. A growing step (lines 7÷10 of Algorithm 1) can be implemented in MapReduce as follows. We represent each node by the key-value pair $((u,0),(c_u,d_u,g_u,s_u))$ and each edge $(u,v) \in E$ by the two key-value pairs $((u,1),(v,w(u,v)))$ and $((v,1),(u,w(u,v)))$. We then sort all key-value pairs by key so that for every node u there will be a segment in the sorted sequence starting from $((u,0),(c_u,d_u,g_u,s_u))$ and followed by the pairs associated to its incident edges u. At this point, a segmented prefix operation can be used to create, for each edge (u,v) two pairs $((u,v);(w(u,v),c_u,d_u,g_u,s_u))$ and $((u,v);(w(u,v),c_v,d_v,g_v,s_v))$, which are then gathered by a single reducer to perform the possible update of (c_u,d_u,g_u,s_u) or (c_v,d_v,g_v,s_v) as specified by lines 8÷10 of Algorithm 1. A final prefix operation can be employed to determine the final value of each 4-tuple (c_u,d_u,g_u,s_u), by selecting the update with minimum d_u.

By Theorem 1, each sorting and prefix operation can be implemented in $\mathrm{MR}(M_L,M_A)$ with $O(\log_{M_L} n)$ rounds and linear aggregate memory. The MapReduce implementation of all other operations specified by Algorithm 1 is simple and can be accomplished within the same memory and round complexity bounds as those required by the growing steps. Recall that for a weighted graph G, we defined ℓ_x to be the maximum number of edges in any path of total weight at most x. By combining the above discussion with the results of Theorems 1 and 2, we have:

Lemma 1. *Let G be a connected weighted graph with n nodes and m edges. On the $MR(M_L,M_A)$ model algorithm* RANDCLUSTER(G,r) *can be implemented in*

$$O\left(\ell_{2r} \cdot \log n \cdot \log_{M_L} n\right)$$

rounds, with $M_A = \Theta(m)$. In particular, if $M_L = \Omega(n^\epsilon)$, for some constant $\epsilon > 0$, the number of rounds becomes $O(\ell_{2r} \log n)$.

We are now ready to describe in detail our MapReduce algorithm for diameter approximation. Consider an n-node connected weighted graph G, and let w_{min} be the minimum edge weight. For a fixed parameter $\epsilon \in (0,1)$, the algorihm runs RANDCLUSTER(G,r) for geometrically increasing values of r, namely $r = w_{min} \cdot 2^i$, with $i = 0,1,\ldots$, until the returned clustering $\mathcal{C}$ is such that the corresponding auxiliary graph $G_\mathcal{C}$ has size (i.e., number of nodes plus number of edges) $O(n^\epsilon)$. At this point, the diameter $\Phi(G_\mathcal{C})$ of $G_\mathcal{C}$ is computed using a single reducer, and the value $\Phi_{\text{approx}}(G) = \Phi(G_\mathcal{C}) + 2 \cdot r_\mathcal{C}$, where $r_\mathcal{C}$ is the radius of $\mathcal{C}$, is returned as an approximation to the true diameter $\Phi(G)$. We have:

Theorem 4. *Let G be connected weighted graph with n nodes, m edges, and weighted diameter $\Phi(G)$. Also, let $\epsilon \in (0,1)$ be an arbitrarily fixed constant and define r^*_ϵ as the minimum radius of any clustering of G with at most $n^{\epsilon/2}/\log^2 n$ clusters. With high probability, the above algorithm returns an estimate $\Phi_{approx}(G)$ such that $\Phi(G) \leq \Phi_{approx}(G) = O(\Phi(G)\log^3 n)$, and can be implemented in the $MR(M_L,M_A)$ model using*

$$O\left(\ell_{4r^*_\epsilon} \log^2 n\right)$$

rounds, with $M_L = \Theta(n^\epsilon)$ and $M_A = \Theta(m)$.

Proof. Let $\bar{r}$ be the first radius in the sequence $\{w_{min} \cdot 2^i : i \geq 0\}$ such that $\bar{r} \geq r^*_\epsilon$. Define k^*_ϵ (resp., $\bar{k}$) as the minimum number of clusters in a clustering of G with radius r^*_ϵ (resp., $\bar{r}$). It is straightforward to see that $\bar{k} \leq k^*_\epsilon$. Since, by hypothesis, $k^*_\epsilon \leq n^{\epsilon/2}/\log^2 n$, we have that $\bar{k} \leq n^{\epsilon/2}/\log^2 n$. By Theorem 2, with high probability RANDCLUSTER$(G,\bar{r})$ returns a clustering with $O(\bar{k}\log^2 n) = O(n^{\epsilon/2})$ clusters whose corresponding auxiliary graph has size $O(n^\epsilon)$. Therefore, with high probability, the MapReduce algorithm

will complete the computation of the approximate diameter after an invocation of RANDCLUSTER with radius at most $\bar{r}$.

By Lemma 1, each invocation of RANDCLUSTER(G, r) can be implemented in MapReduce using $O(\ell_{2r} \log n)$ rounds with $M_L = \Theta(n^\epsilon)$ and $M_A = \Theta(m)$. In particular, the round complexity of the last invocation (with radius at most $\bar{r}$) will upper bound the one of any previous invocation. Considering the fact that $\bar{r} \leq 2r^*_\epsilon$, we have that the round complexity of each invocation is $O(\ell_{4r^*_\epsilon} \log n)$. Note that $r^*_\epsilon \leq \Phi(G)$, therefore the maximum number of values r for which RANDCLUSTER(G, r) is executed is $O(\log(\Phi(G)/w_{min}))$. Since $\Phi(G) \leq n \cdot w_{max}$, where w_{max} is the maximum edge weight, and w_{max}/w_{min} is polynomial in n, as we assumed at the beginning of Section 3, we have that the maximum number of invocations of RANDCLUSTER(G, r) is $O(\log n)$. Therefore, the aggregated round complexity of all the invocations of RANDCLUSTER up to the one finding a suitably small clustering is $O(\ell_{4r^*_\epsilon} \log^2 n)$. Moreover, it is easy to see that the computation of the auxiliary graph after each invocation of RANDCLUSTER and the computation of the diameter after the last invocation, which is performed on an auxiliay graph of size $O(n^\epsilon)$, does not affect neither the asymptotic round complexity nor the memory requirements.

Finally, the bound on the approximation follows directly from Theorem 3, while the bounds on M_L and M_A derive from Lemma 1 and from the previous discussion. □

We observe that the term $\ell_{4r^*_\epsilon}$ which appears in the round complexity of the algorithm stated in the above theorem, depends on the graph topology. In what follows, we will prove an upper bound to $\ell_{4r^*_\epsilon}$, hence an upper bound to the round complexity, in terms of the *doubling dimension* of G, a topological notion which was reviewed in Section 2.2. To this purpose, we first need the following technical lemma.

Lemma 2. *Let G be a graph with n nodes and maximum degree d. If we remove each edge independently with probability at least $1 - 1/d$, then, with high probability, the graph becomes disconnected and each connected component has size $O(\ln n)$.*

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 $c \geq 24d(d-1)$, 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 in G' 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 [32] $\{Z_i,\ i \geq 0\}$, 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 $i \geq 0$, the distribution of Y_i is stochastically upper bounded by the distribution of Z_i. For a branching process with i.i.d. offspring distributions as the one we are considering, the *total progeny* can be bound as follows ([33], Theorem 3.13) [34]

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

where the X_i are independent random variables with the same distribution as the offspring distribution of the branching process, which in this case is a Binomial with parameters $(d-1)$ and p.

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 $\geq k$ is bounded by

$$\begin{aligned} Pr(\sum_{i\geq 0} Y_i \geq k) &\leq Pr(\sum_{i\geq 0} Z_i \geq k) \\ &\leq Pr(\sum_{i=1}^{k} X_i \geq k-1) \\ &\leq \exp\left(-\frac{k}{12d(d-1)}\right) \end{aligned}$$

If $k = c \ln n$, with $c \geq 24d(d-1)$, then we have

$$Pr(\sum_{i\geq 0} Y_i \geq k) \leq \frac{1}{n^2}$$

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

On weighted graphs, the result of Lemma 2 immediately implies the following observation.

Observation 2. *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 larger than or equal to $(w_{min} + w_{max})/d$, 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 number of consecutive edges with weight less than $(w_{min} + w_{max})/d$ is $O(\log n)$, with high probability.*

Observation 2 allows us to derive the following corollary of Theorem 4. Recall that $\Phi(G)$ denotes the weighted diameter of G. With $\Psi(G)$, instead, we denote its unweighted diameter.

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 $w_{max}/w_{min} = O(poly(n))$. Also, let $\epsilon \in (0,1)$ be an arbitrarily fixed constant. With high probability, an estimate $\Phi_{approx}(G)$ such that $\Phi(G) \leq \Phi_{approx}(G) = O(\Phi(G) \log^3 n)$, can be computed in*

$$O\left(\left\lceil \frac{\Psi(G)}{n^{\epsilon/(2D)}} \right\rceil (\log n)^{3+2/D}\right)$$

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

Proof. Given the result of Theorem 4, in order to prove the corollary we only need to show the bound on the number of rounds. From the statement of Theorem 4, recall that, with high probability, the number of rounds is $O(\ell_{4r_\epsilon^*} \log^2 n)$, where r_ϵ^* is the minimum radius of any clustering of G with at most $n^{\epsilon/2}/\log^2 n$ clusters.

By iterating the definition of doubling dimension starting from a single ball of unweighted radius $\Psi(G)$ containing the whole graph, we can decompose G into $n^{\epsilon/2}/\log^2 n$ disjoint clusters of unweighted radius $\psi = O(\lceil \Psi(G) \log^2 n / n^{\epsilon/2})^{1/D} \rceil)$. Since w_{max} is the maximum edge weight, we have that $r_\epsilon^* \leq \psi \cdot w_{max}$. We will now give an upper bound on $\ell_{4r_\epsilon^*}$. By Lemma 2 and Observation 2, we have that by removing all edges of weight $\geq (w_{min} + w_{max})/d$, 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 $\geq (w_{min} + w_{max})/d = \Omega(w_{max})$ every $O(\log n)$ nodes. This implies that a path of weight at most $4r_\epsilon^* = O(\lceil \Psi(G)(\log^2 n / n^{\epsilon/2})^{1/D} \rceil w_{max})$ has at most $\ell_{4r_\epsilon^*} = O(\lceil \Psi(G)(\log^2 n / n^{\epsilon/2})^{1/D} \rceil \log n)$ edges, and the corollary follows. □

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. In contrast, 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 [20] implies that under the linear-space constraint a natural MapReduce implementation of Δ-stepping requires $\Omega(\Psi(G))$ rounds. In the next section, we will assess experimentally the difference in performance between our algorithm and Δ-stepping.

6. 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 growing step is very efficient, since once a node is covered for the first time, it is always with the minimum distance from its center, so it will not be further updated. (In fact, in the unweighted case the growing step is conceptually equivalent to one step of a BFS-like expansion). This results in an improvement in the round complexity by a doubly logarithmic factor, as stated in the following corollary to Theorem 4.

Corollary 2. *Let G be a connected unweighted graph with n nodes, m edges, and doubling dimension D. Also, let $\epsilon \in (0,1)$ be an arbitrarily fixed constant. With high probability, an estimate $\Phi_{approx}(G)$ such that $\Phi(G) \leq \Phi_{approx}(G) = O(\Phi(G)\log^3 n)$ can be computed in*

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

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

Proof. The approximation bound is as stated in Theorem 4. For what concerns the round complexity, we first observe that an unweighted graph can be regarded as a weighted graph with unit weights. On such a graph, we have $\ell_x = x$, for every positive integer x. Thus, by Lemma 1, each execution of RandCluster(G, r) can be implemented in $O(r \log n)$ rounds in the MR(M_L, M_A) model with $M_L = O(n^\epsilon)$ and $M_A = \Theta(m)$. From the statement of Theorem 4, recall that r^*_ϵ is the minimum radius of any clustering of G with at most $n^{\epsilon/2}/\log^2 n$ clusters. As argued in the proof of that theorem, the round complexity of the algorithm is dominated by the executions of RandCluster, which are performed for geometrically increasing values of r up to a value at most $2r^*_\epsilon$, thus yielding an overall round complexity of $O(r^*_\epsilon \log n)$.

By reasoning as in the proof of Corollary 1 we can show that G can be decomposed into $n^{\epsilon/2}/\log^2 n$ disjoint of radius $\phi = O(\left\lceil \Phi(G)(\log^2 n/n^{\epsilon/2})^{1/D} \right\rceil)$, which thus provides an upper bound to r^*_ϵ. The corollary follows. □

In the case of unweighted graphs, the most natural competitor for approximating the diameter is a simple BFS, instead of Δ-stepping. However, the same considerations made at the end of the previous section apply, since any natural MapReduce implementation of BFS also requires $\Theta(\Phi(G))$ rounds, similarly to Δ-stepping. Another family of competitors is represented by neighbourhood function-based algorithms [10,12,15], which we reviewed in Section 1.1. These algorithms, like the BFS, require $\Theta(\Phi(G))$ rounds, and are therefore outperformed by our approach on graphs with constant doubling dimension.

As a final remark on our theoretical results for both the weighted and the unweighted cases, we observe that the $O(\log^3 n)$ bound on the ratio between the diameter returned by our strategy and the

exact diameter may appear rather weak. However, these theoretical bounds are the result of a number of worst-case approximations which, we conjecture, are unlikely to occur in practice. In fact, in the experiments reported in Section 7.1, our strategy exhibited an accuracy very close to the one of the 2-approximation based on Δ-stepping. Although to a lesser extent, the theoretical bounds on the round complexities of the MapReduce implementation may suffer from a similar slackness. An interesting open problem is to perform a tighter analysis of our strategy, at least for specific classes of graphs.

7. Experiments

We implemented our algorithms with Rust 1.41.0 (based on LLVM 9.0) on top of Timely Dataflow (https://github.com/TimelyDataflow/timelydataflow) (compiled in *release* mode, with all available optimizations activated). Our implementation, which is publicly available (https://github.com/Cecca/diameter-flow/), has been run on a cluster of 12 nodes, each equipped with a 4 core I7-950 processor clocked at a maximum frequency of 3.07 GHz and 18 GB RAM, connected by a 10 Gbit Ethernet network. We configured Timely Dataflow to use 4 threads per machine.

We implemented the diameter-approximation algorithm as described in Section 5 (referred to as ClusterDiameter in what follows), by running several instances of RandCluster, each parametrized by a different value of r, until a clustering is found whose corresponding auxiliary graph fits into the memory of a single machine. In order to speed up the computation of the diameter of the auxiliary graph, rather than executing the classical exact algorithm based on all-pairs shortest paths, we run two instances of Dijkstra's algorithm: the first from an arbitrary node, the second from the farthest reachable node, reporting the largest distance found in the process. We verified that this procedure, albeit providing only a 2-approximation in theory, always finds a very close approximation to the diameter in practice, in line with the findings of [6], while being much faster than the exact diameter computation. The only other deviation from the theoretical algorithm concerns the initial value of r, which is set to the average edge weight (rather than w_{min}) in order to save on the number of guesses.

We compared the performance of our algorithm against the 2-approximation algorithm based on Δ-stepping [20], which we implemented in the same Rust/Timely Dataflow framework. In what follows, we will refer to this implementation as DeltaStepping. For this algorithm, we tested several values of Δ, including fractions and multiples of the average edge weight, reporting, in each experiment, the best result obtained over all tested values of Δ.

We experimented on three graphs: a web graph (`sk-2005`), a social network (`twitter-2010`), (both downloaded from the WebGraph collection [35,36]: http://law.di.unimi.it/datasets.php) and a road network (`USA`, downloaded from http://users.diag.uniroma1.it/challenge9/download.shtml). Since both `sk-2005` and `twitter-2010` are originally unweighted graphs, we generated a weighted version of these graphs by assigning to each edge a random integer weight between 1 and n, with n being the number of nodes. Also, since the road network `USA`, on the other hand, is small enough to fit in the memory of a single machine, we inflated it by generating the cartesian product of the network with a linear array of S nodes and unit edge weights, for a suitable scale parameter S. In the reported experiments we used $S = 5, 10$. The rationale behind the use of the cartesian product was to generate a larger network with a topology similar to the original one. For each dataset, only the largest connected component was used in the experiments. Table 1 summarizes the main characteristics of the (largest connected components of the) above benchmark datasets. Being a web and a social graph, `sk-2005` and `twitter-2010` have a small *unweighted* diameter (in the order of the tens of edges), which allows information about distances to propagate along edges in a small number of rounds. Conversely, `USA` has a very large unweighted diameter (in the order of tens of thousands), thus requiring a potentially very large number of rounds to propagate information about distances. We point out that a single machine of our cluster is able to handle weighted graphs of up to around half a billion edges using Dijkstra's algorithm. Thus, the largest among our datasets (`sk-2005`, `twitter-2010`, and `USA` with $S = 10$) cannot be handled by a single machine, hence they provide a good testbed to check the effectiveness of a distributed approach.

Our experiments aim at answering the following questions:

1. How does ClusterDiameter compare with DeltaStepping? (Section 7.1)
2. How does the guessing of the radius in ClusterDiameter influence performance? (Section 7.2)
3. How does the algorithm scale with the size of the graph? (Section 7.3)
4. How does the algorithm scale with the number of machines employed? (Section 7.4)

Table 1. Datasets used in the experimental evaluation.

Dataset	Nodes	Edges	Max. Weight	Avg. Weight
`sk-2005`	50 066 547	1 806 103 236	50 066 547	25 033 273
`twitter-2010`	41 582 394	1 200 658 436	41 582 394	20 791 197
`USA`	23 947 347	28 854 312	368 855	2 950
`USA-x5`	119 736 735	240 060 948	368 855	2 950
`USA-x10`	239 473 470	504 069 243	368 855	2 950

7.1. Comparison with the State of the Art

Figure 4 reports the comparison between ClusterDiameter and DeltaStepping on the benchmark datasets. The left plot shows running times, the right plot shows the diameter.

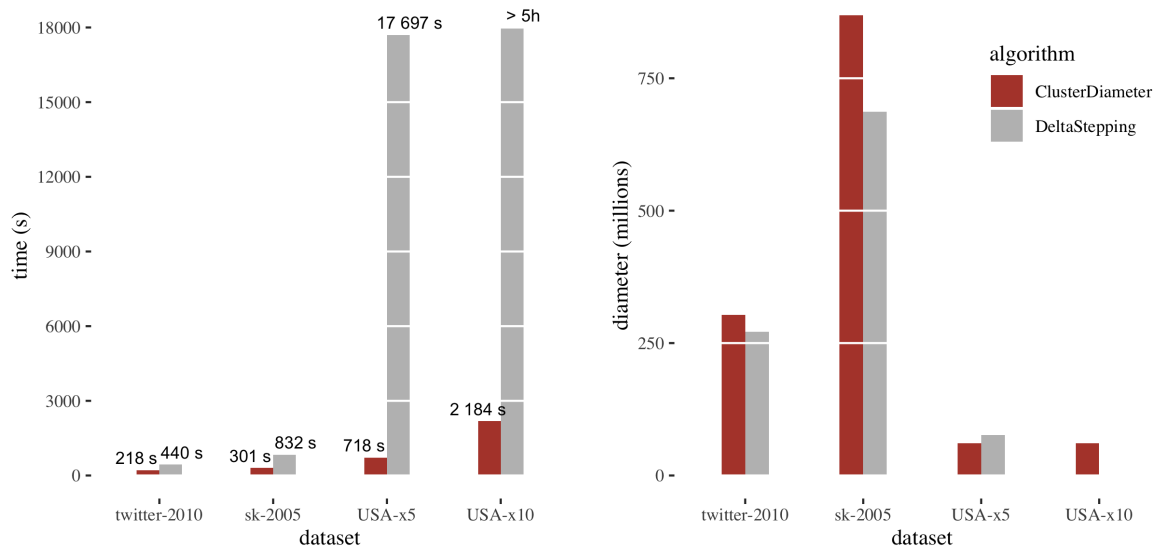

Figure 4. Comparison between DeltaStepping and ClusterDiameter: running time (**left**) and diameter (**right**). On `USA-x10` DeltaStepping timed out after five hours.

Considering the running times, we observe that on `sk-2005` and `twitter-2010` the running times are comparable, with ClusterDiameter being approximately twice as fast. Recall that DeltaStepping performs a number of parallel rounds linear in the unweighted diameter of the graph, whereas ClusterDiameter employs a number of rounds typically sublinear in this metric. However, on these two graphs the unweighted diameter is too small to make the difference in performance evident. On the other hand, `USA` has a very large unweighted diameter, and, as expected ClusterDiameter is much faster than DeltaStepping on this instance.

As for the diameter returned by the two algorithms, we have that both yield similar results, with ClusterDiameter reporting a slightly larger diameter on `twitter-2010` and `sk-2005`. Note that DeltaStepping provides a 2-approximation, whereas ClusterDiameter provides a polylog approximation in theory. The fact that the two algorithms actually give similar results shows that ClusterDiameter is in practice much better than what is predicted by the theory. We notice that,

since the very slow execution of DELTASTEPPING on `USA-x10` was stopped after 5 hours, no diameter approximation was obtained in this case.

7.2. Behaviour of the Guessing of the Radius

To test the behaviour of the guessing of the radius, we ran CLUSTERDIAMETER on `USA` with an initial guess of one tenth the average edge weight, increasing the guess by a factor ten in each step, and stopping at a thousand times the average edge weight. For each guessing step, we recorded the running time, along with the size of the auxiliary graph obtained with that radius guess, and the diameter approximation computed from the auxiliary graph. We remark that, in all experiments, the time required to compute the diameter on the auxiliary graph (not reported) is negligible with respect to the time required by the clustering phase.

Figure 5 reports the results of this experiment. First and foremost we note that, as expected due to exponential growth of the radius parameter in the search, the duration of each guessing step is longer than the cumulative sum of the durations of the previous steps. Also as expected, the size of the auxiliary graph decreases with the increase of the radius used to build the clustering. As for the approximate diameter computed from the different auxiliary graphs, intuition suggests that the auxiliary graph built from a very fine clustering is very similar to the input graph, hence will feature a very similar diameter; whereas a coarser clustering will produce an auxiliary graph which loses information, hence leading to a slightly larger approximate diameter. However, we observe that the diameter approximation worsens only slightly as the size of the auxiliary graph decreases (by less than 12% from the minimum to the maximum guessed radius). This experiment provides evidence that our strategy is indeed able to return good diameter estimates even on platforms where each worker has very limited local memory, although, on such platforms, the construction of a small auxiliary graph requires a large radius and is thus expected to feature large round complexity, as reflected by the increase in running time reported in the leftmost graph of Figure 5.

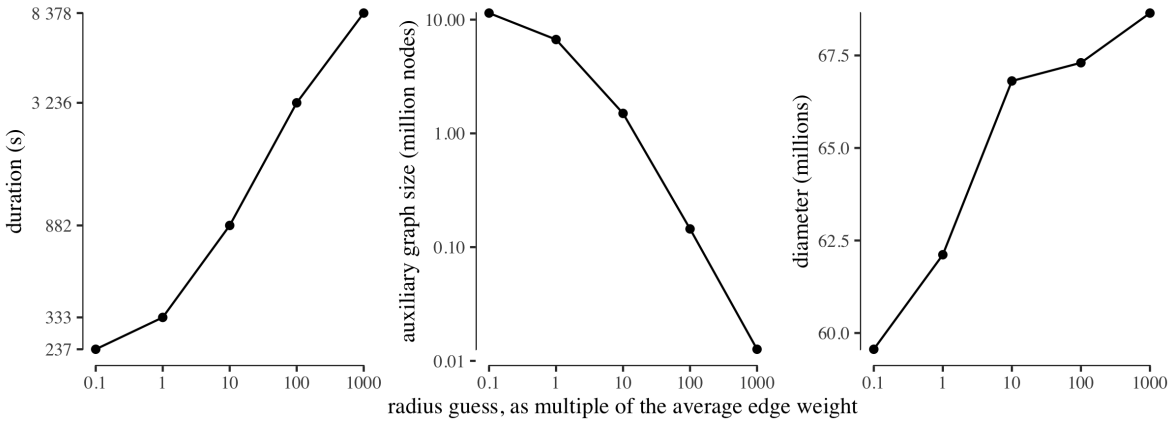

Figure 5. Time employed to build clusterings of `USA` with different radius guesses (**left**), size of the auxiliary graph for each guess (**center**), and diameter approximation obtained from each clustering (**right**). All axes are in logarithmic scale, except for the y axis of the rightmost plot.

7.3. Scalability

The goal of the scalability experiments is twofold. One the one hand, we want to assess the overhead incurred by a distributed algorithm over a sequential one, in case the graph is small enough so that the latter can be run. On the other hand, we want to compare the scalability with respect to the graph size of CLUSTERDIAMETER versus DELTASTEPPING. To this purpose, we used as benchmarks the original `USA` network and the two inflated versions with $S = 5, 10$. We note that both the original `USA` network and the inflated version with $S = 5$ fit in the memory of a single machine. We did not experiment with the `sk-2005` and `twitter-2010` graphs, since it was unclear how to downscale them

to fit the memory of a single machine, while preserving the topological properties. As a fast sequential baseline, we consider a single run of Dijkstra's algorithms from an arbitrary node, which gives a 2-approximation to the diameter.

The results of the experiments are reported in Figure 6. On the original `USA` network, as well as on its fivefold scaled up version, the sequential approach is considerably faster than both ClusterDiameter and DeltaStepping, suggesting that when the graph can fit into main memory it is hard to beat a simple algorithm with low overhead. However, when the graph no longer fits into main memory, as is the case of `USA-x10`, the simple sequential approach is no longer viable. Considering DeltaStepping, it is clearly slower than ClusterDiameter, even on small inputs, failing to meet the five hours timeout on `USA-x10`, On the other hand, ClusterDiameter exhibits a good scalability in the size of the graph, suggesting that it can handle much larger instances, even in the case of very sparse graphs with very large unweighted diameters.

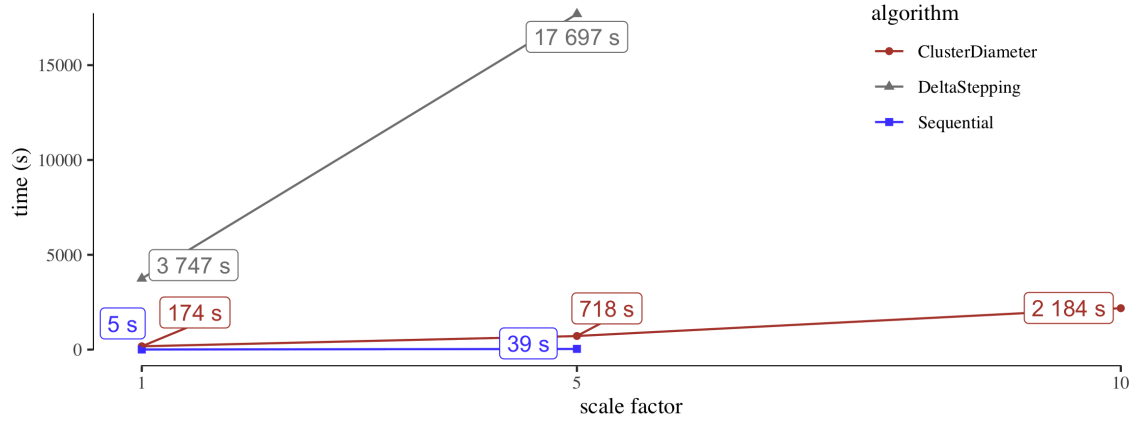

Figure 6. Scalability with respect to the size of the graph for algorithms ClusterDiameter DeltaStepping, and a Sequential approximation obtained running Dijkstra's algorithm. The x axis reports the *scale* of the graph, that is the number of layers used to inflate the original dataset. For `USA-x10` (rightmost point), the sequential algorithm could not load the entire graph in memory, whereas DeltaStepping timed out after five hours.

7.4. Strong Scaling

In this section we investigate the *strong scaling* properties of ClusterDiameter, varying the number of machines employed while maintaining the input instance fixed. The results are reported in Figure 7, where, for each graph, we report results starting from the minimum number of machines for which we were able to complete the experiment successfully, avoiding running out of memory.

We observe that, in general, our implementation of ClusterDiameter scales gracefully with the number of machines employed. It is worth discussing some phenomena that can be observed in the plot, and that are due to peculiarities of our hardware experimental platform. Specifically, on `twitter-2010` we observe a very large difference between the running times on 4 and 6 machines. The reason for this behaviour is that for this dataset, on our system, when using 4 machines our implementation incurs a heavy use of swap space on disk when building the clustering. Interestingly, the slightly larger graph `sk-2005` does not trigger this effect. This difference is due to the fact that `twitter-2010` exhibits faster expansion (the median number of hops between nodes is 5 on `twitter-2010`, and 15 on `sk-2005`), therefore on `twitter-2010` the implementation needs to allocate much larger buffers to store messages in each round. From 6 machines onwards, the scaling pattern is similar to the other datasets, as detailed in the smaller inset plot. Similarly, on `USA-x5`, the scaling is very smooth except between 4 and 6 machines, where the improvement in performance is more marked. The reason again is that on 2 and 4 machines the implementation makes heavy use of swap space on disk during the construction of the auxiliary graph.

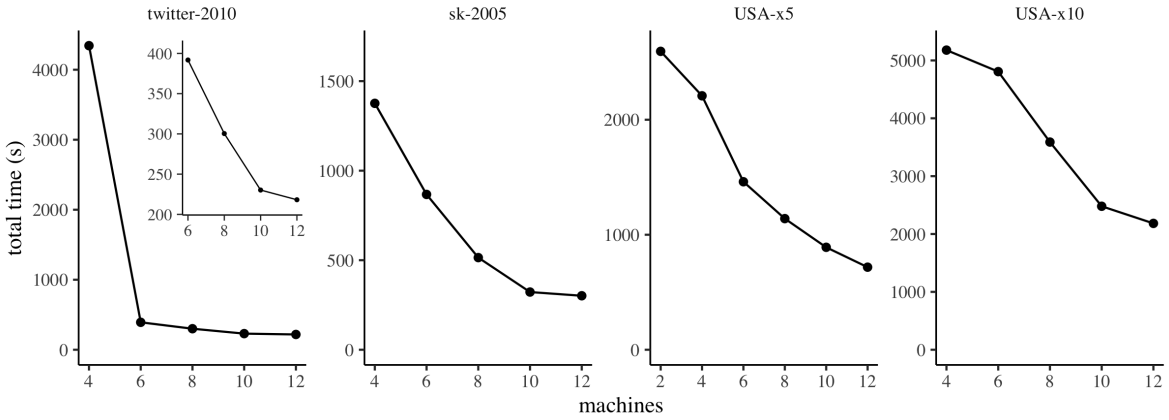

Figure 7. Strong scaling of CLUSTERDIAMETER. For `twitter-2010`, the smaller plot details the strong scaling between 6 and 12 machines, for readability.

We remark that the experimental evaluation reported in this section focus exclusively on weighted graphs. A set of experiments, omitted for the sake of brevity, has shown that, on unweighted graphs of small diameter, a distributed implementation of BFS is able to outperform our algorithm, due to its smaller constant factors, while for unweighted graphs of higher diameter we obtained results consistent with those reported in this section for the weighted `USA` dataset.

Author Contributions: Conceptualization, M.C., A.P., G.P. and E.U.; methodology, M.C., A.P., G.P. and E.U.; software, M.C., A.P., G.P. and E.U.; validation, M.C., A.P., G.P. and E.U.; formal analysis, M.C., A.P., G.P. and E.U.; investigation, M.C., A.P., G.P. and E.U.; resources, M.C., A.P., G.P. and E.U.; data curation, M.C., A.P., G.P. and E.U.; writing–original draft preparation, M.C., A.P., G.P. and E.U.; writing–review and editing, M.C., A.P., G.P. and E.U.; visualization, M.C., A.P., G.P. and E.U.; funding acquisition, M.C., A.P., G.P. and E.U. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded in part by MIUR, the Italian Ministry of Education, University and Research, under PRIN Project n. 20174LF3T8 AHeAD (Efficient Algorithms for HArnessing Networked Data), and grant L. 232 (Dipartimenti di Eccellenza), and by the University of Padova under project "SID 2020: Resource-Allocation Tradeoffs for Dynamic and Extreme Data", and by NSF awards CCF 1740741 and IIS 1813444.

Acknowledgments: The authors wish to thank the anonymous referees for their constructive criticisms which helped improve the quality of the paper.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Barabasi, A.L. *Network Science*; Cambridge University Press: Cambridge, UK, 2016.
2. Roditty, L.; Williams, V.V. Fast approximation algorithms for the diameter and radius of sparse graphs. In Proceedings of the Symposium on Theory of Computing, Palo Alto, CA, USA, 1–4 June 2013; pp. 515–524.
3. Pettie, S.; Ramachandran, V. Computing Shortest Paths with Comparisons and Additions. In Proceedings of the Symposium on Discrete Algorithms, San Francisco, CA, USA, 6–8 January 2002; pp. 267–276.
4. Williams, V.V. Multiplying matrices faster than Coppersmith-Winograd. In Proceedings of the Symposium on Theory of Computing, New York, NY, USA, 19–22 May 2012; pp. 887–898.
5. Aho, A.V.; Hopcroft, J.E.; Ullman, J.D. *The Design and Analysis of Computer Algorithms*; Addison-Wesley: Reading, MA, USA, 1974.
6. Magnien, C.; Latapy, M.; Habib, M. Fast computation of empirically tight bounds for the diameter of massive graphs. *J. Exp. Algorithmics* **2009**, *13*, 10. [CrossRef]
7. Crescenzi, P.; Grossi, R.; Habib, M.; Lanzi, L.; Marino, A. On computing the diameter of real-world undirected graphs. *Theory Comput.* **2013**, *514*, 84–95. [CrossRef]

8. Crescenzi, P.; Grossi, R.; Lanzi, L.; Marino, A. On Computing the Diameter of Real-World Directed (Weighted) Graphs. In Proceedings of the Symposium on Experimental Algorithms, Bordeaux, France, 7–9 June 2012; pp. 99–110.
9. Faloutsos, M.; Faloutsos, P.; Faloutsos, C. On Power-law Relationships of the Internet Topology. In Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Stockholm, Sweden, 30 August–3 September 1999; pp. 251–262.
10. Palmer, C.R.; Gibbons, P.B.; Faloutsos, C. ANF: A Fast and Scalable Tool for Data Mining in Massive Graphs. In Proceedings of the International Conference on Knowledge Discovery and Data Mining, Edmonton, AB, Canada, 23–26 July 2002; pp. 81–90.
11. Flajolet, P.; Martin, G. Probabilistic Counting. In Proceedings of the Symposium on Foundations of Computer Science, Tucson, AZ, USA, 7–9 November 1983; pp. 76–82.
12. Boldi, P.; Rosa, M.; Vigna, S. HyperANF: Approximating the Neighbourhood Function of Very Large Graphs on a Budget. In Proceedings of the International Conference on World Wide Web, Hyderabad, India, 28 March–1 April 2011; pp. 625–634.
13. Flajolet, P.; Fusy, É.; Gandouet, O.; Meunier, F. HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm. In Proceedings of the 2007 Conference on Analysis of Algorithms (AofA'07), Juan des Pins, France, 17–22 June 2007.
14. Boldi, P.; Vigna, S. In-Core Computation of Geometric Centralities with HyperBall: A Hundred Billion Nodes and Beyond. In Proceedings of the Workshop of the International Conference on Data Mining, Dallas, TX, USA, 7–10 December 2013; pp. 621–628.
15. Kang, U.; Tsourakakis, C.E.; Appel, A.P.; Faloutsos, C.; Leskovec, J. HADI: Mining Radii of Large Graphs. *ACM Trans. Knowl. Discov. Data* **2011**, *5*, 8:1–8:24. [CrossRef]
16. Meyer, U. On Trade-Offs in External-Memory Diameter-Approximation. In Proceedings of the Scandinavian Workshop on Algorithm Theory, Gothenburg, Sweden, 2–4 July 2008; Volume 5124, pp. 426–436.
17. Miller, G.L.; Peng, R.; Xu, S.C. Parallel graph decompositions using random shifts. In Proceedings of the Symposium on Parallelism in Algorithms and Architectures, Montreal, QC, Canada, 23–25 July 2013; pp. 196–203.
18. Shun, J.; Dhulipala, L.; Blelloch, G.E. A simple and practical linear-work parallel algorithm for connectivity. In Proceedings of the Symposium on Parallelism in Algorithms and Architectures, Prague, Czech Republic, 23–25 June 2014; pp. 143–153.
19. Cohen, E. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *J. ACM* **2000**, *47*, 132–166. [CrossRef]
20. Meyer, U.; Sanders, P. Δ-stepping: A parallelizable shortest path algorithm. *J. Algorithms* **2003**, *49*, 114 – 152. [CrossRef]
21. Ceccarello, M.; Pietracaprina, A.; Pucci, G.; Upfal, E. Space and Time Efficient Parallel Graph Decomposition, Clustering, and Diameter Approximation. In Proceedings of the Symposium on Parallelism in Algorithms and Architectures, Portland, OR, USA, 13–15 June 2015; pp. 182–191.
22. Ceccarello, M.; Pietracaprina, A.; Pucci, G.; Upfal, E. A Practical Parallel Algorithm for Diameter Approximation of Massive Weighted Graphs. In Proceedings of the International Parallel and Distributed Processing Symposium, Chicago, IL, USA, 23–27 May 2016; pp. 12–21.
23. Abraham, I.; Gavoille, C.; Goldberg, A.V.; Malkhi, D. Routing in Networks with Low Doubling Dimension. In Proceedings of the International Conference on Distributed Computing Systems, Lisboa, Portugal, 4–7 July 2006; p. 75.
24. Murray, D.G.; McSherry, F.; Isaacs, R.; Isard, M.; Barham, P.; Abadi, M. Naiad: A Timely Dataflow System. In Proceedings of the Symposium on Operating Systems Principles, Farmington, PA, USA, 3–6 November 2013; pp. 439–455.
25. Ceccarello, M.; Pietracaprina, A.; Pucci, G. Solving k-center Clustering (with Outliers) in MapReduce and Streaming, almost as Accurately as Sequentially. *Proc. VLDB Endow.* **2019**, *12*, 766–778. [CrossRef]
26. Mazzetto, A.; Pietracaprina, A.; Pucci, G. Accurate MapReduce Algorithms for k-Median and k-Means in General Metric Spaces. In Proceedings of the International Symposium on Algorithms and Computation, Shanghai, China, 8–11 December 2019; pp. 34:1–34:16.

27. Ceccarello, M.; Pietracaprina, A.; Pucci, G.; Upfal, E. MapReduce and Streaming Algorithms for Diversity Maximization in Metric Spaces of Bounded Doubling Dimension. *Proc. VLDB Endow.* **2017**, *10*, 469–480. [CrossRef]
28. Ceccarello, M.; Pietracaprina, A.; Pucci, G. Fast Coreset-based Diversity Maximization under Matroid Constraints. In Proceedings of the International Conference on Web Search and Data Mining, Marina Del Rey, CA, USA, 5–9 February 2018; pp. 81–89.
29. Dean, J.; Ghemawat, S. MapReduce: Simplified data processing on large clusters. *Commun. ACM* **2008**, *51*, 107–113. [CrossRef]
30. Pietracaprina, A.; Pucci, G.; Riondato, M.; Silvestri, F.; Upfal, E. Space-round tradeoffs for MapReduce computations. In Proceedings of the International Conference on Supercomputing, Venice, Italy, 25–29 June 2012; pp. 235–244.
31. Mitzenmacher, M.; Upfal, E. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*; Cambridge University Press: Cambridge, UK, 2005.
32. Kendall, D.G. The genealogy of genealogy branching processes before (and after) 1873. *Bull. Lond. Math. Soc.* **1975**, *7*, 225–253. [CrossRef]
33. Van Der Hofstad, R. *Random Graphs and Complex Networks*; Cambridge University Press: Cambridge, UK, 2016.
34. Dwass, M. The total progeny in a branching process and a related random walk. *J. Appl. Probab.* **1969**, *6*, 682–686. [CrossRef]
35. Boldi, P.; Vigna, S. The WebGraph Framework I: Compression Techniques. In Proceedings of the International Conference on World Wide Web, New York, NY, USA, 17–20 May 2004; pp. 595–601.
36. Boldi, P.; Rosa, M.; Santini, M.; Vigna, S. Layered Label Propagation: A MultiResolution Coordinate-Free Ordering for Compressing Social Networks. In Proceedings of the International Conference on World Wide Web, Hyderabad, India, 28 March–1 April 2011; pp. 587–596.