# A Practical Parallel Algorithm for Diameter Approximation of Massive Weighted Graphs

Matteo Ceccarello, Andrea Pietracaprina, Geppino Pucci
*Department of Information Engineering*
*University of Padova*
*Padova, Italy*
{ceccarel,capri,geppo}@dei.unipd.it

Eli Upfal
*Department of Computer Science,*
*Brown University,*
*Providence, RI USA*
eli_upfal@brown.edu

***Abstract*—We present a space and time efficient practical parallel algorithm for approximating the diameter of massive weighted undirected graphs on distributed platforms supporting a MapReduce-like abstraction. The core of the algorithm is a weighted graph decomposition strategy generating disjoint clusters of bounded weighted radius. Theoretically, our algorithm uses linear space and yields a polylogarithmic approximation guarantee; moreover, for important practical classes of graphs, it runs in a number of rounds asymptotically smaller than those required by the natural approximation provided by the state-of-the-art $\Delta$-stepping SSSP algorithm, which is its only practical linear-space competitor in the aforementioned computational scenario. We complement our theoretical findings with an extensive experimental analysis on large benchmark graphs, which demonstrates that our algorithm attains substantial improvements on a number of key performance indicators with respect to the aforementioned competitor, while featuring a similar approximation ratio (a small constant less than 1.4, as opposed to the polylogarithmic theoretical bound).**

***Keywords*-Graph Analytics; Parallel Graph Algorithms; Weighted Graph Decomposition; Weighted Diameter Approximation; MapReduce**

## I. Introduction

The analysis of very large (typically sparse) weighted 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. 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. For this primitive, which is computationally intensive, resorting to parallelism is inevitable as the graph size grows. Unfortunately, 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 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.

*Previous work.* For general graphs with arbitrary weights, the only known approach for the exact diameter computation requires the solution of the All-Pairs Shortest Paths (APSP) problem, but all known APSP algorithms are characterized by space and/or time requirements which make them impractical for very large graphs. For unweighted graphs, the HyperANF algorithm by [1] computes a tight approximation to the eccentricity of each node (i.e., its maximum distance from every other node), which in turn yields a tight approximation to the graph diameter. HyperANF allows an efficient multithreaded implementation and runs fast on a shared-memory platform. However, it has a critical path equal to the graph diameter and requires a (small) non constant memory blow-up. Also, the algorithm cannot be adapted to deal with weighted graphs. For these reasons, HyperANF is not a viable competitor for the objectives of the present work.

We observe that for connected undirected graphs an upper bound to the diameter within a factor two can be easily computed by solving an instance of the Single-Source Shortest Path (SSSP) problem starting from an arbitrary node. A number of recent works have explored the issue of reducing the approximation factor by running several SSSP instances from carefully selected nodes [2], [3], [4]. While in practice a few such instances suffice to obtain a very good approximation, worst-case approximation guarantees less than two are obtained at the expense of running a nonconstant number of SSSP instances. In fact, there is strong theoretical evidence of the difficulty of approximating efficiently the diameter within a factor less than 3/2 (see [3] and references therein).

A number of parallel SSSP algorithms have been devised in the last two decades [5], [6], [7] (a more extensive

This work was supported, in part, by MIUR of Italy under project AMANDA, and by the University of Padova under project CPDA152255/15. Eli Upfal's work was supported in part by NSF awards IIS-1016648 and IIS-124758.

account of the literature can be found in [7]). To the best of our knowledge, the most practical one is the $\Delta$-*stepping* PRAM algorithm proposed in [7]. The algorithm uses a design parameter $\Delta$ to trade off parallel time and work. In a nutshell, the role of $\Delta$ is to stagger the computation of the SSSP tree into zones of bounded depth $\Delta$. Small values of $\Delta$ reduce work at the expense of a longer critical path (i.e., parallel time) with the algorithm approaching the behaviour of Dijkstra's algorithm; the reverse tradeoff is obtained when $\Delta$ increases, approaching in this case the behaviour of Bellman-Ford's algorithm. It can be shown that, irrespective of $\Delta$, if linear space is required, then the algorithm's parallel time is bounded from below by the unweighted diameter of the input graph. In fact, in [7] the authors show that smaller parallel time can be achieved by introducing suitable shortcut edges connecting all node pairs at distance at most $\Delta$, which, however, have the potential of resulting into superlinear space complexity, especially for sparse graphs.

In [8] we devised a parallel $O\left(\log^3 n\right)$-approximation algorithm for computing the diameter of unweighted undirected graphs. The algorithm first determines a partition of the graph into $k = o(n)$ disjoint clusters through a decomposition strategy based on growing the clusters from batches of centers progressively selected from yet uncovered nodes, which yields a provably small cluster radius. The diameter of the graph is then obtained through the diameter of a suitable quotient graph associated with the decomposition. The algorithm can be efficiently implemented in a MapReduce-like environment yielding good approximation quality in practice. A similar clustering-based approach for diameter estimation had been introduced in [9] in the external-memory setting. We remark that no analytical guarantees would be provided by the weight-oblivious execution of these algorithms on a weighted graph since, for a given topology, the system of shortest paths may radically change once weights are introduced.

*Our contribution.* In this paper we address the more challenging scenario of weighted undirected graphs, and devise a diameter approximation strategy for distributed machines supporting a MapReduce-like abstraction, with provable theoretical guarantees, providing extensive experimental evidence of its efficiency and effectiveness.

Our algorithm adopts a cluster-based strategy similar to the one used in [8] for the unweighted case. The main difficulty in the weighted case stems from the cluster growth process: in the unweighted case we included all nodes in the frontier of all active clusters at each round; in the weighted case we need to avoid traversing heavy edges, which would increase the cluster radius and have an unpredictable impact on the approximation quality. On the other hand, a high rate of cluster growth is crucial to enforce a small round complexity. In order to tackle these conflicting goals, we combine the following three main ingredients: (1) the progressive cluster-growing strategy of [8]; (2) an upper bound $\Delta$ on the weight of the paths along which clusters are grown (an idea inspired by the $\Delta$-stepping algorithm); and (3) a doubling strategy to guess a quasi-optimal value of $\Delta$.

Analytically, we prove that with high probability our algorithm attains an $O\left(\log^3 n\right)$ approximation ratio using a number of rounds which is $O\left(\ell_R \log n\right)$, where $R$ is the maximum weighted radius of a clusters and $\ell_R$ is the number of edges required to connect any pair of nodes at distance at most $R$. Both $R$ and $\ell_R$ are nonincreasing functions of the number $k$ of clusters. To obtain the best round complexity, $k$ can be chosen as the maximum value for which the diameter of the quotient graph can be computed efficiently in a single processor's local memory in one round. The total memory space used by the algorithm is linear in the input graph size. We also show that on graphs of bounded doubling dimension [10] (an important family including, for example, multidimensional arrays), under random edge weights and for some positive constants $\delta < \epsilon < 1$, if the local memory available to each processor is $\Theta\left(n^\epsilon\right)$ then the round complexity of our algorithm can be made asymptotically smaller than the unweighted diameter of the graph by a factor $\Theta\left(n^\delta\right)$, outperforming by at least that factor the approximation strategy based on $\Delta$-stepping, when linear space is required.

We complement the analytical results with an extensive set of experiments on several large benchmark graphs (of up to about one billion nodes and several billion edges), both real and synthetic, running on a 16-node cluster where the Spark engine is employed to provide a MapReduce environment. For all graphs, our algorithm attains an approximation ratio never exceeding 1.4, which is far less than the theoretical upper bound and comparable with the one ensured by the SSSP-based approach. Compared to an implementation of $\Delta$-stepping in Spark, our algorithm is up to two orders of magnitude faster. The better performance of our algorithm is also substantiated by more implementation-independent metrics such as the number of rounds and the aggregate work counted as number of node updates and messages generated. We performed scalability tests which, even in a somewhat small experimental testbed, demonstrate that our algorithmic strategy has the potential to exploit a much higher degree of parallelism, thus affording the analysis of much larger graphs.

*Paper organization.* Section II introduces some basic terminology and defines our reference machine model. Section III illustrates the graph decomposition at the core of the diameter approximation algorithm, which is described in the subsequent Section IV. The experimental analysis is presented in Section V.

## II. Preliminaries

Let $G = (V, E, w)$ be a connected undirected weighted graph with $n$ nodes (set $V$), $m$ edges (set $E$), and a function

$w$ which assigns a positive integral weight $w(e)$ to each edge $e \in E$. We make the reasonable assumption that the edge weights are polynomial in $n$. (Our results generalize to real positive weights with ratio between maximum and minimum weight polynomial in $n$.) The *distance* between two nodes $u, v \in V$, for short $\mathrm{dist}(u, v)$, is the weight of a minimum-weight path between $u, v \in V$. The *diameter* of the graph, which we denote by $\Phi(G)$, is the maximum distance between any two nodes. We remark that while for convenience in the theoretical analysis we consider only connected graphs, our results extend straightforwardly to disconnected graphs, by defining the diameter as the largest distance between any two nodes in the same connected component.

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

An important metric impacting the complexity of our algorithms is the number of edges required to connect nodes with minimum-weight paths. More formally, for a given distance parameter $\Delta$, we define $\ell_\Delta$ to be the minimum value such that for any two nodes $u, v \in V$ with $\mathrm{dist}(u, v) \leq \Delta$ there is a minimum-weight path between $u$ and $v$ with at most $\ell_\Delta$ edges. It is easy to see that $\ell_\Delta$ is a nondecreasing function of $\Delta$ and that for any constant $c > 0$, $\ell_{c\Delta} = \Theta(\ell_\Delta)$.

The performance of the algorithms presented in this paper will be analyzed considering their distributed implementation on the *MR model* of [11]. This model provides a rigorous computational framework based on the popular MapReduce paradigm [12], which is suitable for large-scale data processing on clusters of loosely-coupled commodity servers. Similar models have been recently proposed in [13], [14]. An MR algorithm executes as a sequence of *rounds* where, in a round, a multiset $X$ of key-value pairs is transformed into a new multiset $Y$ of pairs by applying a given reducer function (simply called *reducer*) independently to each subset of pairs of $X$ having the same key. The model features two parameters $M_T$ and $M_L$, where $M_T$ is the total memory available to the computation, and $M_L$ is the maximum amount of memory locally available to each reducer. We use $\mathrm{MR}(M_T, M_L)$ to denote a given instance of the model. The complexity of an $\mathrm{MR}(M_T, M_L)$ algorithm is defined as the number of rounds executed in the worst case, and it is expressed as a function of the input size and of $M_T$ and $M_L$. In the big data realm, practical $\mathrm{MR}(M_T, M_L)$ algorithms should use total space linear in the input size and significantly sublinear local space, while minimizing the round complexity.

The following fact is proven in [14], [11].

**Fact 1.** *The sorting and (segmented) prefix-sum primitives for inputs of size $n$ can be performed in $O\left(\log_{M_L} n\right)$ rounds in MR$(M_T, M_L)$ with $M_T = \Theta(n)$.*

## III. Parallel graph decomposition

In this section we present a parallel algorithm to partition a connected undirected weighted graph $G = (V, E, w)$ into clusters of small radius. The algorithm generalizes the one presented in [8] for the unweighted case. Specifically, 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 challenge in the weighted case 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 approximation quality. In other words, unlike the strategy in [8] for the unweighted case, we cannot afford to boldly grow a cluster by adding all nodes connected to its frontier since some of these additions may entail heavy edges. To tackle this challenge, we use ideas akin to those employed in the $\Delta$-stepping parallel SSSP algorithm proposed in [7]. In particular, we limit a cluster's growth by imposing a threshold $\Delta$ on its radius. Unlike the $\Delta$-stepping algorithm, however, this threshold is not fixed a priori but automatically tuned to a quasi-optimal value during the clustering process.

Before presenting the algorithm, we introduce some technical details. Let $\Delta$ be an integral parameter which we use as a guess for the radius of the clustering. As in [7], we call an edge *light* if its weight is $\leq \Delta$, and *heavy* otherwise. With each node $u \in V$ the algorithm maintains 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 chosen as a cluster center, then $c_u = u$ and $d_u = 0$. The algorithm repeatedly applies, in parallel, edge relaxations of the kind used in the classical Bellman-Ford's algorithm. More precisely, we define the following $\Delta$-*growing step*: for each node $u$ with $d_u < \Delta$ and for each light edge $(u, v)$, in parallel, if $d_u + w(u, v) \leq \Delta$ 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 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, is performed.

Suppose that a sequence of $\Delta$-growing steps is performed starting from some set of centers $X \subseteq V$. After the execution of these steps, we can contract the graph as follows (Procedure `Contract`). For each center $c \in X$, all nodes $u \in V$ with $c_u = c$ are removed, except $c$ itself. For each edge $(u, v)$: 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; and if $c_u$ is defined and $c_v$ is undefined, the edge is replaced by a new edge $(c_u, v)$ of weight $w(u, v)$.

Algorithm CLUSTER$(G, \tau)$, whose pseudocode is given in Algorithm 1, 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 $\Delta$-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 $\Delta$. 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, geometrically increasing values of $\Delta$, starting from a suitable initial value, are guessed until the coverage goal can be attained. (The sequences of $\Delta$-growing steps for the various guesses of $\Delta$ are performed in the inner while loop through Procedure PartialGrowth.) 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 $\Delta$ to the clustering radius. We will show below that the largest guess for $\Delta$ will be $O(R_G(\tau))$, 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 $\Delta_{\text{end}}$ denote the value of $\Delta$ at the end of the execution of CLUSTER$(G, \tau)$. In the following lemma, we show that with high probability $\Delta_{\text{end}}$ does not exceed $R_G(\tau)$ by more than a constant factor.

**Lemma 1.** *$\Delta_{end} = O(R_G(\tau))$, 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 $|V_i - C_i| \geq 8\tau \log n$. 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 $2R_G(\tau)$ traversing only uncovered nodes.

Let $\bar{C}$ be a $\tau$-clustering of the whole graph with optimal radius $r(\bar{C}) = R_G(\tau)$. 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 $\gamma$ in the probability for center selection, we have that with probability $\geq (1 - 1/n^2)$ 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 $w \leq 2R_G(\tau)$, there must be a path in $G_i$ from 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 $\leq w$ and traverses only unconvered nodes. The desired property follows by applying the union bound over all large $\bar{C}$-clusters.

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

The following theorem states the main result of this section.

**Theorem 1.** *Let $\tau$ be a positive integer. With high probability,* CLUSTER$(G, \tau)$ *returns an $O(\tau \log^2 n)$-clustering of radius $O(R_G(\tau) \log n)$ by performing $O(\ell_{R_G(\tau)} \log n)$ $\Delta$-growing steps, with $\Delta = O(R_G(\tau))$.*

**Algorithm 1:** CLUSTER$(G, \tau)$

$\Delta \leftarrow \min\{w(u, v) : (u, v) \in E\}$; $\gamma \leftarrow 4 \ln 2$
$C_1 \leftarrow \emptyset$ /*current set of cluster centers */
$G_1(V_1, E_1) \leftarrow G(V, E)$
$i \leftarrow 1$
**while** $|V_i - C_i| \geq 8\tau \log n$ **do**
  Select $v \in V_i - C_i$ as a new center independently with probability $\frac{(\gamma \tau \log n)}{|V_i - C_i|}$
  X $\leftarrow C_i \cup \{$newly selected centers$\}$
  **foreach** $u \in V_i$ **do**
    **if** $u \in X$ **then** $(c_u, d_u) \leftarrow (u, 0)$ **else** $(c_u, d_u) \leftarrow$ (nil, $\infty$)
  $V' \leftarrow \emptyset$
  **while** $|V'| < |V_i - C_i|/2$ **do**
    PartialGrowth$(G_i, \Delta, C_i)$
    $V' \leftarrow \{u \in V_i - C_i : d_u \leq \Delta\}$
    **if** $|V'| < |V_i - C_i|/2$ **then** $\Delta \leftarrow 2\Delta$
  Assign each $u \in V'$ to the cluster centered at $c_u$
  $G_{i+1}(V_{i+1}, E_{i+1}) \leftarrow$ Contract$(G_i)$
  $C_{i+1} \leftarrow X$
  $i \leftarrow i + 1$
Assign each $u \in V_i - C_i$ to a new singleton cluster centered at $u$
/* $V_i$ *is the final set of cluster centers* */

**Procedure** PartialGrowth$(\bar{G}, \Delta, C)$
**repeat**
  perform a $\Delta$-growing step on $\bar{G}$
  $V' \leftarrow \{u \in V(\bar{G}) - C : d_u \leq \Delta\}$
**until** (no state is updated) or ($|V'| \geq |V(\bar{G}) - C|/2$)

*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 $\log n + \log \Delta_{\text{end}}$, and this number is $O(\log n)$ with high probability by virtue of Lemma 1 and the assumption on the polynomiality of the edge weights. The bound follows by observing that each such iteration increases the radius of cluster by an additive term at most $\Delta_{\text{end}} = O(R_G(\tau))$.

Finally, observe that in every iteration of the inner **while** loop the number of $\Delta$-growing steps executed by procedure `PartialGrowth` is at most $\ell_\Delta \leq \ell_{\Delta_{\text{end}}}$ since, by the properties of edge relaxations, after those many growing steps all nodes at distance less than $\Delta$ 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 $\Delta$-growing steps will be $O\left(\ell_{\Delta_{\text{end}}} \log n\right) = O\left(\ell_{R_G(\tau)} \log n\right)$, with high probability. ∎

## IV. Diameter Approximation

In this section, we present an algorithm to estimate the diameter of a weighted graph as a function of the diameter of a suitable (much smaller) quotient graph of a clustering obtained through a refined version of the strategy devised in the previous section. As it will be clarified by the analysis, the refinement is introduced 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`$(G, \tau)$, whose pseudocode is given in Algorithm 2, builds the required clustering by first computing the radius $R_{\text{CL}}(\tau)$ of the clustering returned by `CLUSTER`$(G, \tau)$ and then executing $\log n$ iterations where uncovered nodes are selected as new cluster centers with probability doubling at each iteration. In the $i$-th iteration, both previous and new clusters are grown using $2R_{\text{CL}}(\tau)$-growing steps until *all* uncovered nodes at distance at most $2R_{\text{CL}}(\tau)$ from them are reached (Procedure `PartialGrowth2`). At the end of the iteration, the graph is contracted using Procedure Contract2, which is similar to Procedure Contract used in `CLUSTER` with the only difference that each original edge $(u, v)$ of weight $w(u, v) \leq 2R_{\text{CL}}(\tau)$ and such that $c_u$ is defined and $c_v$ is undefined, is replaced by a new edge $(c_u, v)$ with rescaled weight $d_u + w(u, v) - 2R_{\text{CL}}(\tau)$. (In fact, original edges of weight greater than $2R_{\text{CL}}(\tau)$ are never used by `CLUSTER2`.)

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

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

**Algorithm 2:** `CLUSTER2`$(G, \tau)$

Let $R_{\text{CL}}(\tau)$ be the radius of the clustering returned by `CLUSTER`$(G, \tau)$
$C_1 \leftarrow \emptyset$ */* (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** $(c_u, d_u) \leftarrow (\text{nil}, \infty)$
  `PartialGrowth2`$(G_i, 2R_{\text{CL}}(\tau))$
  $G_{i+1}(V_{i+1}, E_{i+1}) \leftarrow$ Contract2$(G_i)$
  $C_{i+1} \leftarrow X$

**Procedure** `PartialGrowth2`$(\bar{G}, \Delta)$
**repeat** perform a $\Delta$-growing step on $\bar{G}$ **until** no state is updated

*Proof:* By Theorem 1 we know that with high probability the invocation of `CLUSTER`$(G, \tau)$ at the beginning of the algorithm computes a $K$-clustering, for some $\tau \leq K = O\left(\tau \log^2 n\right)$, of radius $R_{\text{CL}}(\tau) = O(R_G(\tau) \log n)$. In what follows, we condition on this event. Then, the bounds on the number of growing steps and on $R_{\text{CL2}}(\tau)$ are straightforward. For $\gamma = 4/\log_2 e$, define $H$ as the smallest integer such that $2^H/n \geq (\gamma K \log n)/n$, 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 $1 \leq i \leq t$, define the event $E_i$ ="at the beginning of Iteration $H + i$, $|V_{H+i} - C_{H+i}| \leq n/2^{i-1}$". 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 $1 \leq i < t$, 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 of Iteration $H + i$, the number of nodes in $V_{H+i} - C_{H+i}$ is at most $n/2^{i-1}$. Clearly, if $|V_{H+i} - C_{H+i}| \leq n/2^i$ then $E_{i+1}$ trivially holds with probability one. Thus, we consider only the case

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

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 $R_G(K)$ and observe that $R_G(K) \leq R_{\texttt{CL}}(\tau)$. 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 $\gamma$, it is easy to argue that with probability $\geq (1 - 1/n)$ 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 $w \leq R_{\texttt{CL}}(\tau)$, hence there must be a path in $G_i$ from 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 $H + i$, $1 \leq i \leq t$, $O(K \log n)$ 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 $\tau$, 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 *per se*. 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 $2R_{\texttt{CL}}(\tau)$. (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 $\lceil d/2R_{\texttt{CL}}(\tau) \rceil$ 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 CLUSTER2 can be employed 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 $R_{\texttt{CL2}}(\tau)$. 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 [9], we define the weighted quotient 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, it is sufficient to 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 $\Phi_{\text{approx}}(G) = \Phi(G_C) + 2R_{\texttt{CL2}}(\tau)$. It is easy to see that our estimate is conservative, that is, $\Phi_{\text{approx}}(G) \geq \Phi(G)$. We have:

**Theorem 2.** *$\Phi_{\text{approx}}(G) = O\left(\Phi(G) \log^3 n\right)$, with high probability.*

*Proof:* Since $R_{\texttt{CL2}}(\tau) = O\left(R_G(\tau) \log^2 n\right)$ (by Lemma 2), and $R_G(\tau) = O(\Phi(G))$, we have that $R_{\texttt{CL2}}(\tau) = O\left(\Phi(G) \log^2 n\right)$. In order to show that $\Phi(G_C) = O\left(\Phi(G) \log^3 n\right)$, let us fix an arbitrary pair of cluster centers and an arbitrary minimum-weight path $\pi$ between them in $G$, and let $w_\pi$ be the weight of $\pi$. Let $\pi_C$ be the path of clusters in $G_C$ traversed by $\pi$. We now show that with high probability the weight of $\pi_C$ in $G_C$ is $O\left(\Phi(G) \log^3 n\right)$. The bound on the approximation will then follow by applying the union bound over all pairs of cluster centers. Consider first the case $2R_{\texttt{CL}}(\tau) > \Phi(G)$ (note that this can happen since the clustering yielding the radius $R_{\texttt{CL}}(\tau)$ determined at the beginning of CLUSTER2 is built out of paths using only light edges of weight $O(R_G(\tau))$). 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, $\pi_C$ contains $O(\log n)$ clusters and its weight is $O(w_\pi + R_{\texttt{CL2}}(\tau) \log n) = O\left(\Phi(G) \log^3 n\right)$. Suppose now that $2R_{\texttt{CL}}(\tau) \leq \Phi(G)$. We show that at most $O\left(\lceil w_\pi / R_{\texttt{CL}}(\tau) \rceil \log^2 n\right)$ clusters intersect $\pi$ (i.e., contain nodes of $\pi$), with high probability. It can be seen that $\pi$ can be divided into $O(\lceil w_\pi / R_{\texttt{CL}}(\tau) \rceil)$ subpaths, where each subpath is either an edge of weight $> R_{\texttt{CL}}(\tau)$ or a segment of weight $\leq R_{\texttt{CL}}(\tau)$. 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 $R_{\texttt{CL2}}(\tau)$ from $S$ (i.e., light distance at most $R_{\texttt{CL2}}(\tau)$ from the closest node of $S$). Recall that

$R_{\mathrm{CL2}}(\tau) \leq 2R_{\mathrm{CL}}(\tau)\log n$. 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 $(j-1)R_{\mathrm{CL}}(\tau)$ and $jR_{\mathrm{CL}}(\tau) - 1$, 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 $1 \leq j \leq 2\log n + 1$, 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, $\lceil((j+1)R_{\mathrm{CL}}(\tau)-1)/(2R_{\mathrm{CL}}(\tau))\rceil$ 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 $\lceil(j-1)R_{\mathrm{CL}}(\tau)/(2R_{\mathrm{CL}}(\tau))\rceil$ 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 $\pi$, we have that $O\left(\lceil w_\pi / R_{\mathrm{CL}}(\tau)\rceil \log^2 n\right)$ clusters intersect $\pi$, with high probability. Therefore, the weight of $\pi_C$ is $O\left(w_\pi + R_{\mathrm{CL2}}(\tau)\lceil w_\pi/R_{\mathrm{CL}}(\tau)\rceil \log^2 n\right) = O\left(\Phi(G) + R_{\mathrm{CL2}}(\tau)(\Phi(G)/R_{\mathrm{CL}}(\tau))\log^2 n\right) = O\left(\Phi(G)\log^3 n\right)$. ■

### A. Implementation in the MR$(M_T, M_L)$ model

We now discuss the implementation of the above diameter approximation algorithm in the MR model using overall linear space and show that, for a relevant class of graphs, its round complexity can be made asymptotically smaller than the one required to obtain a 2-approximation through the state-of-the-art SSSP algorithm by [7]. For a given connected weighted graph $G = (V, E, w)$, consider the MR$(M_T, M_L)$ model with total memory $M_T$ linear in the graph size, and local memory $M_L \in O(n^\epsilon)$, for some constant $\epsilon \in (0,1)$. We begin by observing that, regardless of the number of active clusters, a $\Delta$-growing step, for any $\Delta$, can be implemented through a constant number of simple prefix and sorting operations which, by Fact 1, require $O(1)$ rounds on MR$(M_T, M_L)$. By combining this observation with the results of Theorem 1 and Lemma 2, we obtain that CLUSTER2$(G,\tau)$ can be implemented in $O\left(\ell_{R_G(\tau)\log n}\log n\right)$ rounds in MR$(M_T, M_L)$.

For an arbitrary positive constant $\epsilon' < \epsilon$, let $\tau = \lceil n^{\epsilon'}\rceil$ and let $G_C = (V_C, E_C, w_C)$ be the weighted quotient graph associated with the clustering returned by CLUSTER2$(G,\tau)$. The diameter $\Phi(G_C)$ (in fact, a constant approximation to this quantity), and, consequently, the value $\Phi_{\mathrm{approx}}(G)$, can then be computed in $O(1)$ rounds in MR$(M_T, M_L)$ by adopting the same techniques described in [8].

The following theorem summarizes the above discussion.

**Theorem 3.** *Let $G$ be a connected weighted graph with $n$ nodes, $m$ edges and weighted diameter $\Phi(G)$. Also, let $0 < \epsilon' < \epsilon < 1$ be two arbitrary constants, and fix $\tau = \lceil n^{\epsilon'}\rceil$. On the MR$(M_T, M_L)$ model, with $M_G = \Theta(m)$ and $M_L = \Theta(n^\epsilon)$, an upper bound $\Phi_{\mathrm{approx}}(G) = O\left(\Phi(G)\log^3 n\right)$ to the diameter of $G$ can be computed in $O\left(\ell_{R_G(\tau)\log n}\log n\right)$ rounds, with high probability.*

Note that the round complexity depends on the characteristics of the graph and is nonincreasing in the number of clusters as controlled by $\tau$, which is in turn a function of $M_L$. For general graphs, the value $\ell_{R_G(\tau)\log n}\log n$ is $O\left(\ell_{\Phi(G)}\log n\right)$ while the analysis in [7] implies that under the linear-space constraint a natural MR-implementation of $\Delta$-stepping requires $\Omega\left(\ell_{\Phi(G)}\log n\right)$ rounds. Hence, our algorithm cannot be asymptotically slower than $\Delta$-stepping. However, we show below that our algorithm becomes considerably faster for an important class of graphs.

The following definition introduces a concept that a number of recent works have shown to be useful in relating algorithms' performance to graph properties [10].

**Definition 2.** *Consider an undirected graph $G = (V, E)$. The* ball of radius $R$ *centered at node $v$ is the set of nodes reachable through paths of at most $R$ edges from $v$. Also, the* doubling dimension *of $G$ is the smallest integer $b > 0$ such that for any $R > 0$, any ball of radius $2R$ can be covered by at most $2^b$ balls of radius $R$.*

We can specialize the result of Theorem 3 as follows.

**Corollary 1.** *Let $G$ be a connected graph with $n$ nodes, $m$ edges, maximum degree $d \in O(1)$, doubling dimension $b \in O(1)$, and positive integral edge weights chosen uniformly at random from a polynomial range. Denote by $\Phi(G)$ and $\Psi(G)$, respectively, the weighted and unweighted diameter of $G$. Also, let $0 < \epsilon' < \epsilon < 1$ be two arbitrary constants. On the MR$(M_T, M_L)$ model, with $M_G = \Theta(m)$ and $M_L = \Theta(n^\epsilon)$, an upper bound $\Phi_{\mathrm{approx}}(G) = O\left(\Phi(G)\log^3 n\right)$ to the diameter of $G$ can be computed in*

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

*rounds, with high probability.*

*Proof:* 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 $\tau$ disjoint clusters of unweighted radius $\psi = O\left(\lceil \Psi(G)/\tau^{1/b}\rceil\right)$. Letting $W$ be the maximum edge weight, we have that $\psi \cdot W$ upper bounds $R_G(\tau)$. We know that our algorithm computes the diameter approximation in $O\left(\ell_{R_G(\tau)\log n}\log n\right)$ rounds. We will now give an upper bound on $\ell_{R_G(\tau)\log n}$. By using results from the theory of branching processes [15] we can prove that by removing all edges of weight $\geq \lfloor W/d\rfloor$ with high probability the graph

Table I: Benchmark graphs. $\Phi(G)$ is the weighted diameter.

| Graph | $n$ | $m$ | $\Phi(G)$ |
|---|---|---|---|
| roads-USA[0] | 23,947,347 | 29,166,673 | $55.8 \cdot 10^6$ |
| roads-CAL[0] | 1,890,815 | 2,328,872 | $16.4 \cdot 10^6$ |
| livejournal[1]$\star$ | 3,997,962 | 32, 681, 189 | 9.41 |
| twitter$\star$[2] | 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 $S > 1$.

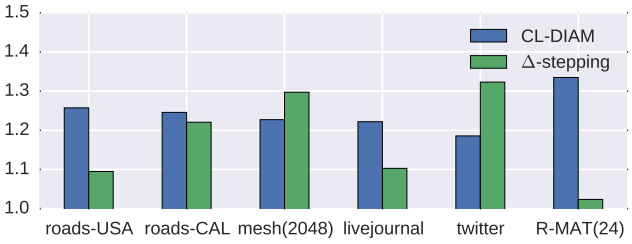


Figure 1: Approximation ratio of CL-DIAM and $\Delta$-stepping.

becomes disconnected and each connected component has $O(\log n)$ nodes. (More details will be provided in the full version of the paper.) As a consequence, with high probability any simple path in $G$ will traverse an edge of weight $\geq \lfloor W/d \rfloor = \Omega(W)$ every $O(\log n)$ nodes. This implies that a path of weight at most $R_G(\tau) \log n = O\left(\left\lceil \frac{\Psi}{\tau^{1/b}} \right\rceil W \log n\right)$ has $\ell_{R_G(\tau)\log n} = O\left(\left\lceil \frac{\Psi}{\tau^{1/b}} \right\rceil \log^2 n\right)$ edges. The theorem follows by setting $\tau = \lceil n^{\epsilon'} \rceil$. ■

For what concerns the comparison with $\Delta$-stepping, the analysis in [7] implies that under the linear-space constraint, for a graph $G$ with random uniform weights a natural MR-implementation of $\Delta$-stepping requires $\Omega(\Psi(G))$ rounds. Thus, by the above corollary, 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 conclude this section by observing that in the case of very skewed graph topologies and/or weight distributions under which the hypotheses of Corollary 1 do not hold, the factor $\ell_{R_G(\tau)\log n}$ in the round complexity of our algorithm can be large, thus reducing the competitive advantage with respect to $\Delta$-stepping. We can somewhat overcome this limitation by imposing an upper limit $O(n/\tau)$ (resp., $O((n/\tau)\log n)$) to the number of growing steps performed in each execution of PartialGrowth within CLUSTER($\tau$) (resp., PartialGrowth2 within CLUSTER2($\tau$)). It can be shown that, in this case, the round complexity of our algorithm, for general graphs, becomes $O\left(\min\{(n/\tau)\log n, \ell_{R_G(\tau)\log n}\} \log n\right)$ at the expenses of an extra $O\left(\lceil \ell_{R_G(\tau)\log n} / ((n/\tau)\log n) \rceil\right)$ factor in the approximation ratio. The argument revolves around the existence of quasi-optimal clusterings of bounded unweighted depth. (More details will be provided in the full version of this paper.)

## V. Experimental Analysis

Our experimental platform is a cluster of 16 nodes, each equipped with a 4-core I7 processor and 18GB RAM, connected by a 10Gbit Ethernet network. Our algorithms are implemented using Apache Spark,[3] a popular framework for big data computations that supports the MapReduce abstraction adopted by our algorithms. The experiments have been run on several graphs whose properties are summarized in Table I and can be classified as follows: a) road networks (roads-USA and roads-CAL), b) social networks (livejournal and twitter), c) synthetic graphs (mesh(S), R-MAT(S), and roads(S), where S is a parameter controlling the size of the graph). The latter class contains artificially generated graphs whose size can be made arbitrarily large and whose topological properties reflect those of the real networks in the first two classes. In particular, R-MAT(S) are graphs with a power-law degree distribution and small diameter included in the Graph500 benchmark set [16], and roads(S) are graphs obtained as the cartesian product of a linear array of $S > 1$ nodes and unit edge weights with roads-USA. Finally, mesh(S) is an $S \times S$ square mesh included since it is a graph of known doubling dimension $b = 2$ for which the results of Corollary 1 hold. All road networks come with original integer weights, while for the other graphs we assigned uniform random edge weights in $(0, 1]$ according to the approach commonly adopted in the literature.

We implemented a simplified version of our diameter approximation algorithm, dubbed CL-DIAM, where, for efficiency, we used CLUSTER, rather than CLUSTER2, for computing the graph decomposition. In fact, CLUSTER2 first runs CLUSTER to obtain an estimate of the radius, and then computes a second decomposition 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 $\Delta$ larger than the minimum edge weight, as was specified in the pseudocode. We observe that by increasing the initial value of $\Delta$, the round complexity improves since less doublings are required before hitting the final value. On the other hand, setting the initial value of $\Delta$ too large may yield a larger cluster radius, possibly incurring a worse diameter approximation. To explore this phenomenon, we

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

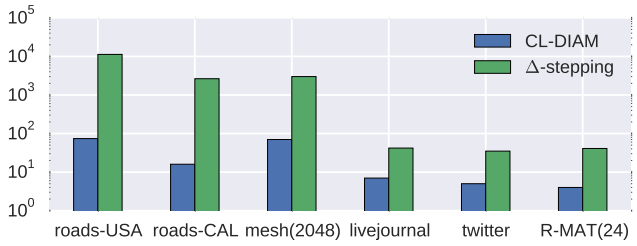


Figure 2: Number of rounds required by CL-DIAM and Δ-stepping. The scale is logarithmic.

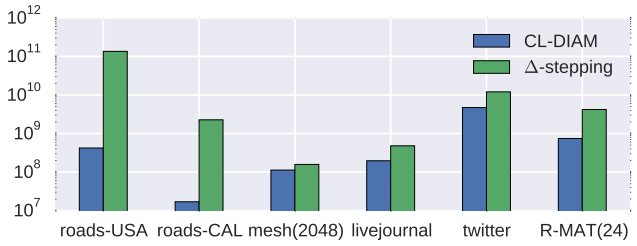


Figure 3: Work performed by CL-DIAM and Δ-stepping. The scale is logarithmic.

experimented on $\texttt{mesh}(S)$ with $S = 2048$ and 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 with 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 to let the algorithm tune itself to the final value $\Delta$ ($= 6.4 \cdot 10^{-5}$); the second configuration started with an inital $\Delta$ equal to the graph diameter ($\approx 2.004$) so that no doubling of $\Delta$ 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 brevity) showed that a good initial guess for $\Delta$ 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 $\Delta$.

Finally, in all of our experiments the parameter $\tau$ was set to yield a number of nodes in the quotient graph $\leq 100,000$. This choice of $\tau$ was made to ensure that, for all instances, the final diameter computation in the quotient graph would not dominate the running time.

The following paragraphs describe in detail the different sets of experiments that we performed.

*Comparison with the SSSP-based approximation.* Recall that an SSSP algorithm can be used to yield a 2-approximation to the diameter 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 is the state of the art for parallel SSSP and is in fact our only practical competitor on weighted graphs. In Δ-stepping, parameter $\Delta$ 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 with several values of $\Delta$, 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, not surprisingly, for all graphs the best value of $\Delta$ was always the one minimizing the number of rounds.

The results of the comparison are summarized in Table II and graphically represented in Figures 1, 2, and 3. In the table we report, for each graph, the diameter 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 quality returned by CL-DIAM on all benchmark graphs, a value 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 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 2.

For what concerns the work, the better performance of CL-DIAM, as shown in Figure 3, is mainly due to the smaller number relaxations with respect to Δ-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 $\Delta$, 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 II 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

Table II: CL-DIAM vs $\Delta$-stepping. 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. The approximation ratio is expressed in terms of a lower bound to the true diameter computed by running the sequential SSSP algorithm multiple times.

| | approximation | | time | | rounds | | work | |
|---|---|---|---|---|---|---|---|---|
| graph | CL-DIAM | $\Delta$-stepping | CL-DIAM | $\Delta$-stepping | CL-DIAM | $\Delta$-stepping | CL-DIAM | $\Delta$-stepping |
| 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 | $7.45 \cdot 10^8$ | $4.20 \cdot 10^9$ |

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 machine 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 have topologies of different nature. The results, omitted for lack of space, report that the running time on both graphs scales linearly with the number of machines used.

Finally, we ran CL-DIAM on R-MAT(29) and roads(32), which are respectively 32 and 57 times larger than R-MAT(24) and roads-USA, used in Table II. The running times were 6218 and 14054 seconds, respectively. This experiment shows that CL-DIAM's performance scales well with the graph size on the same machine configuration. A slight penalty in the running time on large graphs 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.

## REFERENCES


[1] P. Boldi, M. Rosa, and S. Vigna, "HyperANF: approximating the neighbourhood function of very large graphs on a budget," in *Proc. WWW*, 2011, pp. 625–634.

[2] P. Crescenzi, R. Grossi, L. Lanzi, and A. Marino, "On computing the diameter of real-world directed (weighted) graphs," in *Proc. SEA*, 2012, pp. 99–110.

[3] S. Chechik, D. Larkin, L. Roditty, G. Schoenebeck, R.E.Tarjan, and V. Williams, "Better approximation algorithms for the graph diameter," in *Proc. ACM-SIAM SODA*, 2014, pp. 1041–1052.

[4] J. Shun, "An evaluation of parallel eccentricity estimation algorithms on undirected real-world graphs," in *Proc. ACM KDD*, 2015, pp. 1095–1104.

[5] E. Cohen, "Polylog-time and near-linear work approximation scheme for undirected shortest paths," *J. ACM*, vol. 47, no. 1, pp. 132–166, 2000.

[6] P. Klein and S. Subramanian, "A randomized parallel algorithm for single-source shortest paths," *Journal of Algorithms*, vol. 25, no. 2, pp. 205–220, 1997.

[7] U. Meyer and P. Sanders, "$\Delta$-stepping: a parallelizable shortest path algorithm," *Journal of Algorithms*, vol. 49, no. 1, pp. 114–152, 2003.

[8] M. Ceccarello, A. Pietracaprina, G. Pucci, and E. Upfal, "Space and time efficient parallel graph decomposition, clustering, and diameter approximation," in *Proc. ACM-SPAA*, 2015, pp. 182–191.

[9] U. Meyer, "On trade-offs in external-memory diameter-approximation," in *Proc. SWAT*, 2008, pp. 426–436.

[10] I. Abraham, C. Gavoille, A. Goldberg, and D. Malkhi, "Routing in networks with low doubling dimension," in *Proc. IEEE-ICDCS*, 2006.

[11] A. Pietracaprina, G. Pucci, M. Riondato, F. Silvestri, and E. Upfal, "Space-round tradeoffs for MapReduce computations," in *Proc. ACM-ICS*, 2012, pp. 235–244.

[12] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *CACM*, vol. 51, no. 1, pp. 107–113, 2008.

[13] H. Karloff, S. Suri, and S. Vassilvitskii, "A model of computation for mapreduce," in *Proc. ACM-SIAM SODA*, 2010, pp. 938–948.

[14] M. Goodrich, N. Sitchinava, and Q. Zhang, "Sorting, searching, and simulation in the MapReduce framework," in *Proc. ISAAC*, 2011, pp. 374–383.

[15] M. Dwass, "The total progeny in a branching process and a related random walk," *Journal of Applied Probability*, vol. 6, no. 3, pp. 682–686, 1969.

[16] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the graph 500," *Cray Users Group (CUG)*, 2010.